

DL408/D
REV 1



8-bit MCU Applications Manual

8-bit MCU Applications Manual

DL408/D
REV 1




MOTOROLA



MOTOROLA

8-bit MCU Applications Manual

All products are sold on Motorola's Terms & Conditions of Supply. In ordering a product covered by this document the Customer agrees to be bound by those Terms & Conditions and nothing contained in this document constitutes or forms part of a contract (with the exception of the contents of this Notice). A copy of Motorola's Terms & Conditions of Supply is available on request.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

The Customer should ensure that it has the most up to date version of the document by contacting its local Motorola office. This document supersedes any earlier documentation relating to the products referred to herein. The information contained in this document is current at the date of publication. It may subsequently be updated, revised or withdrawn.

Includes literature available at July 1992
All trademarks recognized.

© MOTOROLA INC.
All Rights Reserved
First Edition DL408/D, 1990
DL408/D Rev. 1, 1992

Printed in Great Britain by Tavistock Press (Bedford) Ltd. 5000 8/92

Preface

This compilation of Application Notes, Engineering Bulletins, Design Concepts, etc. was originally published by the European Literature Centre of Motorola Ltd. in Milton Keynes, England, and has subsequently gained worldwide acceptance.

Because of the worldwide popularity of the Application Manuals Series it is important for the reader to take note of the following:

The various Application Notes, Engineering Bulletins, Design Concepts, etc. which are included were developed at Design Centres strategically located throughout the global community and many were originally written to support a local need. Whilst the basic concepts of each of the publications included may have broad global applicability, specific Motorola semiconductor parts may be referred to that are currently available for limited distribution in a specific region and may only be supported by the country of origin of the document in which it is referenced.

Also included in the series for completeness and historical significance are documents that may no longer be available individually because obsolete devices are referenced or perhaps, simply, the original document is out of print. Such items are marked in the Table of Contents, Cross Reference, Abstracts and on the first page of the document with the letters 'HI' to indicate that these documents are included for Historical Information only.

All the Application Notes, Engineering Bulletins, Design Concepts, etc. are included to enhance the user's knowledge and understanding of Motorola's products. However, before attempting to design-in a device referenced in this Series, the user should contact the local Motorola supplier or sales office to confirm product availability and if application support is available.

Thank you.

Other books in this series include:

<i>DL409/D Rev. 1</i>	<i>16/32-bit Applications Manual</i>
<i>DL410/D</i>	<i>Power Applications Manual</i>
<i>DL411/D</i>	<i>Communications Application Manual</i>
<i>DL412/D</i>	<i>Industrial Control Applications Manual</i>
<i>DL413/D</i>	<i>Radio, RF and Video Applications Manual</i>
<i>DL414/D</i>	<i>FET Applications Manual</i>

Contents

	<i>page</i>
Device Cross Reference	9
Abstracts of Applications Documents	13
Applications Documents	
AN427 MC68HC11 EEPROM Error Correction Algorithms in C	21
AN431 Temperature Measurement and Display Using the MC68HC05B4 and the MC14489	33
AN432 128K byte Addressing with the M68HC11	49
AN433 TV On-Screen Display Using the MC68HC05T1	73
AN434 Serial Bootstrap for the RAM and EEPROM1 of the MC68HC05B6	93
AN436 Error Detection and Correction Routines for M68HC05 Devices Containing EEPROM	105
AN440 MC68HC805B6 and MC68HC705B5 Serial/Parallel Programming Module	117
AN441 MC68HC05E0 EPROM Emulator	121
AN442 Driving LCDs with M6805 Microprocessors	153
AN446 MCM2814 Gang-Programmer Using an MC68HC805B6	169
AN448 "FLOF" Teletext using M6805 Microcontrollers	181
AN452 Using the MC68HC11K4 Memory Mapping Logic	217
AN459 A Monitor for the MC68HC05E0	229
AN890 Low Voltage Inhibit (LVI) Capability of the M6805 HMOS Microcomputer Family ^{MI}	265
AN900 Using the M6805 Family On-Chip 8-Bit A/D Converter	285
AN940 Telephone Dialling Techniques Using the MC6805	305
AN974 MC68HC11 Floating-Point Package	323
AN991 Using the Serial Peripheral Interface to Communicate Between Multiple Microcomputers	365
AN1010 MC68HC11 EEPROM Programming from a Personal Computer	385
AN1050 Designing for Electromagnetic Compatibility (EMC) with HCMOS Microcontrollers	399
AN1055 M6805 16-bit Support Macros	423
AN1057 Selecting the Right Microcontroller Unit	465
AN1058 Reducing A/D Errors in Microcontroller Applications	473
AN1060 MC68HC11 Bootstrap Mode	485
AN1064 Use of Stack Simplifies M68HC11 Programming	527
AN1065 Use of the MC68HC68T1 Real-Time Clock with Multiple Time Bases	559
AN1066 Interfacing the MC68HC05C5 SIOP to an I2C Peripheral	567
AN1067 Pulse Generation and Detection with Microcontroller Units	587
AN1091 Low Skew Clock Drivers and their System Design Considerations	613
AN1097 Calibration-Free Pressure Sensor System	619
AN1102 Interfacing Power MOSFETs to Logic Devices	625
AN1120 Basic Servo Loop Motor Control Using the MC68HC05B6 MCU	635
AN1203 A Software Method for Decoding the Output from the MC14497/MC3373 Combination	643
ANE405 Bi-Directional Data Transfer Between MC68HC11 and MC6805L3 Using SPI ^{MI}	649
ANE418 MC68HC805B6 Low-Cost EEPROM Microcomputer Programming Module ^{MI}	659
ANE420 Monitor Program for the MC68HC05B6 Microcomputer Unit ^{MI}	661
EB400 Secure Single Chip Microcomputer Manufacture	683
EB401 SCAM Modules for Smart Cards	691
EB404 "Memories Are Made of This" ... a Look at Memory Considerations for Smart Card Applications	693
EB405 Smart Cards: How to Deal Yourself a Winning Hand	705
EB408 MC68HC705T3 Bootloader	713
Additional Information	723

Device Cross Reference

Device Cross Reference

This quick-reference list indicates where specific components are featured in applications documents reproduced in this Manual.

M68HC05	AN431	MC68HC11A8	AN1067
.....	AN436	MC68HC11A8P1	AN1065
.....	AN442	MC68HC11G5	AN432
.....	AN1203	MC68HC11K4	AN452
M68HC05E0	AN459	MC68HC68T1	AN1065
M68HC11	AN427	MC68HC705B5	AN440
.....	AN432	MC68HC705C8	AN1067
.....	AN1058	MC68HC705T3	EB408
.....	AN1060	MC68HC805B6	AN440
.....	AN1064	AN446
.....	AN1102	ANE418 ^{MI}
.....	AN1203	MC74LS26	AN1102
M6805	AN442	MC3373	AN1203
.....	AN1055	MC6805L3	ANE405 ^{MI}
MC68HC05B4	AN431	MC6805SC01	EB401
MC68HC05B6	AN434	MC6805SC03	EB401
.....	AN1097	MC14489	AN431
.....	AN1120	MC14497	AN1203
.....	ANE418 ^{MI}	MC68705P3	AN940
.....	ANE420 ^{MI}	MC68705R3	AN991
MC68HC05C4	AN991	MC144115	AN441
.....	AN1067	MC144115P	AN442
MC68HC05C5	AN1066	MC145000	AN442
MC68HC05E0	AN441	MC145003	AN442
MC68HC05J1	AN1067	MC145004	AN442
MC68HC05L6	AN442	MCC68HC05SC11	EB400
MC68HC05SC11	EB401	MCC68HC05SC21	EB400
MC68HC05SC21	AN436	MCM60L256	AN441
.....	EB401	MCM2814	AN446
MC68HC05T1	AN433	MPM3004	AN1120
MC68HC05T7	AN448	MPX2000	AN1097
MC68HC11	AN974	MTP3055E	AN1102
.....	AN1010	MTP3055EL	AN1102
.....	ANE405 ^{MI}	PCF8573	AN1066

Abstracts of Applications Documents

Abstracts

AN427 MC68HC11 EEPROM Error Correction Algorithms in C

A modified Hamming code is used to correct one-bit errors and detect two-bit errors in data blocks of up to 11 bits – avoiding the problem of erroneous correction of two-bit errors. The technique is implemented entirely in 'C', and additional functions are provided to program and read MC68HC11 EEPROM using the encoding/decoding algorithms.

AN431 Temperature Measurement and Display Using the MC68HC05B4 and the MC14489

Shows the basic building blocks of a temperature control system based on the M68HC05 B-series MCUs. Software routines provided include Look-Up Table Interpolation, Binary to BCD Conversion, Degrees C to Degrees F Conversion, and the basis of a real-time counter/clock. Uses a thermistor as the sensing element to allow easy interfacing to the A/D converter of the MC68HC05B4, but the software principles are easily adapted to other sensors.

AN432 128K byte Addressing with the M68HC11

The 64K byte direct addressing capability of the M68HC11 family is insufficient for some applications. This note describes two methods of memory paging – one software only, the other hardware plus software – that allow the MCU to address a 1Mbit EPROM (128K bytes) by manipulation of the address lines. The two methods illustrate the concept of paging and the inherent compromises; the technique may be expanded to other memory combinations. Includes full software listings.

AN433 TV On-Screen Display Using the MC68HC05T1

The T-series devices in the M68HC05 MCU Family provide a convenient and cost-effective means of adding On Screen Display capability (OSD) to TVs and VCRs. The MC68HC05T1 is at the centre of the T-series price/performance range, and is used in this example. Full software listings are provided for a ROM-efficient implementation of an 8-row by 16-character display, including Programme Change, Channel Mode, Automatic Search, Analogues and Channel Name.

AN434 Serial Bootstrap for the RAM and EEPROM1 of the MC68HC05B6

The MC68HC05B6 has 256 bytes of on-chip EEPROM, called EEPROM1, which can be used for non-volatile data storage. In many applications EEPROM1 stores a look-up table or system set-up variables – in these cases it is necessary to initialise the memory during system manufacture. The RAM bootstrap program in the 'B6 mask ROM uses a simple protocol in order to save ROM space, and cannot accept the S-records that are the normal assembler output. This note explains how to convert assembler output to the 'B6 bootstrap format, and how to bootstrap data into EEPROM1.

AN436 Error Detection and Correction Routines for M68HC05 Devices Containing EEPROM

Applications based on M68HC05 MCUs increasingly require large amounts of critical data to be stored in the on-chip EEPROM. This note describes 'HC05 software routines which allow stored data to be encoded so that single bit errors in retrieved data may be corrected, and two bit errors detected. The routines use a simple Linear Block Code (Hamming Code) for encoding the stored data. They were written originally for the MC68HC05-SC21 Smart Card MPU, but can be modified easily to run on any 'HC05 MCU with EEPROM.

AN440 MC68HC805B6 and MC68HC705B5 Serial/Parallel Programming Module

The MC68HC05B serial/parallel programmer module allows the user to program MC68HC805B6 and MC68HC705B5 MCUs. This note describes its various operating modes, and gives details of its construction and use. Includes circuit diagram and parts list.

AN441 MC68HC05E0 EPROM Emulator

Unlike other members of the M6805 family, the MC68HC05E0 has no on-chip ROM but can address a full 64K bytes of external memory; the external memory may be ROM, EPROM, RAM and/or additional hardware. This EPROM emulator illustrates a typical use of this type of MCU; it includes a keyboard, LCD, serial communication and 64K of paged RAM. It can replace with RAM the program ROM or EPROM in a target system through a cable connection to the system's EPROM socket, and can be used to debug and modify the target system software. Includes an assembled listing of the emulator control program.

AN442 Driving LCDs with M6805 Microprocessors

The MC68HC05L series of MCUs include circuitry for direct LCD drive. Other MCUs in the M6805 and M68HC05 families have a variety of I/O and display drive capabilities. This comprehensive note describes alternative LCD drive arrangements for applications with different numbers of backplanes and display drive capabilities, including software-based and display driver chip solutions. Circuits and software listings are provided. The techniques apply equally to other MCU families such as the M6801 and M68HC11.

AN446 MCM2814 Gang-Programmer Using an MC68HC805B6

Non-volatile memories (NVM) such as the MCM2814 are widely used in consumer equipment such as television receivers to store semi-permanent, user-defined information. They may also contain data such as optimum sound and picture settings. In a production environment, the initial loading of this data can be achieved quickly by copying an existing NVM. This note describes a programmer based on an MC68HC805B6 which in four

Abstracts (continued)

seconds can fully program eight MCM2814s in parallel and verify them individually.

AN448 "FLOF" Teletext using M6805 Microcontrollers

The "-T" members of Motorola's M68HC05 MCU family provide a cost-effective method of adding On Screen Display (OSD) to TVs and VCRs. This note describes an example of Full Level One Feature (FLOF) Teletext control software written for the MC68HC05T7 to control type 5243 Teletext chips. Around 3K bytes of ROM are used, allowing the code to fit with tuning, OSD and stereo functions into the 7.9K bytes of the MC68HC05T7. The example software includes the Spanish implementation of Packet 26; Packet 26 allows for the substitution of specific characters for a particular country.

AN452 Using the MC68HC11K4 Memory Mapping Logic

The MC68HC11K4 includes memory expansion logic which allows the 64 KByte addressing range of the M68HC11 CPU to be extended to more than 1 MByte. This note discusses the operation of this logic and provides examples of memory maps and possible hardware configurations.

AN459 A Monitor for the MC68HC05E0

Development systems for single-chip MCUs can be complex and relatively expensive. This can dissuade potential users from designing them into new applications. This note describes a simple "entry level" development system suitable for debugging hardware and software for the M6805 family of microprocessors. Includes full descriptions, circuit diagram and a listing of the monitor software.

AN890 Low Voltage Inhibit (LVI) Capability of the M6805 HMOS Microcomputer (MCU) Family

HI

The LVI option provides a cost effective means for the MCU to sense a drop in supply voltage and then shut itself down in well-defined manner. Because the option does not require any additional external parts it provides an overall product cost reduction. The LVI option is provided at the time of manufacture by on-chip circuitry contained in part of the user's ROM pattern. This application note includes an LVI schematic diagram as well as a listing of the monitor and self-check programs.

AN900 Using the M6805 Family On-Chip 8-Bit A/D Converter

Factors which should be considered when using on-chip analog-to-digital (A/D) converters are covered. The pertinent circuit elements and terminology are defined and a self-test hardware/software technique is illustrated. An example on how to manipulate the converted analog data from a temperature sensor is given. It is intended for the digital designer with little or no programming experience.

AN940 Telephone Dialling Techniques Using the MC6805

Intelligent telephones are increasing in popularity - MCUs from the versatile M6805 family make ideal controllers. This demonstration board, based on an MC68705P3 single-chip MCU, shows two cost-effective methods of DTMF and pulse-type dialling. Full hardware schematic and software listings included.

AN974 MC68HC11 Floating-Point Package

While most MC68HC11 applications can be implemented using 16-bit integer precision, certain algorithms may be difficult or impossible without floating-point. This application note details an efficient floating-point package that includes basic trig functions and square root in addition to add, subtract, multiply and divide. It requires just over 2k bytes of memory, with only 10 bytes of page zero RAM in addition to stack RAM.

AN991 Using the Serial Peripheral Interface to Communicate Between Multiple Microcomputers

Communication between multiple processors can be difficult when different types are used. One solution is the SPI, an interface intended for communication between ICs on the same board. It can be implemented in software, allowing communication between two MCUs where one has SPI hardware and the other does not. Costly expansion buses and UARTs are eliminated. The scheme is illustrated with a temperature/time display circuit using an MC68HC05C4 and an MC68705R3.

AN1010 MC68HC11 EEPROM Programming from a Personal Computer

Describes a simple and reliable method of programming the MC68HC11's internal EEPROM (or EEPROM connected to its external bus) by downloading data in Motorola S-record format from a standard personal computer (PC) fitted with a serial communications port. Includes BASIC program for the PC (to Program External EEPROM/RAM, Program Internal EEPROM, or Verify internal or External EEPROM/RAM) and the source listing of MC68HC11 code for downloading to RAM to receive S records.

AN1050 Designing for Electromagnetic Compatibility (EMC) with HCMOS Microcontrollers

As the operating speeds of the latest HCMOS devices increase, the MCU system designer must take more account of the electromagnetic compatibility (EMC) of the finished product. This discussion relates mainly to emission control, but most of the techniques also reduce electromagnetic susceptibility. Subjects include Legal Requirements, RFI Problems, types of radiation, Supply Decoupling, Grounding Techniques and PCB Layouts. Incorporates an article reprint from EMC Technology describing an EMI/RFI diagnostic probe.

Abstracts (continued)

AN1055 M6805 16-bit Support Macros

MCUs from the M6805 family are usually chosen for applications requiring small program memory and low computing power, where their low cost is an important benefit. However they may also be used in more advanced applications by employing the advanced software techniques described here. The examples are suitable for 'black box' operation (they may be used without knowing how they work) and consist of macros and subroutines that support pseudo registers on the '6805, simulating registers and addressing modes available on the M68HC11.

AN1057 Selecting the Right Microcontroller Unit

Selecting the proper MCU for an application is one of the critical decisions which can control the success or failure of the project. There are numerous criteria to consider; many of them are presented here along with the thought processes guiding their selection. The reader should attach an appropriate grading scale before evaluating the total and making the correct decision.

AN1058 Reducing A/D Errors in Microcontroller Applications

The MCU with integrated Analogue to Digital Converter provides a highly cost-effective solution for many mixed analogue/digital applications. However, combining a wide bandwidth ADC system on the same die as a high-speed CPU can lead to noise problems in the analogue measurements. This comprehensive note lays down basic system guidelines for the design phase of an MCU-based product, to avoid ADC problems. Includes an examination of a real-world system.

AN1060 MC68HC11 Bootstrap Mode

The M68HC11 Bootstrap Mode allows a user program to be loaded into internal RAM through the Serial Communications Interface (SCI). In addition to operating normally, this program can do anything a factory test program can do since the protected control bits become accessible; Expanded Mode resources are available because the control bits can be changed by the bootstrap program. Although the basic concepts are simple, some subtle implications of this mode need careful consideration, both to avoid problems and to find useful applications. Includes commented listings for selected M68HC11 bootstrap ROMs.

AN1064 Use of Stack Simplifies M68HC11 Programming

Architectural extensions to the M6800 family built in to the MC68HC11 allow easy manipulation of data on the stack. The CPU uses the stack for subroutine and interrupt return addresses. This note discusses two additional uses – the storage of local variables and subroutine parameter passing – that can simplify programming and debugging. It describes the basic operation of the MC68HC11 stack, the concept of local and

global variables, subroutine parameter passing, and the use of the instruction set to achieve the additional uses. Includes example listings illustrating the techniques.

AN1065 Use of the MC68HC68T1 Real-Time Clock with Multiple Time Bases

The MC68HC68T1 Real Time Clock plus RAM can use a crystal or the 50/60Hz line frequency as its timebase; a Serial Peripheral Interface is provided for communication with a microcomputer. Applications are often line powered during normal operation, using the line frequency as timebase, but must continue to maintain the correct time of day from a crystal source when mains power is lost. The MC68HC68T1 is not capable of switching between the two frequency sources directly, and additional support by the MCU is necessary. This note describes the necessary hardware and software, based on an MC68HC11A8P1 MCU.

AN1066 Interfacing the MC68HC05C5 SIOP to an I²C Peripheral

A standard MCU may not have all the peripherals required in a system on chip. The problem can be solved by interfacing the MCU to off-chip peripherals, ideally using a synchronous serial communication port. Unfortunately these peripherals may not have an interface that is compatible with Motorola's simple synchronous Serial I/O Port (SIOP). This note describes how the SIOP on the MC68HC05C5 can be interfaced to an I²C peripheral, in this case the PCF8573 Clock/Timer. Includes circuit and software listings for a timer/calendar application that can interface with a terminal.

AN1067 Pulse Generation and Detection with Microcontroller Units

MCUs are often required to generate timed output pulses, and to detect and measure input pulses. Output pulses might strobe a display latch, transmit a code or meter an action in a process control system. Input pulses can range from microseconds to hours, and include detecting pushbutton closures, receiving codes or measuring engine rotation. This note describes various methods of generation and detection using several families of Motorola MCUs with differing timer structures. Includes program listings.

AN1091 Low Skew Clock Drivers and their System Design Considerations

With microprocessor-based systems now running at 33MHz and beyond, low-skew clock drivers have become essential – Motorola produces several devices with less than 1ns skew between outputs. Unfortunately, simply plugging one of these high performance clock drivers into a board does not guarantee trouble-free operation. Careful board layout and system noise considerations must also be taken into account.

Abstracts (continued)

AN1097 Calibration-Free Pressure Sensor System

The MPX2000 Series of pressure transducers give an output signal proportional to applied pressure. They are available as both ported and unported assemblies for pressure, vacuum and differential measurement. By using the on-chip A/D converter of the MC68HC05B6 MCU, an accurate, reliable and versatile pressure measurement system can be designed which needs no external calibration.

AN1102 Interfacing Power MOSFETs to Logic Devices

Most popular power MOSFETs need 10 volts of gate drive to support their maximum drain current. This creates problems when attempting to drive from 5 volt logic. The new Logic Level power MOSFETs solve some but not all of the problems. This note discusses easy methods of directly interfacing both types of MOSFET to TTL and CMOS logic, and to microprocessors such as the M68HC11. Discusses a method of calculating switching times, to minimise switching losses, and stresses the significance of logic power supply variations.

AN1120 Basic Servo Loop Motor Control Using the MC68HC05B6 MCU

A Proportional Derivative (PD) closed-loop speed control for a brush motor can be created using four integrated circuits, two opto discretes and less than 200 bytes of code. The use of an MCU in feedback control systems is increasingly commonplace. It is justified when system flexibility is needed, for example to accommodate varying drive motors or to allow wear parameters to be stored in EEPROM. This design is based on an MC68HC05B6 MCU and an MPM3004 power MOSFET H-bridge.

AN1203 A Software Method for Decoding the Output from the MC14497/MC3373 Combination

Infrared communication is now widely used as a simple and effective means of remote control over short distances. A variety of encoding methods is used, including the biphasic scheme implemented by the MC14497, a complete building block for IR data transmission. The MC3373 is a companion receiver chip to the MC14497, providing front-end processing to interface a photo detector to a TTL level. This note describes, with software listings for the MC68HC11 and the MC68HC05, the decoding of the data at the output of the MC3373.

ANE405 Bi-Directional Data Transfer Between MC68HC11 and MC6805L3 Using SPI

The powerful Serial Peripheral Interface available on many Motorola MCUs is implemented in 2 forms (the HCMOS families support only Level 1, Level 2 is implemented only on HMOS processors). Both levels communicate easily with each other, but Level 2 has additional

capabilities including asynchronous communication. This note describes a method of achieving synchronous communication between levels 1 and 2, and explains the on-chip differences in SPI implementation.

ANE418 MC68HC805B6 Low-Cost EEPROM Microcomputer Programming Module

HI

The EEPROM feature of the MC68HC805B6 microcomputer enables the user to emulate the MC68HC05B6 and the MC68HC05B4. This note describes one programming technique for the MC68HC805B6 internal EEPROM, and describes the design of the simple programming module required.

ANE420 Monitor Program for the MC68HC05B6 Microcomputer Unit

HI

A monitor program is available in the mask ROM of a 68HC05B6 MCU (XC68HC05B6FN MONITOR) which allows the user to write and debug small portions of 68HC05B6 code. It is used in conjunction with a monitor circuit module, +5V power supply and RS-232 terminal. This note includes a description of the facilities available from the software, a circuit diagram of the module and a listing of the monitor code.

EB400 Secure Single Chip Microcomputer Manufacture

Security is the fundamental requirement in designing and manufacturing Smart Card MCUs. This Bulletin summarises the purpose and history of Smart Cards, and explains some of the problems of testing devices after manufacture without prejudicing security. The manufacturing process is necessarily different to that of 'normal' MCUs.

EB401 SCAM Modules for Smart Cards

Motorola's SCAM range of assembly modules consists of the Smart Card product family packaged for insertion in ISO standard plastic cards. This Bulletin lists the planned devices and shows the IS7816/2 contact dimensions, locations and connections. All devices conform to all relevant ISO standards.

EB404 "Memories Are Made of This" ... a Look at Memory Considerations for Smart Card Applications

A Smart Card application typically uses many millions of units per year, so unit cost is crucial to its success. This paper discusses some of the issues concerning memory size and type - and their effect on the specification and cost of secure microcomputers - with particular reference to physical size. (11pp)

EB405 Smart Cards: How to Deal Yourself a Winning Hand

An overview of the current Smart Card market and the various types of product on offer. It looks at ways of determining what features must be provided by a suc-

Abstracts (continued)

cessful Smart Card implementation in a given application. Because of the high production volumes, it is essential to choose the optimum product, and to ask the right questions at the start.

EB408 MC68HC705T3 Bootloader

This bootloader for the MC68HC705T3 has four switch-selected modes of operation. In addition to programming and verifying the internal EPROM from an external EPROM, it is also possible to load and execute a program in RAM locations \$0100-\$01FF. A handshake facility is included to allow the external EPROM to be replaced by an intelligent data source and to provide a limited debug capability. Includes circuit diagram and software listing.

Applications Documents

1. The first part of the document discusses the importance of maintaining accurate records of all transactions and activities. It emphasizes the need for transparency and accountability in financial reporting.

2. The second part of the document outlines the various methods and techniques used to collect and analyze data. It highlights the importance of using reliable sources and ensuring the accuracy of the information gathered.

MC68HC11 EEPROM Error Correction Algorithms in C

By Richard Soja
 Motorola Ltd
 East Kilbride
 Glasgow

INTRODUCTION

This application note describes a technique for correcting one bit errors, and detecting two bit errors, in a block of data ranging from 1 to 11 bits in length. The technique applied is a modified version of a Hamming code, and has been implemented entirely in C. Additional functions have been provided to program and read the EEPROM on an MC68HC11 microcontroller unit using the error encoding and decoding algorithms.

ENCODING AND DECODING ALGORITHMS

Some texts [1], [2] describe the use of simultaneous equations to calculate check bits in Hamming distance-3 error correcting codes. These codes are so named because there are at least 3 bit differences between each valid code in the set of available codes. The codes are relatively easy to generate and can be used to correct one bit errors. However, their main drawback is that if two bit errors occur, then the correction will be made erroneously. This is because the condition of two bit errors corresponds exactly with a one bit error from another valid code.

The technique described here is based on an algorithmic strategy which produces Hamming distance-4 codes over the range of 1 to 11 data bits. This type of code is capable of correcting single bit errors and detecting 2 bit errors.

Alternatively, if the errors are only to be detected, without correction, then up to 3 bit errors can be detected. The reason for this is that the condition of a 3 bit error in one code corresponds to a one bit error from an adjacent valid code. The implication of this is that, if the algorithms are used to correct errors, then a 3 bit error will be corrected erroneously, and flagged as a 1 bit error.

The C program is divided into 3 modules, plus one header file:

1. EECOR1.C
 This is the main program segment, and serves only to illustrate the method of calling and checking the algorithms.
2. HAMMING.C
 This module contains the functions which encode and decode the data.
3. EEPROG.C
 This module contains the EEPROM programming functions tailored for an MC68HC11 MCU.
4. HC11REG.H
 This is the header file which contains the MC68HC11 I/O register names, defined as a C structure.

IMPLEMENTATION OF ERROR CORRECTION STRATEGY

The basic principle of decoding the error correcting codes is to use a Parity check matrix, H, to generate a syndrome word which identifies the error. The H matrix can be generated as follows:

1. Identify how many data bits are needed. For example: 8 data bits
2. Use the standard equation to derive the number of check bits required: If k is the number of check bits, and m the number of data bits, then for the Hamming bound to be satisfied:

$$2^k \geq m + k + 1$$

A simple way to understand why this equation holds true is as follows: If one can generate a check code which is able to identify where a single error occurs in a bit stream, then the check code must have at least the same number of unique combinations as there are bits in the bit stream, plus 1 extra combination to indicate that no error has occurred. e.g. if the total number of data plus check bits were 7, then the check code must consist of 3 bits, to cover the range 1 to 7 plus one extra (0) to indicate no error at all.

In this example, if $m=8$ then, by rearranging the above equation:

$$2^k - k - 1 \geq 8$$

One way to solve for k is to just select values of k starting at say, 1 and evaluating until the bound is reached. This method is implemented algorithmically in function `InitEncode()` in module `HAMMING.C`

For $m=8$, the solution is $k=4$. Note that this value exceeds the Hamming bound, which means that additional data bits can be added to the bit stream, thus increasing the efficiency of the code. In fact, the maximum number of data bits is 11 in this case.

- A Parity matrix, H is created from a 'horizontally orientated' binary table. The number of columns (b1 to b12) in the matrix correspond to the total number of data and check bits, and the number of rows (r1 to r4) to the number of check bits.

i.e.

	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
r1	1	0	1	0	1	0	1	0	1	0	1	0
r2	0	1	1	0	0	1	1	0	0	1	1	0
r3	0	0	0	1	1	1	1	0	0	0	0	1
r4	0	0	0	0	0	0	0	1	1	1	1	1

Because the H matrix in this form, is simply a truncated 4 bit binary table, it can easily be generated algorithmically.

- The position of all the check bits (C1 to C4) within the encoded word is the position of the single 1s in the columns of H . The remaining bits correspond to the data bits (D1 to D8).

i.e.

	C1	C2	D1	C3	D2	D3	D4	C4	D5	D6	D7	D8
1	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	0	0	1	1	0	0
0	0	0	1	1	1	1	0	0	0	0	0	1
0	0	0	0	0	0	0	1	1	1	1	1	1

- Each check bit is generated by taking each row of H in turn, and modulo-2 adding all bits with a 1 in them except the check bit positions.

i.e.

$$C1 = D1 + D2 + D4 + D5 + D7$$

$$C2 = D1 + D3 + D4 + D6 + D7$$

$$C3 = D2 + D3 + D4 + D8$$

$$C4 = D5 + D6 + D7 + D8$$

- The syndrome, s , is the binary weighted value of all check bits.

i.e. $s = 1 * C1 + 2 * C2 + 4 * C3 + 8 * C4$

- The error position (i.e. column) is determined by the value of the syndrome word, provided it is not zero. A zero syndrome means no error has occurred. Note that this error correction technique can correct errors in either data or check bits - which is not necessarily the case with certain other error correction strategies.

The advantage of this method, where the check bits are interspersed in a binary manner throughout the code word, is that the error position can be calculated algorithmically.

An important point to note is that the parity check matrix described above generates Hamming distance-3 codes, which means that 2 errors will cause erroneous correction. This can be fixed by adding an extra parity check bit, C5, which is the modulo-2 addition of all data and check bits together.

i.e. $C5 = C1 + C2 + D1 + C3 + D2 + D3 + D4 + C4 + D5 + D6 + D7 + D8$

The code word then becomes:

C1 C2 D1 C3 D2 D3 D4 C4 D5 D6 D7 D8 C5

To determine if an uncorrectable error has occurred (i.e. 2 errors) in the received word, the extra parity bit is tested. If the syndrome is non-zero and the parity bit is wrong, then a correctable error has occurred. If the syndrome is non-zero and the parity bit is correct, then an uncorrectable error has occurred.

EFFICIENCY

The following table lists the relative efficiencies of this algorithm, against data size.

Data bits	Encoded bits	Efficiency %
1	4	25
2	6	33
3	7	43
4	8	50
5	10	50
6	11	55
7	12	58
8	13	62
9	14	64
10	15	67
11	16	69

The implementation of the above techniques are given in the module HAMMING.C.

In order to maintain orthogonality in the EEPROM algorithms, the encoded data used by the functions in module EEPORG.C are forced to either 1 byte or 2 byte (word) sizes. This also eliminates the complexities of packing and unpacking data in partially filled bytes.

CONCLUSIONS

In this application note, the encoding algorithm's generator matrix is the same as the parity check matrix.

The C functions <read> and <write> in the module HAMMING.C return a status value - 0, 1 or 2 - which indicates whether the data has no errors, 1 corrected error, or 2 erroneously corrected errors. This means that if the status value is 0 or 1, then the data can be assumed good. If the status value is 2, then the data will be bad.

Alternatively the functions can be used for error detection only, without correction. In this case, a status value of 1 corresponds to either 1 or 3 bit errors, while a status value of 2 indicates that 2 bit errors have occurred.

By using the C functions listed in this application note, the encoded data size can easily be changed dynamically. To do this, the function <InitEncode> must be called with

the required new data size. The global variables used by all the encoding, decoding and EEPROM programming and reading functions are automatically updated. This allows the encoding and error correction process to be virtually transparent to the user. In addition, the functions <write> and <read> will automatically increment the address pointer by the correct encoded data size set up by <InitEncode>. This simplifies the structure of loops to program and read back data. Example code is provided in module EECOR1.C.

The encoding and decoding algorithms listed here may be applied to other forms of data, such as that used in serial communications, or for parallel data transfers.

By incorporating the error correction or detection-only schemes described in this application note, the integrity of data storage and transfer can be greatly improved. The impact on EEPROM usage is to increase its effective reliability and extend its useful life beyond the manufacturers' guaranteed specifications.

REFERENCES

- [1] Carlson, 'Communication Systems', Chapter 9, McGraw-Hill.
- [2] Harman, 'Principles of the Statistical Theory of Communication', Chapter 5, McGraw-Hill

MODULE EECOR1.C

Tests EEPROM error detection using a modified hamming encoding scheme.

```
typedef unsigned char byte;
typedef unsigned int word;
```

```
/* Global variables used by main() */
byte *ee_addr,*start_addr,*end_addr,i,Error;
word data;
```

```
/******
```

```
/* External global variables */
extern byte CodeSize; /* = number of bits in encoded data */
```

```
/* External Functions */
extern byte read(word *data,byte **addr); /* Function returns error status */
extern byte write(word data,byte **addr); /* "
```

```
/* Table of Status returned by read and write functions
Returned Status Condition
0 No errors detected or corrected.
1 One error detected and corrected.
2 Two errors detected, but correction is erroneous.
```

Notes:

1/ When the returned value is 2, the function <read> will returned a bad value in variable <data> due to the inability to correctly correct two errors. <read> also automatically increments the address pointer passed to it, to the next memory space. The incremented value takes into account the actual size of the encoded data. i.e. either 1 or 2 byte increment.

2/ Function <write> also performs a read to update and return an error status. This gives an immediate indication of whether the write was successful. <write> also automatically increments the address pointer passed to it, to the next free memory space. The incremented value takes into account the actual size of the encoded data. i.e. either 1 or 2 byte increment.

```
*/
```

```
/******
```

```
int main()
```

```
{
    CodeSize=InitEncode(11); /* Get code size (less 1) needed */
                             /* by 11 data bits */
    ee_addr=(byte *)0xb600; /* Initialise EEPROM start address */
    for(i=1;i<=0x10;i++) /* and 'erase' EEPROM */
        Error=write(0x7ff,&ee_addr); /* Function successful if Error<=2 */
    ee_addr=(byte *)0xb600; /* Reset EEPROM address */
    Error=write(0x5aa,&ee_addr); /* Write 0x5aa & increment ee_addr */
    Error=write(0x255,&ee_addr); /* Write 0x255 at next available address */
    CodeSize=InitEncode(4); /* Change number of data bits to 4 */
    start_addr=ee_addr; /* Save start address for this data */
    for(i=1;i<0x10;i<=1) /* Program 'walking 1s' */
        Error=write(i,&ee_addr);
    end_addr=ee_addr; /* Save end address */
    ee_addr=start_addr;
    while (ee_addr<end_addr) /* Read back all the 4 bit data */
        Error=read(&data,&ee_addr); /* <data> good if Error=0 or 1 */
} /* main */
```

MODULE HAMMING.C

/*Modules to Generate hamming codes of distance 4, for data sizes in the range 1 bit to 11 bits. The upper bound is limited by the encoded word type bit range (16 bits).

Corrects 1 bit error in any position (check or data), and detects 2 bit errors in any position.

After execution of the <Decode> function, the global variable <ErrFlag> is updated to indicate level of error correction.

i.e.	ErrFlag	Condition
	0	No errors detected or corrected.
	1	One error detected and corrected.
	2	Two errors detected, but correction is erroneous.

Note that when ErrFlag is 2, function <Decode> will return a bad value, due to its inability to correctly correct two errors.

*/

```
#define TRUE 1
```

```
#define FALSE 0
```

```
typedef unsigned char byte;
```

```
typedef unsigned int word;
```

```
byte DataSize, CodeSize, EncodedWord, ErrFlag;
```

```
/* Function prototypes */
```

```
byte OddParity(word Code);
```

```
word Power2(byte e);
```

```
byte InitEncode(byte DataLength);
```

```
word MakeCheck(word Data);
```

```
word Encode(word Data);
```

```
word Decode(word Code);
```

```
byte OddParity(Code)
```

```
word Code;
```

```
/*
```

```
Returns TRUE if Code is odd parity, otherwise returns FALSE
```

```
*/
```

```
{  
  byte p;
```

```
  p=TRUE;
```

```
  while (Code!=0)
```

```
  {  
    if (Code & 1) p=!p;  
    Code>>=1;
```

```
  }  
  return(p);
```

```
}
```

```
word Power2(e)
```

```
byte e;
```

```
/*
```

```
Returns 2^e
```

```
*/
```

```
{
```

```
  word P2;
```

```
  signed char i;
```

```
  P2=1;
```

```
  if ((signed char) (e)<0)
```

```
    return(0);
```

```
  else
```

```

    {
        for (i=1;i<=(signed char) (e);i++)
            P2<<=1;
        return (P2);
    }
}

byte InitEncode(DataLength)
byte DataLength;
/*
Returns the minimum number of total bits needed to provide
Hamming distance 3 codes from a data size defined by passed
variable <DataLength>. This value also updates global variable <DataSize>.
i.e. finds the minimum solution of (k+m) for the inequality:
 $2^k \geq k + m + 1$ 

In addition, updates global variable <EncodedSize> to reflect number of bytes
per encoded data. <EncodedSize> will be either 0 or 1.
*/

{
    byte CheckLength,i;

    DataSize=DataLength; /* DataSize used by other functions in this module */
    CheckLength=1;
    while ((Power2(CheckLength)-CheckLength-1)<DataLength)
        CheckLength++;
    i=CheckLength+DataLength;
    EncodedWord=i / 8; /* =0 if byte sized, =1 if word sized */
    return (CheckLength+DataLength);
}

word MakeCheck(Data)
word Data;
/*
Returns a check word for Data, based on global variables <DataSize>
and <CheckSize>. The H parity matrix is generated by a simple for loop.
*/

{
    byte i,H,CheckSize,CheckValue,Check,CheckMask;
    word DataMask;

    Check=0;
    CheckMask=1;
    CheckSize=CodeSize-DataSize;
    for (i=1;i<=CheckSize;i++)
    {
        CheckValue=FALSE;
        DataMask=1;
        for (H=1;H<=CodeSize;H++)
        {
            if ((0x8000 % H)!=0) /* Column with single bit set */
            {
                if ((H & CheckMask)!=0)
                    CheckValue^=((DataMask & Data)!=0);
                DataMask<<=1;
            }
        }
        if (CheckValue) Check|=CheckMask;
        CheckMask<<=1;
    }
    return (Check);
}

```

```

word Encode(Data)
word Data;
/*
Returns an encoded word, consisting of the check bits
concatenated on to the most significant bit of <Data>.
A single odd parity bit is concatenated on to the Encoded word to
increase the hamming bound from 3 to 4, and provide 2 bit error
detection as well as 1 bit correction.
Uses global variables <DataSize> and <CodeSize> to determine the
concatenating positions.
*/
{
word Code;

Code=Data | (MakeCheck(Data)<<DataSize);
if (OddParity(Code))
Code|=Power2(CodeSize);
return (Code);
}

word Decode(Code)
word Code;
/*
Returns the error corrected data word, decoded from <Code>.
Uses global variable <DataSize> to determine position of the
check bits in <Code>.
Updates global variable <ErrFlag> to indicate error status i.e.:
ErrFlag      Status
0             No errors found
1             Single error corrected
2             Double error - invalid correction
*/
{
word ParityBit,Data,Check,ErrorCheck,Syndrome,DataMask;
byte DataPos,CheckSize,CheckPos,H,DataBit;

ErrFlag=0;
ParityBit=Code & Power2(CodeSize);           /* Extract parity bit.          */
DataMask=Power2(DataSize)-1;                 /* Make data mask */
Data=Code & DataMask;                         /* Extract data bits.          */
CheckSize=CodeSize-DataSize;                 /* Extract check bits,        */
Check=(Code>>DataSize) & (Power2(CheckSize)-1); /* ignoring parity.          */
ErrorCheck=MakeCheck(Data);
Syndrome=Check ^ ErrorCheck;                 /* Get bit position of error.  */
if (Syndrome>0) ErrFlag++;                   /* Increment flag if error exists. */
H=0;
DataPos=0;
CheckPos=DataSize,
DataBit=TRUE;

while ((H!=Syndrome) & (DataPos<DataSize)) /* Identify which data or      */
{
H++;                                         /* code bit is in error.      */
DataBit=(0x8000 % H);
if (DataBit) DataPos++;
else CheckPos++;
}
if (DataBit) Code^=Power2(DataPos-1);
else Code^=Power2(CheckPos-1);
Code|=ParityBit;
if (OddParity(Code)) ErrFlag++;
return (Code & DataMask);
}

```

MODULE EEPROM.C

/*Module to program MC68HC11 EEPROM.

Contains <read> and <write> functions to encode and decode data formatted by modified hamming scheme.

*/

#include <HC11REG.H>

#define regbase (*(struct HC11IO *) 0x1000)

#define eras 0x16

#define writ 0x02

typedef unsigned char byte;

typedef unsigned int word;

union twobytes

```
{
  word w;
  byte b[2];          /* Word stored as MSB,LSB */
} udata;              */
```

extern byte EncodedWord,ErrFlag;

/* Function prototypes */

extern word Encode(word Data);

extern word Decode(word Code);

void delay(word count);

void eeprog(byte val,byte byt,byte *addr,word count);

void program(byte byt,byte *addr);

byte read(word *data,byte **addr);

byte write(word data,byte **addr);

void delay(count)

```
word count;
{
  regbase.TOC1=regbase.TCNT+count;      /* Set timeout period on OC1 and */
  regbase.TFLG1=0x80;                   /* clear any pending OC1 flag.   */
  do;while ((regbase.TFLG1 & 0x80)==0);  /* Wait for timeout flag.       */
}
```

void eeprog(val,byt,addr,count)

```
byte val;                               /* val determines Erase or Write operation */
byte byt;                                /* byt is byte to be programmed           */
byte *addr;                              /* addr is address of encoded byte in EEPROM */
word count;                              /* count is number of E clock delays      */
{
  regbase.PPROG=val;                     /* Enable address/data latches           */
  *addr=byt;                              /* Write value to required eeprom location */
  ++regbase.PPROG;                       /* Enable voltage pump                   */
  if (count<100) count=100;              /* Allow for software overhead           */
  delay(count);                          /* wait a bit                            */
  --regbase.PPROG;                       /* Disable pump,then addr/data latches    */
  regbase.PPROG=0;
}
```

```

void program(byt, addr)
byte byt;
byte *addr;
{
    eeprog(eras, byt, addr, 20000);          /* First erase byte          */
    eeprog(writ, byt, addr, 20000);        /* Then write value         */
}

byte read(data, addr)
word *data;
byte **addr;
{
    udata.b[1]=*(*addr)++;                /* Read back data LSB first, and inc address */
    if (EncodedWord)                      /* If word stored then read MSB             */
        udata.b[0]=*(*addr)++;            /* Inc address for next call to this function */
    else                                    /* else only byte stored, so clear MSB      */
        udata.b[0]=0;
    *data=Decode(udata.w);                /* Decode data, which updates <ErrFlag>,    */
    return(ErrFlag);                      /* and return ErrFlag          */
}

byte write(data, addr)
word data;
byte **addr;
{
    byte *oldaddr;

    udata.w=Encode(data);                 /* Encode data.                */
    oldaddr=*addr;                        /* Save initial address for verification. */
    program(udata.b[1], (*addr)++);       /* Program LSB first to allow for either    */
    if (EncodedWord)                      /* 1 or 2 byte encoded data            */
        program(udata.b[0], (*addr)++);   /* MSB of word sized data, & inc address   */
    return(read(&udata.w, &oldaddr));     /* Return <ErrFlag> to calling segment    */
}

```

HC11REG.H

```

/*      HC11 structure - I/O registers for MC68HC11 */

struct HC11IO {
  unsigned char  PORTA;          /* Port A - 3 input only, 5 output only */
  unsigned char  Reserved;
  unsigned char  PIOC;          /* Parallel I/O control */
  unsigned char  PORTC;        /* Port C */
  unsigned char  PORTB;        /* Port B - Output only */
  unsigned char  PORTCL;       /* Alternate port C latch */
  unsigned char  Reserved1;
  unsigned char  DDRC;         /* Data direction for port C */
  unsigned char  PORTD;        /* Port D */
  unsigned char  DDRD;         /* Data direction for port D */
  unsigned char  PORTE;        /* Port E */
};

/*      Timer Section */

  unsigned char  CFORC;        /* Compare force */
  unsigned char  OC1M;         /* Ocl mask */
  unsigned char  OC1D;         /* Ocl data */
  int            TCNT;         /* Timer counter */
  int            TIC1;         /* Input capture 1 */
  int            TIC2;         /* Input capture 2 */
  int            TIC3;         /* Input capture 3 */
  int            TOC1;         /* Output compare 1 */
  int            TOC2;         /* Output compare 2 */
  int            TOC3;         /* Output compare 3 */
  int            TOC4;         /* Output compare 4 */
  int            TOC5;         /* Output compare 5 */
  unsigned char  TCTL1;        /* Timer control register 1 */
  unsigned char  TCTL2;        /* Timer control register 2 */
  unsigned char  TMSK1;        /* Main timer interrupt mask 1 */
  unsigned char  TFLG1;        /* Main timer interrupt flag 1 */
  unsigned char  TMSK2;        /* Main timer interrupt mask 2 */
  unsigned char  TFLG2;        /* Main timer interrupt flag 2 */

/*      Pulse Accumulator Timer Control */

  unsigned char  PACTL;        /* Pulse Acc control */
  unsigned char  PACNT;        /* Pulse Acc count */

```

```

/*      SPI registers */

unsigned char  SPCR;          /* SPI control register      */
unsigned char  SPSR;          /* SPI status register       */
unsigned char  SPDR;          /* SPI data register         */

/*      SCI registers */

unsigned char  BAUD;          /* SCI baud rate control     */
unsigned char  SCCR1;         /* SCI control register 1    */
unsigned char  SCCR2;         /* SCI control register 2    */
unsigned char  SCSR;          /* SCI status register       */
unsigned char  SCDR;          /* SCI data register         */

/*      A to D registers */

unsigned char  ADCTL;          /* AD control register       */
unsigned char  ADR[4];        /* Array of AD result registers */

/*      Define each result register */

#define      adr1      ADR[0]
#define      adr2      ADR[1]
#define      adr3      ADR[2]
#define      adr4      ADR[3]

unsigned char  Rsrv[4];        /* Reserved for A to D expansion */

/*      System Configuration */

unsigned char  OPTION;         /* System configuration options */
unsigned char  COPRST;         /* Arm/Reset COP timer circuitry */
unsigned char  PPROG;          /* EEPROM programming control reg */
unsigned char  HPRIO;          /* Highest priority i-bit int & misc */
unsigned char  INIT;           /* RAM - I/O mapping register */
unsigned char  TEST1;          /* Factory TEST control register */
unsigned char  CONFIG;         /* EEPROM cell - COP,ROM,& EEPROM en */

};

/*      End of structure HC11 */

```


Temperature measurement and display using the MC68HC05B4 and the MC14489

By Jeff Wright,
Motorola Ltd., East Kilbride

INTRODUCTION

This application note is intended to show the basic building blocks of a temperature control system based on the MC68HC05Bx family of MCUs. Software routines in the application include look-up table interpolation, binary to BCD conversion, DegC to DegF conversion and the basis of a real time counter/clock. For temperature display the Multi-character LED display driver MC14489 is used, driven from the B4's SCl, resulting in simple hardware with a low component count. The temperature sensing element used here is a thermistor to allow easy interfacing to the A/D converter of the HC05B4, but the software principles shown would be the same for many other types of sensors. A software listing is included at the end of this application note.

TEMPERATURE MEASUREMENT

A pre-calibrated thermistor was chosen as the temperature sensing element. Its characteristic curve over the temperature range of -40 to 80 °C is shown in Figure 1. To get the best accuracy from the HC05B4's on-board A/D, the input signal should be scaled to use as much of the available VRH-VRL range as possible. Here VRH is connected to Vdd and VRL is tied to Vss. In this case, using the thermistor as potential divider with a 20k Ω resistor results in a signal range of approximately 0.3V to 4.7V over the -40 to 80 °C temperature range. The voltage across the thermistor (input to the A/D), plotted against temperature, is shown in Figure 2.

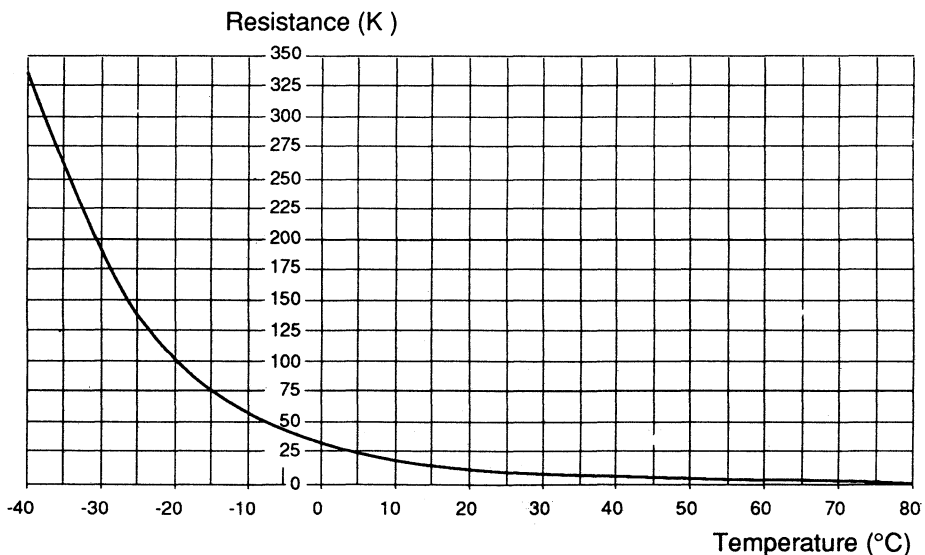


Figure 1. Thermistor resistance vs Temperature

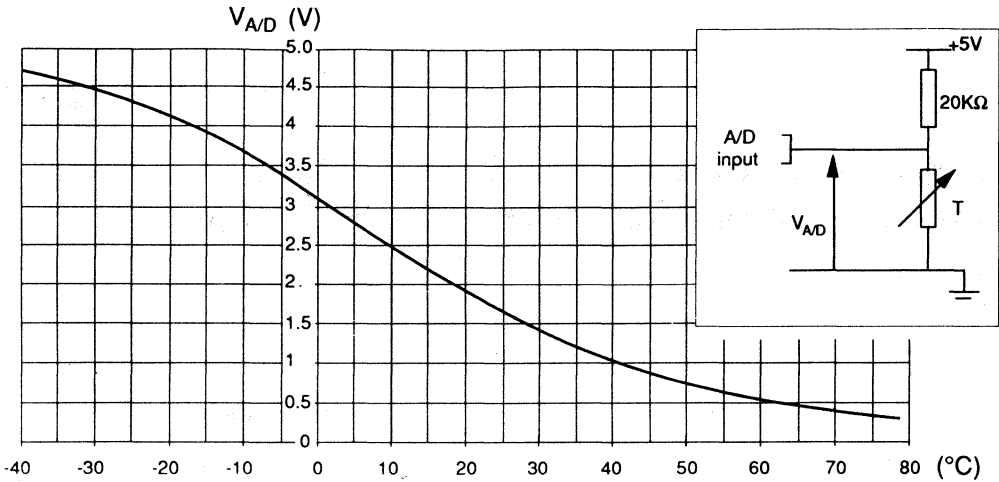


Figure 2. A/D input voltage vs Temperature (inset: circuit used)

As can be seen from Figure 2, the response is non-linear and so a look-up table approach is the simplest way of obtaining the required accuracy. The thermistor characteristics are stored as a series of points in a table in ROM and a linear interpolation between adjacent points is used to obtain the temperature that corresponds to a given A/D reading. The number of points that must be stored depends on how non-linear the response is and the required accuracy of the result. In this case 16 points were chosen; in order to keep the software simple (and

therefore fast), they are spread at intervals of 16 through the A/D result range of 0-255. For each point (16, 32, 48 etc.), the voltage on the A/D input was calculated and the corresponding temperature was obtained from the graph of Figure 2. These points were then used to form the look-up table shown in Figure 3, resulting in a temperature range of -40 to 79 °C. Figure 4 shows the reconstructed response of the thermistor obtained by linear interpolation of the points in the look-up table.

A/D RESULT	A/D (volts)	TEMP (°C)	TEMP (°C 2s Compl)
0	0	-	-
16	0.31	79	4F
32	0.63	56	38
48	0.94	43	2B
64	1.26	34	22
80	1.57	27	1B
96	1.88	21	15
112	2.20	15	0F
128	2.51	10	0A
144	2.82	5	05
160	3.14	-1	FF
176	3.45	-6	FA
192	3.77	-11	F5
208	4.08	-18	EE
224	4.39	-26	E6
240	4.71	-40	D8
255	5.0	-	-

Figure 3. Interpolated A/D input voltage vs Temperature

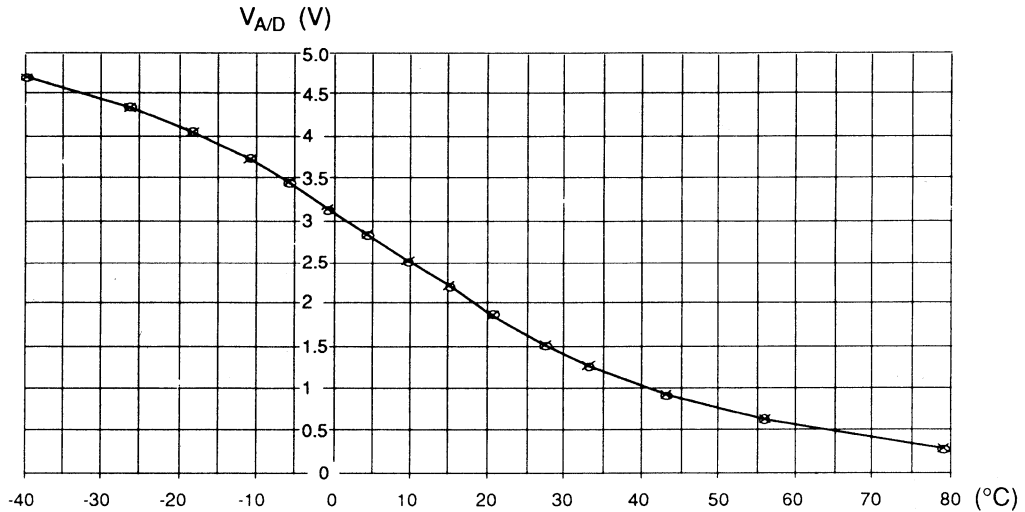


Figure 4. Interpolated A/D input voltage vs Temperature

The temperature reading is updated every second; the software to accomplish this is relatively simple:

The timer is set to overflow every 125 mS with a 4.1934 MHz crystal. The timer overflow interrupt routine updates the real time counters TICKS, SECS, MINS & HRS and sets the flag bit SEC every time a second has elapsed.

The main program loop is executed every second (via the SEC flag bit) and after checking the metric/imperial selector switch the temperature is measured by the subroutine ADCONV. This routine starts by reading the thermistor selector switch and setting up the A/D control register accordingly. An A/D conversion is then carried out four times on the selected channel and the results accumulated in the accumulator and the temporary register TEMP. This result is then divided by 4 by rotating, to obtain the average A/D result. The averaging technique is employed to try and reduce the effect of noise on the A/D input. The number of conversions to average is determined by time constraints and the noise levels in the surrounding environment. The upper nibble of the result is then used to access the look-up table to obtain the 'base' temperature value. If the temperature limit is exceeded then the TLIMIT flag is set before exiting from the routine.

Temperature table entries are stored in 2's complement form so that the interpolation between positive and negative values will work successfully. The interpolation is carried out by obtaining the difference between the base value and the next in the table, multiplying this by the lower nibble of the A/D result and then dividing by 16. This result is then subtracted from the base value to obtain the real temperature in 2's complement °C which is stored in the register NEWTMP before exiting from the routine. The difference information is subtracted from the base value rather than added because the thermistor has a negative temperature co-efficient (NTC) so that an increase in the A/D result corresponds to a drop in temperature.

If the imperial mode is selected (°F) then the next stage before updating the display is to convert from °C to °F and this is carried out in the subroutine CTOF.

Converting from °C to °F is accomplished by multiplying by 1.8 and adding 32. First the sign of the temperature in °C is stored via the flag bit NEGNUM, then the maximum °F limit (53 °C) is checked before the magnitude is multiplied by 1.8 (multiply by 115 and divide by 64). Again, use is made of rotating to do the dividing, in order to increase execution speed. The sign of the result is then restored and 32 added to obtain the temperature in 2's complement °F.

TEMPERATURE DISPLAY

An MC14489 multi-character display driver was chosen for this purpose as it can be easily interfaced to a wide range of Motorola MCUs, requires almost no external components and has a character set that includes the degree symbol ($^{\circ}$). The MC14489 can also be cascaded if the application was expanded to require a larger display. The MC14489 would normally be driven from an SPI on the MCU but here, since the the 68HC05B family does not have an SPI, use is made of the SCI clock output feature that is available on this family.

Before the temperature can be written to the display driver it has to be converted into the correct data format.

The first stage of this is to convert from 2's complement binary to BCD. This is carried out in the routine CONBCD which is called from SETDISP. The sign of the temperature is stored in the flag bit NEGNUM before SETDISP is called; then, after first checking if the TLIMIT flag is set, the temperature is converted to BCD in DECO-2 by CONBCD. This is accomplished by rotating left the binary number followed immediately by a rotate left of the BCD result; this has the effect of multiplying the current BCD result by 2 and adding in the new binary bit at the same time. After each rotate the BCD registers are checked and adjusted for overflow (>\$09) before the bit counter contained in the index register is decremented. This process of rotate then adjust is continued until all the binary bits have been used; the BCD result will then be resident in the registers DECO, 1 & 2.

The rest of the routine SETDISP is concerned with setting up the display registers DISP1, 2, 3 and the display control register DISPC. The MC14489 data format is msb first whereas the 68HC05B4 SCI transmits lsb first; this means that the bit order of the data stream has to be stored in reverse in the display registers. This can be confusing when trying to work out the codes that have to be stored in the B4 to generate a specific character.

Figures 5a and 5b show the 14489 data format and the corresponding bit positions in the B4 registers DISP1, 2, 3 & C. The sign of the temperature is restored and the numeric display registers are configured to display '-' if the temperature limit has been exceeded before exiting from the SETDISP routine.

The main program loop then calls the subroutine DISPL which actually transmits the contents of the display registers to the MC14489 via the SCI. The MC14489 contains special Bit Grabber circuitry that allows either the internal display registers or the configuration register to be updated without address or steering bits so that updating the display involves a simple transmission of either 3 bytes for the display registers or 1 byte for the configuration register. Even for cascaded 14489s there is no need for address bits - see the MC14489 data sheet for more details.

The MC14489 can be clocked at up to 4 MHz at 5 volts so here the maximum transmit baud rate of the SCI is used - 131.072 KHz with a 4.19304 MHz crystal. The transmission of the display data only takes place if there has been a change in the data since the last time. If there has been a change, the 3 data registers are transmitted in turn starting with DISP3 and the OLD registers are updated ready for the change check next time round. After the last byte has gone, the SCI and 14489 are disabled before returning to the main loop.

The last subroutine called from the main program is the 14489 configuration update routine DISCON. This routine operates in a similar manner to DISPL, checking to see if there has been a change to the config. data before transmitting it.

This completes the operation of the program which now jumps back to the start of the main loop and waits for the SEC bit to be set again before repeating the temperature measurement and display sequence.

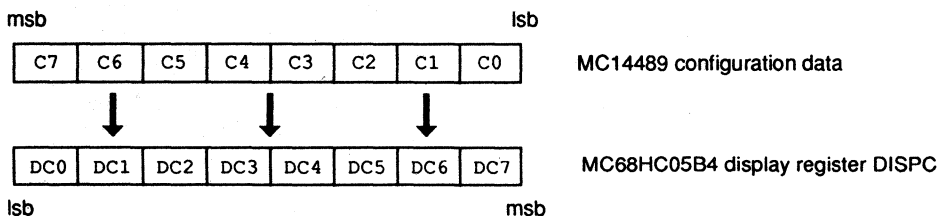


Figure 5a. MC14489 to MC68HC05B4 display register mapping

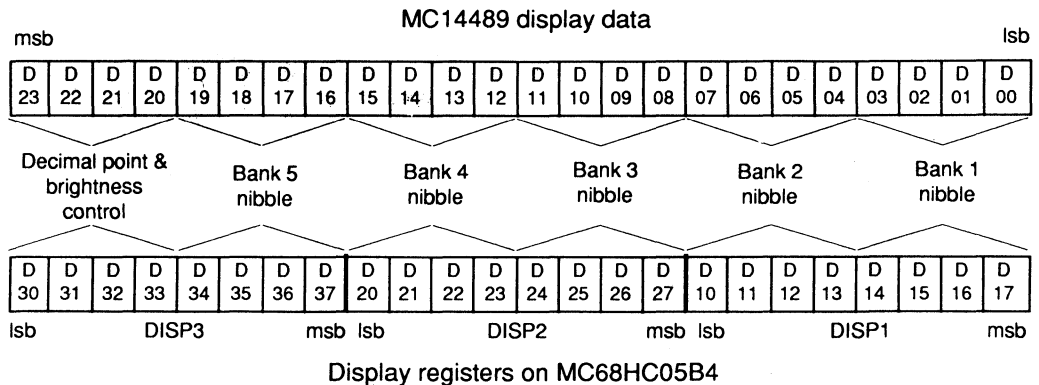


Figure 5b. MC14489 to MC68HC05B4 display register mapping

HARDWARE

As already mentioned, the use of the MC14489 results in a very low component count for the application; the hardware schematic can be seen in Figure 6. The only I/O pins required are for reading the option switches and for controlling the enable of the MC14489. Pull-downs are required on the clock and data pins as these become high impedance when the SCI is disabled. The LED displays are common cathode; a single external resistor is all that is required to set the brightness

level of the displays. In this case though, a light dependent resistor, R12 (ORP12), has been used to control the display brightness for a variety of background lighting conditions. The resistance of R12 decreases with increasing light and so R11 must be incorporated to ensure that the maximum source current spec. of the MC14489 is not exceeded in very bright lighting conditions. R13 ensures there is still enough drive current for the LEDs in dark conditions.

APPLICATION AREAS

As mentioned in the introduction, this application note is designed only to show some fundamental building blocks of a temperature control system based on the 68HC05Bx family of MCUs. Where possible, the software has been written in a modular fashion, so that the routines can easily be transported to another application and the binary to BCD routine could be expanded to handle larger numbers. The large number of I/O,

PWMs and timer functions unused show that the 68HC05B family has plenty of functionality left to perform other control functions. For example, in process control, fluid flow or speed sensors could be connected to the timer input capture pins, pressure sensors to the other A/D pins, a keypad to the I/O lines and the other I/O & PWMs used to perform output control functions.

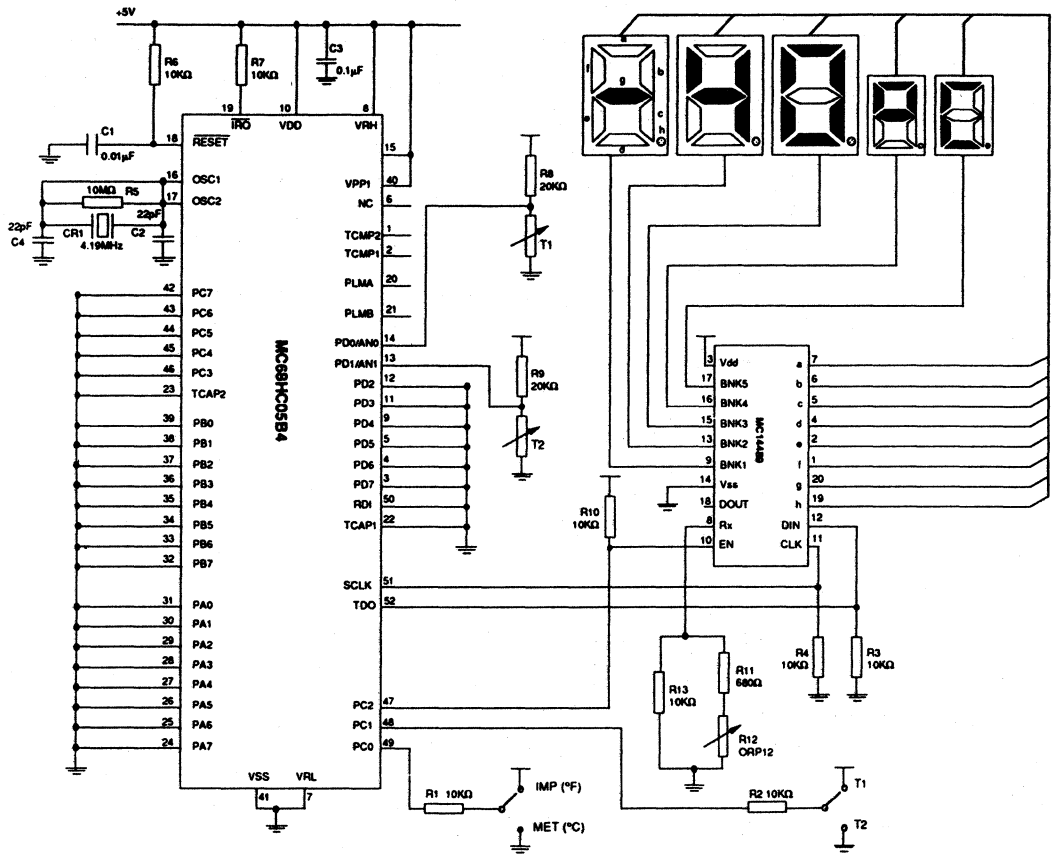


Figure 6. Hardware schematic

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64

```

```

*****
*****
**
**      68HC05B4  TEMPERATURE MEASUREMENT & DISPLAY  **
**
**      Jeff Wright, Motorola East Kilbride.   Last Updated  22/02/90  **
**
**      This software was written by Motorola for demonstration  **
**      purposes only. Motorola does not assume any liability arising  **
**      out of the application or use of this software and does not  **
**      guarantee its functionality  **
**
**
*****
*****

*****      I/O and INTERNAL registers definition      *****
*
*
*      I/O registers
*
PORTA  EQU    $00      port A.
PORTB  EQU    $01      port B.
PORTC  EQU    $02      port C.
PORTD  EQU    $03      port D.
DDRA   EQU    $04      port A DDR.
DDRB   EQU    $05      port B DDR.
DDRC   EQU    $06      port C DDR.

*
*      A/D registers
*
ADDATA EQU    $08      A/D data register.
ADSTC  EQU    $09      A/D status and control register.
COCO   EQU     7        Conversion complete flag.

*
*
*      SCI registers
*
BAUD   EQU    $0D      SCI baud register.
SCCR1  EQU    $0E      SCI control register 1.
SCCR2  EQU    $0F      SCI control register 2.
SCSR   EQU    $10      SCI status register.
TDRE   EQU     7
TC     EQU     6
SCDAT  EQU    $11      SCI data register.

*
*      TIMER registers
*
TCR    EQU    $12      Timer control register.
TOIE   EQU     5      Timer overflow interrupt enable.
OCIE   EQU     6      Timer output compares interrupt enable.
ICIE   EQU     7      Timer input captures interrupt enable.

TSR    EQU    $13      Timer status register.
OCF2   EQU     3      Timer output compare 2 flag.
ICF2   EQU     4      Timer input capture 2 flag.
TOF    EQU     5      Timer overflow flag.
OCF1   EQU     6      Timer output compare 1 flag.
ICF1   EQU     7      Timer input capture 1 flag.

```



```

65
66 00000014      TIC1HI EQU    $14      Timer input capture register 1 (16-bit).
67 00000015      TIC1LO EQU    $15
68 00000016      TOC1HI EQU    $16      Timer output compare register 1 (16-bit).
69 00000016      TOC1LO EQU    $16
70 00000018      TIMHI  EQU    $18      Timer free running counter (16-bit).
71 00000019      TIMLO  EQU    $19
72 0000001a      TIMAHI EQU    $1A      Timer alternate counter register (16-bit).
73 0000001b      TIMALO EQU    $1B
74 0000001c      TIC2HI EQU    $1C      Timer input capture register 2 (16-bit).
75 0000001d      TIC2LO EQU    $1D
76 0000001e      TOC2HI EQU    $1E      Timer output compare register 2 (16-bit).
77 0000001f      TOC2LO EQU    $1F
78
79      *
80      *
81      *      MEMORY MAP DEFINITION
82      *
83      *
84 00000020      TEST  EQU    $20      TEST register
85 00000020      ROM0  EQU    $0020    Start address of ROM0.
86 00000050      RAM   EQU    $0050    Start address of RAM.
87 00000f00      UROM  EQU    $0F00    Start address of main user ROM.
88      *
89
90
91
92      ***** RAM ALLOCATION *****
93
94      SECTION.S .RAM, ADDR=$50
95
96
97 00000050      TICKS RMB    1
98 00000051      SECS  RMB    1
99 00000052      MINS  RMB    1
100 00000053      HRS   RMB    1
101
102 00000054      FLAG  RMB    1
103 00000000      OVERFL EQU    0
104 00000001      NEGNUM EQU    1
105 00000002      TLIMIT EQU    2
106 00000003      SEC   EQU    3
107
108 00000055      MODE  RMB    1
109 00000000      IMP   EQU    0
110
111 00000056      BIN0  RMB    1
112 00000057      DEC2  RMB    1
113 00000058      DEC1  RMB    1
114 00000059      DECO  RMB    1
115
116 0000005a      NEWTMP RMB    1
117 0000005b      TEMP  RMB    1
118 0000005c      TEMP1 RMB    1
119 0000005d      TEMP2 RMB    1
120
121 0000005e      DISP1 RMB    1
122 0000005f      DISP2 RMB    1
123 00000060      DISP3 RMB    1
124 00000061      DISPC RMB    1
125 00000062      OLDD1 RMB    1
126 00000063      OLDD2 RMB    1

```

```

127 00000064          OLDD3  RMB    1
128 00000065          OLDDC  RMB    1
129
130
131                      SECTION .PAGE0,ADDR=$020
132
133 00000020 004f382b221b150f ADTAB  FCB    $00,$4F,$38,$2B,$22,$1B,$15,$0F
134 00000028 0a05ffffaf5eee6d8  FCB    $0A,$05,$FF,$FA,$F5,$EE,$E6,$D8
135
136                      *****
137                      *
138                      *      START OF CODE
139                      *
140                      *****
141
142                      SECTION .USROM,ADDR=$F00
143
144 00000f00          RESET  EQU    *
145 00000f00 a600          LDA    #$00    Initialise Ports.
146 00000f02 b700          STA    PORTA
147 00000f04 b701          STA    PORTB
148 00000f06 b704          STA    DDRA
149 00000f08 b705          STA    DDRB
150 00000f0a ae65          LDX    #OLDDC
151 00000f0c f7          INIRAM STA    ,X    Initialise all used RAM locations.
152 00000f0d 5a          DECX
153 00000f0e a350          CPX    #RAM
154 00000f10 26fa          BNE   INIRAM
155
156 00000f12 a604          LDA    #$04
157 00000f14 b702          STA    PORTC
158 00000f16 a604          LDA    #$04    PC2 output high.
159 00000f18 b706          STA    DDRC
160
161 00000f1a b613          TIMINT LDA    TSR    clr any pending flags.
162 00000f1c b619          LDA    TIMLO
163 00000f1e a620          LDA    #$20    Enable timer overflow
164 00000f20 b712          STA    TCR    interrupt.
165 00000f22 9a          CLI
166
167                      *----- START OF MAIN PROGRAM LOOP -----*
168
169 00000f23          MAINLUP EQU    *
170 00000f23 0754fd          BRCLR  SEC,FLAG,MAINLUP
171 00000f26 1754          BCLR  SEC,FLAG
172 00000f28 1155          BCLR  IMP,MODE    Check metric/imperial selector.
173 00000f2a 010202          BRCLR  0,PORTC,NOIMP    Check degC/degF switch.
174 00000f2d 1055          BSET  IMP,MODE
175 00000f2f 1354          NOIMP BCLR  NEGNUM,FLAG    Clear sign indicator.
176 00000f31 cd0ffd          JSR   ADCONV    Go measure temperature
177 00000f34 015503          BRCLR  IMP,MODE,GOMETR    (in degC - 2s compl)
178 00000f37 cd0f4e          JSR   CTOF    Convert to degF.
179 00000f3a b65a          GOMETR LDA    NEWTMP
180 00000f3c 2a03          BPL   GOMORE
181 00000f3e 40          NEGA
182 00000f3f 1254          BSET  NEGNUM,FLAG    Only use magnitude to do BCD conv.
183 00000f41 b756          GOMORE STA    BIN0    Remember the sign of the number.
184 00000f43 cd0f78          GODISP JSR   SETDISP    Store temperature for conv to BCD.
185 00000f46 cd1085          JSR   DISPL    Set-up display bytes.
186 00000f49 cd10ca          JSR   DISCON    Update display if neccessary.
187 00000f4c 20d5          BRA   MAINLUP    Update 14489 config if neccessary.
188
189

```

```

190
191
192
193
194
195
196 0000f4e          CTOF EQU *
197 0000f4e b65a     LDA NEWTMP
198 0000f50 2a05     BPL NONEG
199 0000f52 1254     BSET NEGNUM,FLAG    Remember if No is negative or not.
200 0000f54 40       NEGA
201 0000f55 2007     BRA MULIP8
202 0000f57 a135     NONEG CMP #53          Check for max degF limit of 127F.
203 0000f59 2503     BLO MULIP8
204 0000f5b 1454     BSET TLIMIT,FLAG    Set limit and return if over range.
205 0000f5d 81       RTS
206
207 0000f5e ae73     MULIP8 LDX #115
208 0000f60 42       MUL          Multiply by 115 and divide by 64.
209 0000f61 56       RORX
210 0000f62 46       RORA          (same as multiplying by 1.8)
211 0000f63 56       RORX
212 0000f64 46       RORA
213 0000f65 56       RORX
214 0000f66 46       RORA
215 0000f67 56       RORX
216 0000f68 46       RORA
217 0000f69 56       RORX
218 0000f6a 46       RORA
219 0000f6b 56       RORX
220 0000f6c 46       RORA
221
222 0000f6d 035401    BRCLR NEGNUM,FLAG,NONEG1
223 0000f70 40       NEGA          Return sign of number.
224 0000f71 1354    NONEG1 BCLR NEGNUM,FLAG
225 0000f73 ab20     ADD #32       Add 32 to get degF.
226 0000f75 b75a     STA NEWTMP
227 0000f77 81       RTS
228
229
230
231
232
233
234
235
236 0000f78          SETDISP EQU *
237 0000f78 04543e    BRSET TLIMIT,FLAG,FORCE    If temp out of range, force to -
238 0000f7b ae08     LDX #8
239 0000f7d cd1052    JSR CONBCD    Convert 8 bit binary to 3 digit BCD.
240 0000f80 ae04     LDX #4
241 0000f82 4f       CLRA
242 0000f83 3458    LUPDIS1 LSR DEC1    Shuffle bit order of digits to allow
243 0000f85 49       ROLA          for SCI lsb first and 14489 msb first
244 0000f86 5a       DECB          incompatability.
245 0000f87 26fa     BNE LUPDIS1
246 0000f89 be57     LDX DEC2
247 0000f8b 2704     BEQ TSTNEG
248 0000f8d aa80     ORA #80       If over 100deg, add the 100 digit.
249 0000f8f 2005     BRA STD1
250 0000f91 035402    TSTNEG BRCLR NEGNUM,FLAG,STD1
251 0000f94 aab0     ORA #80       Add code for a - if temp is negative.
252 0000f96 b75e     STD1 STA DISP1    Store in 1st display register.

```

```

253 00000f98 ae08          LDX      #8
254 00000f9a 4f          CLRA
255 00000f9b 3459      LUPDIS2  LSR      DECO
256 00000f9d 49          ROLA
257 00000f9e 5a          DECX                               Shuffle bit order of digits as above.
258 00000f9f 26fa      BNE      LUPDIS2
259 00000fa1 aa0f      ORA      #0f          add code for the deg symbol.
260 00000fa3 b75f      STA      DISP2       Store in second display register.
261 00000fa5 a631      LDA      #31         Big C, all d.ps off.
262 00000fa7 015502   BRCLR   IMP,MODE,STDIS3
263 00000faa a6f1      LDA      #F1         Big F, all d.ps off.
264 00000fac b760      STDIS3  STA      DISP3
265 00000fae a6cb      LDA      #CB
266 00000fb0 be57      LDX      DEC2
267 00000fb2 2702      BEQ      STDISC
268 00000fb4 a68b      LDA      #8B
269 00000fb6 b761      STDISC  STA      DISPC
270 00000fb8 81          RTS
271 00000fb9 a6bb      FORCE    LDA      #BB          FORCE DISPLAY TO ^C
272 00000fbb b75e      STA      DISP1
273 00000fbd a6bf      LDA      #BF          or ^F
274 00000fbf b75f      STA      DISP2
275 00000fc1 a631      LDA      #31
276 00000fc3 015502   BRCLR   IMP,MODE,STDI3
277 00000fc6 a6f1      LDA      #F1
278 00000fc8 b760      STDI3   STA      DISP3
279 00000fca a6fb      LDA      #FB
280 00000fcc b761      STA      DISPC
281 00000fce 81          RTS
282
283
284
285          *0000000000000000000000000000000000000000000000000000000000000000*
286          *0                                                    0*
287          *0          TOVINT                - Timer Overflow IRQ routine          0*
288          *0                                                    0*
289          *00000000000000000000000000000000000000000000000000000000000000*
290
291 00000fcf 0b132a   TOVINT  BRCLR   TOF,TSR,NOOVF  Check Tim overflow has really happened.
292 00000fd2 3c50          INC      TICKS
293 00000fd4 b650          LDA      TICKS          UPDATE REAL TIME CLOCK COUNTERS
294 00000fd6 a108          CMP      #8
295 00000fd8 2520          BLO      NOINC
296 00000fda 3f50          CLR      TICKS
297 00000fdc 3c51          INC      SECS
298 00000fde 1654          BSET    SEC,FLAG
299 00000fe0 b651          LDA      SECS
300 00000fe2 a13c          CMP      #60
301 00000fe4 2514          BLO      NOINC
302 00000fe6 3f51          CLR      SECS
303 00000fe8 3c52          INC      MINS
304 00000fea b652          LDA      MINS
305 00000fec a13c          CMP      #60
306 00000fee 250a          BLO      NOINC
307 00000ff0 3f52          CLR      MINS
308 00000ff2 b653          LDA      HRS
309 00000ff4 a1ff          CMP      #FF
310 00000ff6 2702          BEQ      NOINC
311 00000ff8 3c53          INC      HRS
312
313 00000ffa b619      NOINC   LDA      TIMLO          Clear TOF flag.
314 00000ffc 80          NOOVF   RTI
315

```



```

379
380
381
382
383
384
385 00001052      CONBCD EQU      *
386 00001052 4f          CLRA
387 00001053 b759        STA      DEC0      Clear BCD result bytes.
388 00001055 b758        STA      DEC1
389 00001057 b757        STA      DEC2
390
391
392 00001059 3956      LUPBCD ROL      BIN0      Put the next binary bit in carry.
393 0000105b 3959        ROL      DEC0      Multiply current result by 2 and
394 0000105d 3958        ROL      DEC1      add in new bit at same time.
395 0000105f 3957        ROL      DEC2
396 00001061 b659        LDA      DEC0
397 00001063 a00a        SUB      #$0A
398 00001065 2b04        BMI      TSTD1      Now check the BCD bytes
399 00001067 b759        STA      DEC0      for overflow.
400 00001069 3c58        INC      DEC1
401 0000106b b658      TSTD1 LDA      DEC1
402 0000106d a00a        SUB      #$0A
403 0000106f 2b04        BMI      TSTD2
404 00001071 b758        STA      DEC1
405 00001073 3c57        INC      DEC2
406 00001075 b657      TSTD2 LDA      DEC2
407 00001077 a00a        SUB      #$0A
408 00001079 2b06        BMI      NOOVR
409 0000107b a609        LDA      #9
410 0000107d b757        STA      DEC2      BCD number has overflowed so set flag
411 0000107f 1054      BSET     OVERFL,FLAG and set upper digit to 9.
412 00001081 5a          NOOVR   DECX
413 00001082 26d5      BNE     LUPBCD      Any more bits to do?
414 00001084 81          RTS
415
416
417
418
419
420
421
422
423 00001085      DISPL EQU      *
424 00001085 b65e      LDA      DISP1      Only update display registers if any
425 00001087 b162      CMP      OLDD1      of them have changed since the last
426 00001089 260d      BNE     UPDATE      time.
427 0000108b b65f      LDA      DISP2
428 0000108d b163      CMP      OLDD2
429 0000108f 2607      BNE     UPDATE
430 00001091 b660      LDA      DISP3
431 00001093 b164      CMP      OLDD3
432 00001095 2601      BNE     UPDATE
433 00001097 81          RTS
434
435 00001098 a601      UPDATE  LDA      #$01
436 0000109a b70e      STA      SCCR1      Clock idle low, edge in mid data, last clk.
437 0000109c 4a          DECA
438 0000109d b70d      STA      BAUD      131.072KHz baud with 4.19etc XTAL.
439 0000109f a608      LDA      #$08
440 000010a1 b70f      STA      SCCR2      Transmit enabled.
441 000010a3 0d10fd  PREAM  BRCLR   TC,SCSR,PREAM Wait for preamble to finish.
442 000010a6 1502      BCLR   2,PORTC      Enable transmission to 14489.

```

```

443 000010a8 b660          LDA    DISP3
444 000010aa b764          STA    OLDD3
445 000010ac b711          STA    SCDAT          Send first byte.
446 000010ae 0f10fd DWAIT1 BRCLR  TDRE,SCSR,DWAIT1 Wait until it has been transfered
447 000010b1 b65f          LDA    DISP2          - then load second.
448 000010b3 b763          STA    OLDD2
449 000010b5 b711          STA    SCDAT
450 000010b7 0f10fd DWAIT2 BRCLR  TDRE,SCSR,DWAIT2
451 000010ba b65e          LDA    DISP1
452 000010bc b762          STA    OLDD1
453 000010be b711          STA    SCDAT
454 000010c0 0d10fd DWAIT3 BRCLR  TC,SCSR,DWAIT3 Wait until 3rd byte has actually gone
455 000010c3 a600          LDA    #S00
456 000010c5 b70f          STA    SCCR2          Dissable SCI transmissions,
457 000010c7 1402          BSET   2,PORTC       then disable 14489.
458 000010c9 81           RTS
459
460
461 *????????????????????????????????????????????????????????????????????????????????????*
462 *?                                                                ?*
463 *?      DISCON - Updates 14489 Config register via SCI          ?*
464 *?                                                                ?*
465 *????????????????????????????????????????????????????????????????????????????????????*
466
467
468 000010ca          DISCON EQU    *
469 000010ca b661          LDA    DISPC          Only update config register if it has
470 000010cc b165          CMP    OLDDC          changed since last time.
471 000010ce 2601          BNE    UPDCON
472 000010d0 81           RTS
473
474 000010d1 a601          UPDCON LDA    #S01
475 000010d3 b70e          STA    SCCR1          Clock idle low, edge in mid data, last clk.
476 000010d5 4a          DECA
477 000010d6 b70d          STA    BAUD          131.072KHz baud with 4.19etc XTAL.
478 000010d8 a608          LDA    #S08
479 000010da b70f          STA    SCCR2          Transmit enabled.
480 000010dc 0d10fd PREAM1 BRCLR  TC,SCSR,PREAM1 Wait for preamble to finish.
481 000010df 1502 DOCONF BCLR   2,PORTC     Enable transmission to 14489.
482 000010e1 b661          LDA    DISPC
483 000010e3 b765          STA    OLDDC
484 000010e5 b711          STA    SCDAT
485 000010e7 0f10fd DWAIT4 BRCLR  TDRE,SCSR,DWAIT4 Wait until config byte has transfered.
486 000010ea a600          LDA    #S00
487 000010ec b70f          STA    SCCR2          Now disable SCI transmission.
488 000010ee 0d10fd DWAIT5 BRCLR  TC,SCSR,DWAIT5 Wait until config byte has actually gone.
489 000010f1 1402          BSET   2,PORTC       Disable 14489 & return.
490 000010f3 81           RTS
491
492
493 *****
494 *                                                                *
495 *      VECTOR ADDRESSES                                          *
496 *                                                                *
497 *****
498 SECTION .VECT,ADDR=$1FF2
499
500 00001ff2 0f00          SCIINT FDB    RESET
501 00001ff4 0fcf          TOVFLW FDB    TOVINT
502 00001ff6 0f00          TOCMP  FDB    RESET
503 00001ff8 0f00          TICAP  FDB    RESET
504 00001ffa 0f00          EXTINT FDB    RESET
505 00001ffc 0f00          SOFTI  FDB    RESET
506 00001ffe 0f00          POR    FDB    RESET

```

Section synopsis

```

1 00000016 (      22) .RAM
2 00000010 (      16) .PAGE0
3 000001f4 (     500) .USROM
4 0000000e (      14) .VECT

```

Symbol table

```

.PAGE0 2 00000000 | DISPC 1 00000061 | LUPBCD 3 00001059 | OLDD3 1 00000064 | TEMP 1 0000005b
.RAM 1 00000000 | DOCONF 3 000010df | LUPDIS1 3 00000f83 | OLDDC 1 00000065 | TEMP1 1 0000005c
.USROM 3 00000000 | DWAIT1 3 000010ae | LUPDIS2 3 00000f9b | POR 4 00001ffe | TEMP2 1 0000005d
.VECT 4 00000000 | DWAIT2 3 000010b7 | MINS 1 00000052 | PREAM 3 000010a3 | TICAP 4 00001ff8
ADLUP1 3 0000100f | DWAIT3 3 000010c0 | MODE 1 00000055 | PREAM1 3 000010dc | TICKS 1 00000050
ADTAB 2 00000020 | DWAIT4 3 000010e7 | MULLP8 3 00000f5e | SCIINT 4 00001ff2 | TIMINT 3 00000f1a
BIN0 1 00000056 | DWAIT5 3 000010ee | NEWTMP 1 0000005a | SECS 1 00000051 | TOCMP 4 00001ff6
CONT1 3 00001008 | EXTINT 4 00001ffa | NOIMP 3 00000f2f | SETAD 3 0000100a | TOVFLW 4 00001ff4
DEC0 1 00000059 | FLAG 1 00000054 | NOINC 3 00000ffa | SOFTI 4 00001ffc | TOVINT 3 00000fcf
DEC1 1 00000058 | FORCE 3 00000fb9 | NONEG 3 00000f57 | STD1 3 00000f96 | TRANGE 3 0000104f
DEC2 1 00000057 | GODISP 3 00000f43 | NONEG1 3 00000f71 | STD13 3 00000fc8 | TSTD1 3 0000106b
DECCX 3 00001018 | GOMETR 3 00000f3a | NOOVF 3 00000ffc | STDIS3 3 00000fac | TSTD2 3 00001075
DISP1 1 0000005e | GOMORE 3 00000f41 | NOOVR 3 00001081 | STDISC 3 00000fb6 | TSTNEG 3 00000f91
DISP2 1 0000005f | HRS 1 00000053 | OLDD1 1 00000062 | TABL 3 00001023 | UPDATE 3 00001098
DISP3 1 00000060 | INIRAM 3 00000f0c | OLDD2 1 00000063 | TEMP 1 0000005b

```

Symbol cross-reference

```

.PAGE0 *131
.RAM *94
.USROM *142
.VECT *498
ADLUP1 *334 334 339
ADTAB *133 351 357
BIN0 *111 183 392
CONT1 327 *330
DEC0 *114 255 387 393 396 399
DEC1 *113 242 388 394 400 401 404
DEC2 *112 246 266 389 395 405 406 410
DECCX 336 *338
DISP1 *121 252 272 424 451
DISP2 *122 260 274 427 447
DISP3 *123 264 278 430 443
DISPC *124 269 280 469 482
DOCONF *481
DWAIT1 *446 446
DWAIT2 *450 450
DWAIT3 *454 454
DWAIT4 *485 485
DWAIT5 *488 488
EXTINT *504
FLAG *102 170 171 175 182 199 204 222 224 237 250 298 325 376 411
FORCE 237 *271
GODISP *184
GOMETR 177 *179
GOMORE 180 *183
HRS *100 308 311
INIRAM *151 154
LUPBCD *392 413
LUPDIS1 *242 245
LUPDIS2 *255 258
MINS *99 303 304 307
MODE *108 172 174 177 262 276

```


MULIP8	201	203	*207				
NEWTMP	*116	179	197	226	374		
NOIMP	173	*175					
NOINC	295	301	306	310	*313		
NONEG	198	*202					
NONEG1	222	*224					
NOOVF	291	*314					
NOOVR	408	*412					
OLDD1	*125	425	452				
OLDD2	*126	428	448				
OLDD3	*127	431	444				
OLDDC	*128	150	470	483			
POR	*506						
PREAM	*441	441					
PREAM1	*480	480					
SCIINT	*500						
SECS	*98	297	299	302			
SETAD	329	*331					
SOFTI	*505						
STD1	249	250	*252				
STD13	276	*278					
STD1S3	262	*264					
STD1SC	267	*269					
TABL	*346						

Symbol cross-reference

TEMP	*117	326	337	340	342	344	359
TEMP1	*118	355	372				
TEMP2	*119	358	361	371	373		
TICAP	*503						
TICKS	*97	292	293	296			
TIMINT	*161						
TOCMP	*502						
TOVFLW	*501						
TOVINT	*291	501					
TRANGE	352	354	*376				
TSTD1	398	*401					
TSTD2	403	*406					
TSTNEG	247	*250					
UPDATE	426	429	432	*435			
UPDCON	471	*474					

128K byte addressing with the M68HC11

By Ross Mitchell
MCU Applications Engineering
Motorola Ltd., East Kilbride, Scotland

OVERVIEW

The maximum direct addressing capability of the M68HC11 device is 64K bytes, but this can be insufficient for some applications. This application note describes two methods of memory paging that allow the MCU to fully address a single 1 megabit EPROM (128K bytes) by manipulation of the address lines.

The two methods illustrate the concept of paging and the inherent compromises. The technique may be expanded to allow addressing of several EPROM, RAM or EEPROM memories or several smaller memories by using both address lines and chip enables.

PAGING SCHEME

The M68HC11 8-bit MCU is capable of addressing up to 64K bytes of contiguous address space. Addressing greater than 64K bytes requires that a section of the memory be replaced with another block of memory at the same address range. This technique of swapping memory is known as paging and is simply a method of overlaying blocks of data over each other such that only one of the blocks or pages is visible to the CPU at a given time.

In a system requiring more than 64K bytes of user code and tables, it is possible to use the port lines to extend the memory addressing range of the M68HC11 device. This has certain restrictions but these can be minimised by careful consideration of the user code implementation.

There are two basic configurations; method A uses only software plus a single port line to control the high address bit A16; method B is a combination of a small amount of hardware and software controlling the top 3 address bits A14, A15 and A16.

In the examples below, the MC68HC11G5 device is used to demonstrate the paging techniques since this device has a non-multiplexed data and address bus; any M68HC11 device may be used in a similar way.

Method A has the advantage of no additional hardware and very few limitations in the software. The user code main loop can be up to 64K bytes long and remain in the same page but this is at the expense of longer interrupt latency. The vector table and a small amount of code must be present in both pages of memory to allow correct swapping of the pages.

Method B has the advantage of not affecting the interrupt latency and has just one copy of the vector table. The maximum length of the user code main loop in this example is 48K bytes with a further 5 paged areas of 16K bytes for subroutines and tables.

METHOD A – SOFTWARE TECHNIQUE

Address A16 of the EPROM is directly controlled by port D(5) of the M68HC11 as shown in figure 1. This port is automatically configured to be in the input state following reset. It is vital that the state of the port line controlling address A16 is known following reset and so there is a 10K Ω pull-up resistor on this port line to force the A16 address bit to a logic high state following reset. This port bit is then made an output during the set-up code execution but care must be taken in ensuring that the data register is written to a logic one before the data direction register is written with a one to make the port line output a high state.

This port bit allows the M68HC11 to access the 128K byte EPROM as two memories of 64K bytes each which are paged by changing the state of the address A16 line on the EPROM. It is important to make sure that the port timing enables the port line to change state at least the setup and hold time before the address strobe (E clock rising edge on the MC68HC11G5), otherwise there could be problems with address timing.

Figure 2 shows a schematic representation of the paging technique for this method where there are two separate 64K byte pages of memory which may only be addressed individually.

This paging scheme means that code cannot directly jump from one 64K page to another without running some common area of code during the page switch. This may be accomplished in 2 basic ways. The user code could build a routine in RAM (which is common to both pages since it is internal and therefore unaffected by the port D(5) line) or have the same location in both pages devoted to a page change routine. The example software listing in appendix A uses the latter approach.

Interrupt routines

The change of page routine stores the current page before setting or clearing the port D(5) line and then has a jump command which must be at exactly the same address in both pages of memory. This is because the setting or clearing of the port D(5) line will immediately change the page of memory but the program counter will increment normally. Thus a change from page 0 to page 1 will result in the BSET PORTD command from page 0 followed by the JMP 0,X instruction from page 1 (the new page). To enable a jump to work, the X index register has been loaded with the address of the routine to be run in the new page. Figure 3 shows the execution of code to perform a change of page from page 1 to page 0.

Returning from the interrupt routine requires the RTI command to be replaced with a return from interrupt routine that checks the RAM location containing the memory page number prior to the interrupt routine execution. The routine then either performs an RTI command immediately if it is to remain in the same page or otherwise changes the state of the port D(5) line and then performs an RTI command in the correct page. Note that as with the JMP 0,X command, the RTI must be at the same address in both pages. It is important that the

I-bit in the CCR (interrupt inhibit) is set during this time for the example code to run correctly, otherwise the return page may be altered. This limitation can be overcome by using the stack to maintain a copy of the last page prior to the current interrupt.

The latency for an interrupt routine in a different page from the currently running user code is increased by 21 cycles on entering the interrupt routine and 18 cycles on leaving the interrupt routine. Any interrupt code that could not tolerate any such latency could be repeated in both pages of memory.

Other routines

Jumping from one page to another may be done at any time by using the same change of page routine but there is no need to store the current page in RAM and so these two lines of code become redundant. In the example, the change page routine could be started at the BCLR or BSET command and save 4 cycles. This would therefore reduce the page change delay to 17 cycles. Note that it is not possible to perform a JSR command to move into the other page with the method shown in the example since the RTS would not return to the original page, however, a modification to the return from interrupt routine would allow an equivalent function for a return from subroutine. In this case the stack should be used to maintain the correct return page or the I-bit in the CCR should be set to prevent interrupts.

Important conditions

The state of the port line controlling address A16 after reset is very important. In the example, port D(5) is used which is an input after reset and has a pull-up resistor to force a logic high on A16. If an output only port line was used then it could be reset such that A16 is a logic zero (no pull-up resistor required) which has an important consequence. The initialisation routine which sets up the ports must be in the default page dictated by the state of address A16 following reset otherwise the user code may not be able to correctly configure the ports and hence be unable to manipulate address A16. Similarly, a bidirectional port line could have a pull-down resistor to determine the address A16 line after reset with the same implications.

The assembler generates two blocks of code with identical address ranges used by the user code. This could not be programmed directly into an EPROM since the second page would simply attempt to overwrite the first page. The code must therefore be split into two blocks and programmed into the correct half of the EPROM. Some linkers may be capable of performing this function automatically. Figure 2 illustrates the expansion of the pages into the 128K byte EPROM memory.

The RAM and registers, and internal EEPROM if available and enabled, will all appear in the memory map in preference to external memory so care must be taken to avoid these addresses or move the RAM or registers away to different addresses by writing to the INIT register.

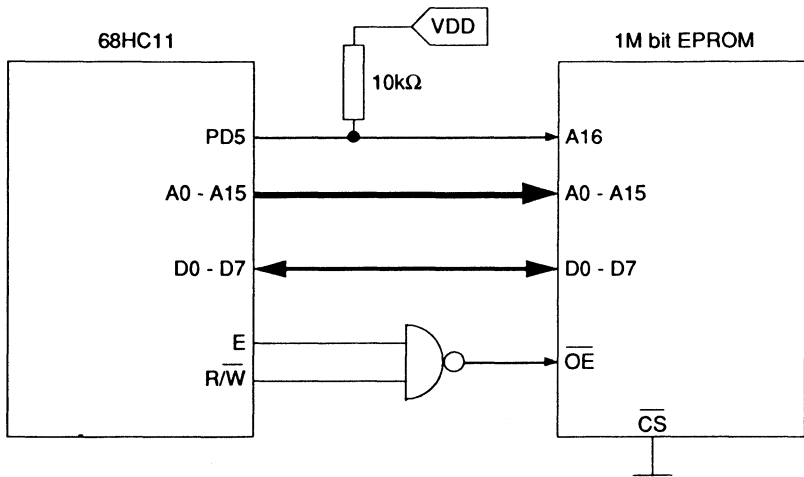


Figure 1. Software Paging Schematic Diagram

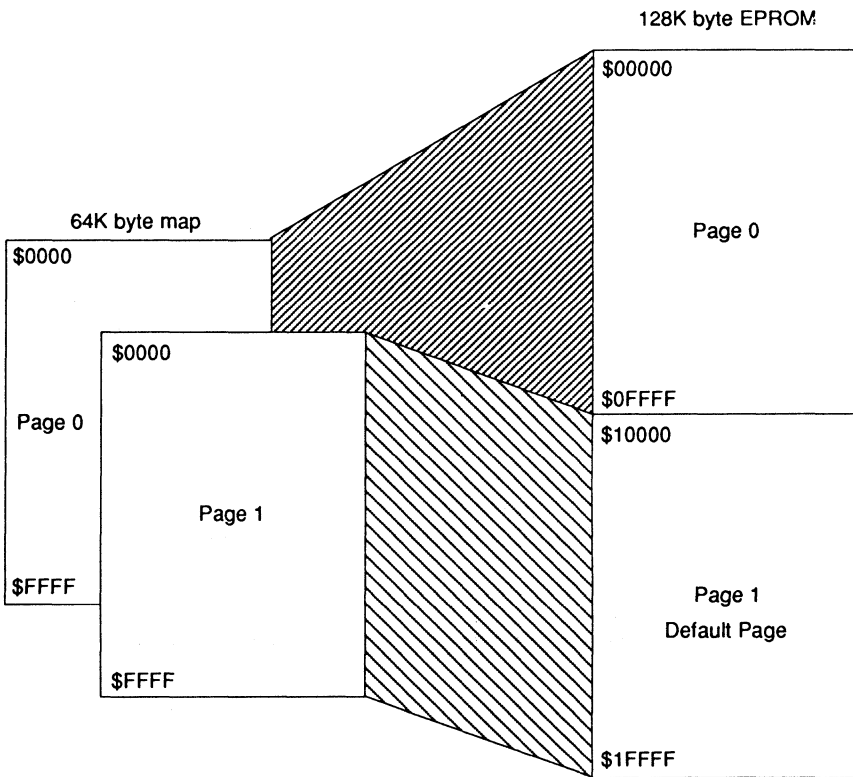
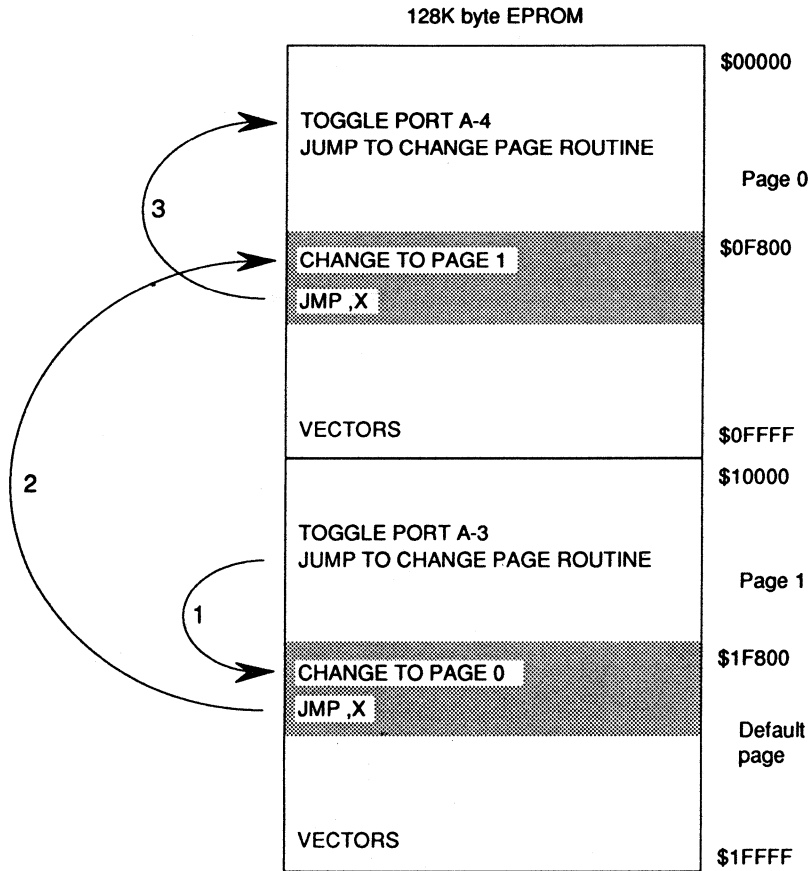


Figure 2. Software Paging Representation



- 1 - Jump to change page routine
- 2 - Page changes to page 0
- 3 - Jump to address in X register (in page 0)

Figure 3. Flow of program changing from Page 1 to Page 0

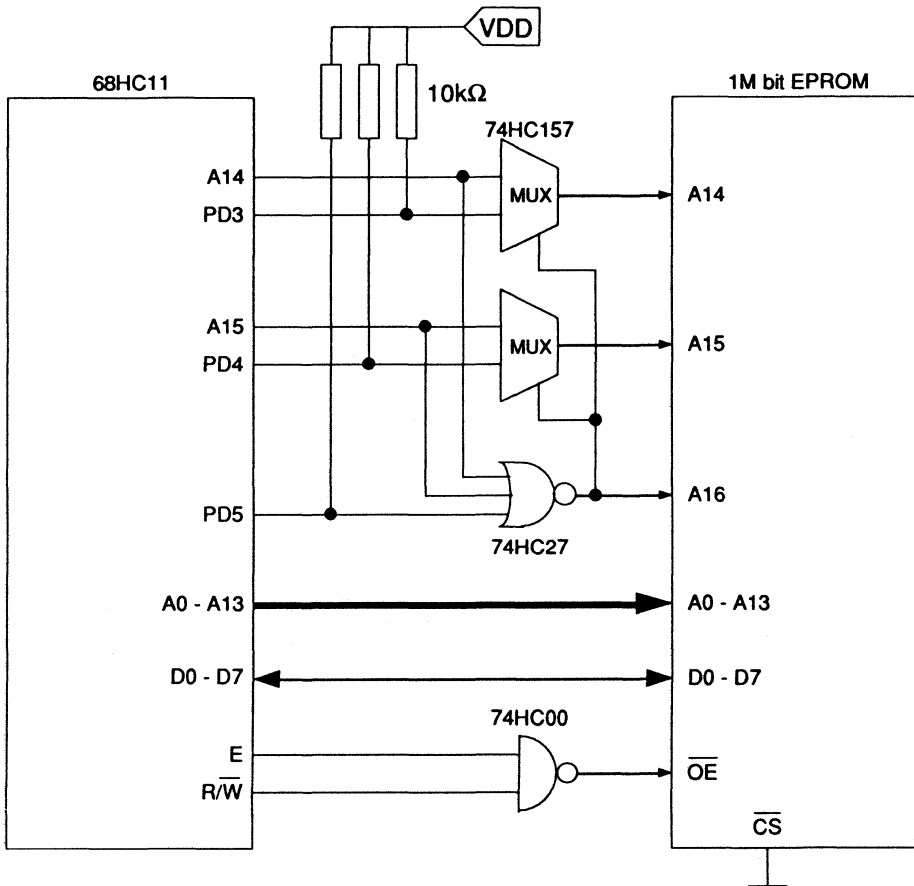


Figure 4. Hardware and Software Paging Schematic Diagram

METHOD B – COMBINED HARDWARE AND SOFTWARE TECHNIQUE

The basic approach to this method is the same as above except that hardware replaces some of the software. A port line together with M68HC11 addresses A14 and A15 are NOR'd to control the address A16 line of the EPROM. This signal is also used to select between the port line and address line for A14 and A15 (see figure 4). The hardware between the port lines controlling the A14 and A15 addresses enables 64K bytes of user code to be addressed at all times with 48K bytes common to all the pages and then selecting one of five 16K byte pages of EPROM memory.

In the example, port D(3) and address A14 are connected to the input of a 2 channel multiplexer such that port D(5), address A14 and address A15 control which of these two signals reaches the A14 pin of the EPROM. If addresses A14 or A15 are logic 1, the NOR gate outputs a logic 0 state, ensuring the A16 pin of the EPROM is a logic 0. In this case address A14 controls the A14 pin of the EPROM and similarly A15 and port D(4) are selected such that address A15 controls the A15 pin of the EPROM. Thus the main 48K byte portion of the EPROM memory may be addressed at all times at addresses \$4000 up to \$FFFF. With Port D(5) and address A14 and A15 all at logic 0 (address range \$0000 to \$3FFF), the port lines Port D(3) and Port D(4) are selected in place of address lines A14 and A15. Page 0 is always selected whenever Port D(5) is a logic 1. This makes it possible to have one of the five pages of 16 K bytes paged into the 64K addressing range of the HC11 while always maintaining the main 48K bytes of user code in the memory map.

There are few restrictions on the user code since the hardware provides the switching logic. Code can be made to run from one paged area to another by jumping to an intermediate routine in the main page. Port D is configured to be in the input state following reset which results in the main page plus page 0 of the paged memory in the 64K byte address map since the port D lines each have a pull-up resistor to maintain a logic high state after reset. A simple change memory map routine can then bring in the desired page at any time. Appendix B shows the assembly code for a program that toggles different port pins in each of the 5 pages controlled from a main routine in the main page. Figure 5 shows the 5 overlaid pages expanded to a 128K map with the flow of the program demonstrating a change from page 0 to page 1 by running the change page subroutine shown in bold type.

Implementation in 'C' language

The demonstration code was originally written in assembly language but it may also be implemented in 'C' as shown in appendix C. The change of page routines were written in 'C'

with the first part an example of using in-line code and the second part calling a function. The short example shows the assembly code on the left, generated by the 'C' code on the right. This is very similar to the assembly code example in appendix B and so it is possible to extend the memory addressing beyond 64K bytes with the 'C' language just as with assembly language.

Interrupt conditions

The interrupt routines have normal latency when they reside in the main 48K bytes page since this is always visible to the CPU. The 25 cycle delay for changing pages may cause problems for interrupt routines in a paged area of memory.

Important conditions

There are few special conditions for this method. The vectors must point to the main page of memory where the page changing routine must also reside. Routines in a paged area can only move to another page via the main 48K page unless the technique in method A is utilised (i.e. page change routine duplicated at identical addresses in both pages).

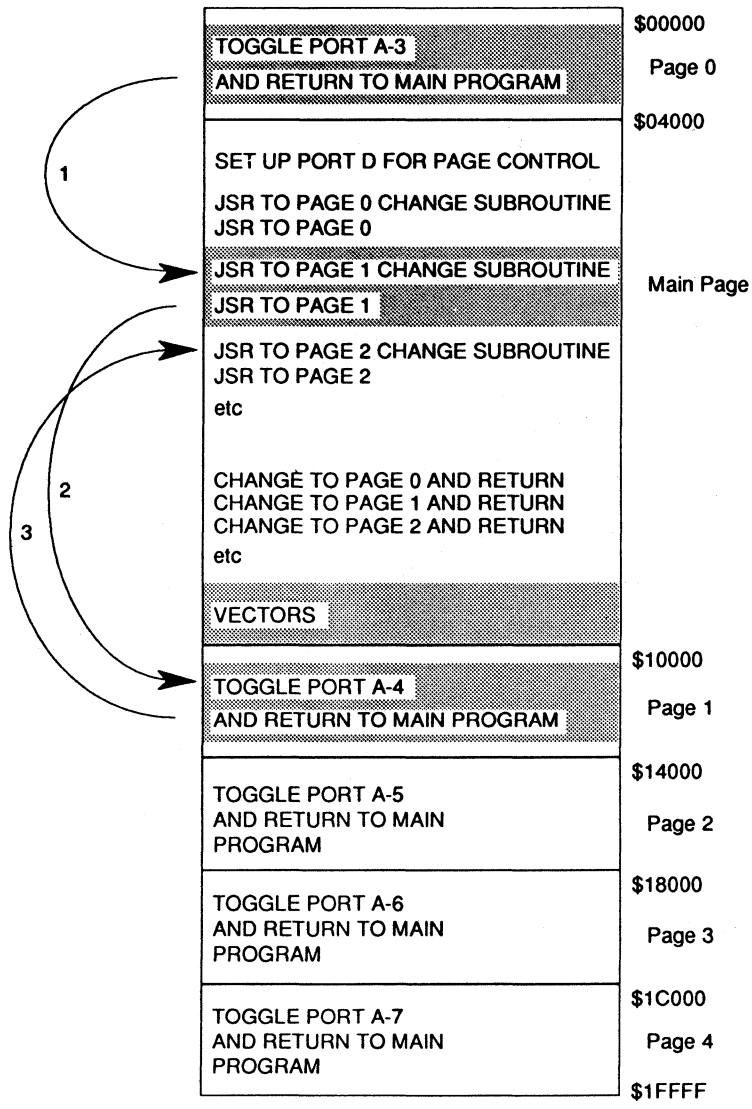
As with method A, the RAM and registers, and internal EEPROM if available and enabled, will all appear in the memory map in preference to external memory so care must be taken to avoid these addresses or move the RAM or registers away to different addresses.

The assembler generates 5 blocks of code with identical address ranges used by the user code plus the main 48K byte section. This could not be programmed directly into an EPROM since the second and subsequent pages would simply attempt to overwrite the first page. The code must therefore be split into blocks and programmed into the correct part of the EPROM. Some linkers may be capable of performing this function automatically.

Figure 6 illustrates the expansion of the pages into a single 128K byte EPROM memory.

Customisation

Clearly the size of the paged areas may be made to suit the application with for example a 32K byte main page and three 32K bytes of paged memory simply by not implementing control over the A14 address of the EPROM and not including Port D(3) control. Similarly by adding another port line to control address A13, the main program can be 56K bytes with 9 pages of 8K bytes each.



- 1 - Return from page 0
- 2 - Jump to page 1 routine
- 3 - Return from page 1 to main page

Figure 5. Illustration of changing from Page 0 to Page 1

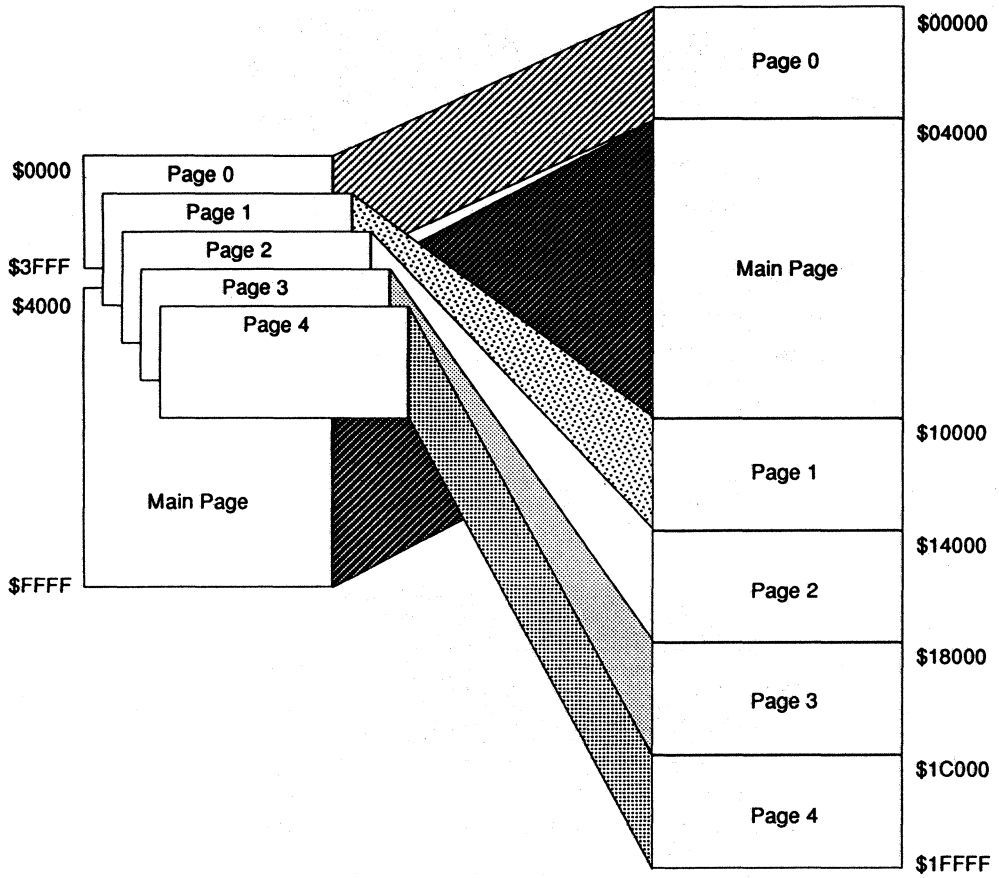


Figure 6. Hardware and software paging representation

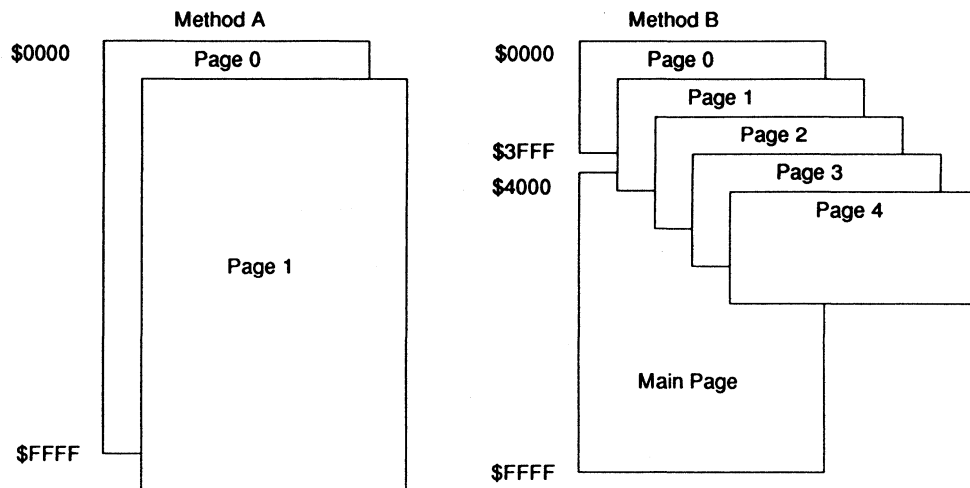


Figure 7. Comparison of paging schemes

IN GENERAL

In both methods, the registers may be moved to more appropriate addresses. If the usage of RAM is not critical the registers may be moved to address \$0000 by writing \$00 to the INIT register immediately after reset. For the MC68HC11G5 this means losing 128 bytes of RAM but results in a clean memory map above \$1FF. In the examples, the registers and RAM remain at the default addresses and so care must be taken not to have user code from address \$0000 to \$01FF and \$1000 to \$107F for the MC68HC11G5. Note that the MC68HC11E9 and MC68HC11A8 have slightly different RAM and register address ranges plus the internal EEPROM which should be disabled if not used.

Figure 7 demonstrates the differences between the paging techniques by showing the overlap of the pages. The number and size of the pages can easily be modified by small changes to the page change routines and hardware.

Beyond 128K bytes

Both techniques may be scaled up with several port lines controlling address lines beyond address A15 with the addition of further change page routines and enhancing the return from interrupt routine to allow a return to a specific page in method A or the addition of further multiplexing logic in method B.

IN CONCLUSION

The two methods described in detail are the basis for many other ways of controlling paging on a single large EPROM memory device or several smaller EPROMs. It is a simple matter to scale up or modify the techniques to suit a particular application or EPROM. The software approach is the cheapest and allows for a main program of up to the full size of the EPROM while the combined hardware and software approach has a maximum main program size of 48K bytes (in this example) and no additional interrupt latency.

APPENDIX A - SOFTWARE PAGING SCHEME

```

1      **** EXTENDA.ASC *****
2      *
3      *   TESTS EXTENDED MEMORY CONTROL
4      *
5      *   For a single 1M bit (128K byte) EPROM split into 2 x 64K byte pages.
6      *   A16 is connected to Port D(5) which then selects which half of
7      *   the EPROM is being accessed. PD5 = 1 after reset since it is in
8      *   the input state with a pull-up resistor to Vdd.
9      *
10     *   This code is written for the 68HC11G5 MCU but can be easily modified
11     *   to run on any 68HC11 device. The 68HC11G5 has a non-multiplexed
12     *   address and data bus in expanded mode.
13     *
14     *
15     *
16     *
17     *****
18     *
19     00000000    PORTA    EQU        $00
20     00000001    DDRA     EQU        $01
21     00000004    PORTB    EQU        $04
22     00000006    PORTC    EQU        $06
23     00000007    DDRC     EQU        $07
24     00000008    PORTD    EQU        $08
25     00000009    DDRD     EQU        $09
26     00000024    TMSK2    EQU        $24
27     00000025    TFLG2    EQU        $25
28     00000040    RTII     EQU        $40
29     00000040    RTIF     EQU        $40
30     00000026    PACTL    EQU        $26
31     00000080    DDRA7    EQU        $80
32     00001000    REGS     EQU        $1000
33     *
34     *****
35     *
36     *           RAM definitions (from $0000 to $01FF)
37     *
38     *****
39     *           ORG           $0000
40     00000000    PAGE        RMB        1           page number prior to interrupt
41     00000001    TIME        RMB        2           counter value for real time interrupt routine
42     *
43     00000020    NPAGE       EQU        $20           PORT D-5 page control line
44     00000200    ROMBASE     EQU        $0200        Avoid RAM (from $0 to $1FF)
45     0000f800    CHANGE      EQU        $F800
46     0000ffcc    VECTORS     EQU        $FFCC
47     *
48     *
49     *****
50     *           START OF MAIN PROGRAM
51     *
52     *
53     *           page 0 (1st half of EPROM)
54     *
55     *
56     *
57     *           org           ROMBASE
58     *
59     *
60     *           Redirect reset vector to page 1
61     *
62     *****

```

```

63 00000200 ce0200 RESETO LDX      #RESET
64 00000203 7ef800 JMP        CHGPAGE0
65
66 *****
67 *
68 *           2nd half of page 0 loop running in page 1
69 *
70 *****
71 00000206 181c0010 LOOPP0 BSET     PORTA,Y,#$10    Toggle bit 4
72 0000020a 181d0010 BCLR    PORTA,Y,#$10
73 0000020e ce0216 LDX     #LOOPP1    get return address in page 1
74 00000211 7ef800 JMP     CHGPAGE0    jump to change page routine
75 *
76 *****
77 *
78 *           Real time interrupt service routine
79 *
80 *****
81 00000214 181e254001 RTISRV BRSET    TFLG2,Y,#RTIF,RTISERV
82 00000219 3b RTI          return if not correct interrupt source
83 *           This is an RTI because interrupt vector
84 *           only points here when in page 1
85 *
86 0000021a RTISERV
87 0000021a 8640 LDAA     #$01000000    page 0 interrupt starts here
88 0000021c 18a725 STAA    TFLG2,Y        clear RTI flag
89 0000021f 9602 LDAA    TIME+1        get the time counter
90 00000221 4c INCA          increment counter
91 00000222 b71004 STAA    PORTB+REGS    store time in port B
92 00000225 de01 LDX     TIME
93 00000227 08 INX
94 00000228 df01 STX     TIME          and copy back into RAM
95 0000022a 7ef80a JMP     RETRTIO       jump to RTI routine
96 *
97 *****
98 *
99 *
100 * CHANGE PAGE ROUTINE
101 *
102 * This code must be executed with the I-bit set to prevent interrupts
103 * during the change if it is a jump for an interrupt routine.
104 * Otherwise PAGE could be updated and then another interrupt could
105 * occur before the PAGE was changed causing the first interrupt
106 * routine to return to the wrong page.
107 * The PAGE variable is not required for a normal jump and so it does
108 * not require the I-bit to be set (only the BSET is important).
109 *
110 * This code is repeated for the same position in both pages
111 *****
112 *           jump routine
113 *           ORG     CHANGE          Address for this routine is fixed
114 *                               cycles
115 0000f800 CHGPAGE0
116 0000f800 8600 LDAA     #0           2   set current page number = 0
117 0000f802 9700 STAA    PAGE         2   store page page number
118 0000f804 181c0820 BSET    PORTD,Y,#NPAGE 8   change page by setting PD-5
119 0000f808 6e00 JMP     0,X          3   This code is the same in both pages
120 *
121 *****
122 *           return from interrupt routine running in page 0
123 *
124 *
125 *           check if interrupt occurred while code was running in page 1
126 *           and return to page 1 before the RTI command is performed
127 *
128 *****

```

```

129
130 0000f80a          *          cycles
RETRTIO
131 0000f80a 9600      LDA   PAGE          2   get page the interrupt occured in
132 0000f80c 8101      CMPA  #1           2   is it page 1
133 0000f80e 2701      BEQ   RTIPAGE0     3   if yes then change page
134 0000f810 3b        RTI           12   otherwise, return from interrupt
135 0000f811          RTIPAGE0
136 0000f811 181c0820          BSET   PORTD,Y,#NPAGE 8   change page and return from interrupt
137 0000f815 3b        RTI           12   This codes is the same in both pages
138
139
140
141          *
142          *****
143          *          VECTORS
144          *          *****
145          *          *          *
146          *          *          *
147          *          *          *
148          *          *          *
149          *          *          *
150          *          *          *
151          *          *          *
152          *          *          *
153          *          *          *
154          *          *          *
155          *          *          *
156          *          *          *
157          *          *          *
158          *          *          *
159          *          *          *
160          *          *          *
161          *          *          *
162          *          *          *
163          *          *          *
164          *          *          *
165          *          *          *
166          *          *          *
167          *          *          *
168          *          *          *
169          *          *          *
170          *          *          *
171          *          *          *
172          *          *          *
173          *          *          *
174          *          *          *
175          *          *          *
176          *          *          *
177          *          *          *
178          *          *          *
179          *          *          *
180          *          *          *
181          *          *          *
182          *          *          *
183          *          *          *
184          *          *          *
185          *          *          *
186          *          *          *
187          *          *          *
188          *          *          *
189          *          *          *
190          *          *          *
191          *          *          *
192          *          *          *
193          *          *          *
194          *          *          *
195          *          *          *
196          *          *          *
197          *          *          *

```

```

198
199
200
201
202 0000021b 0f
203 0000021c 18ce1000
204 00000220 86ff
205 00000222 b71001
206 00000225 b71008
207 00000228 b71009
208 0000022b 8640
209 0000022d b71025
210 00000230 b71024
211 00000233 0e
212 00000234 39
213
214
215
216
217
218
219 00000235 181e254001
220 0000023a 3b
221
222
223
224 0000023b
225 0000023b ce021a
226 0000023e 7ef800
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246 0000f800
247 0000f800 8601
248 0000f802 9700
249 0000f804 181d0820
250 0000f808 6e00
251
252
253
254
255
256
257
258
259

*****
*
* INITIALISATION ROUTINE
*****
*
*
SETUP SEI
LDY #1000 Register address offset
LDAA #5FF
STAA DDRA+REGS make port A all outputs
STAA PORTD+REGS make sure port D-5 is written a 1
STAA DDRD+REGS and only then make all outputs
LDAA #01000000
STAA TFLG2+REGS clear RTI flag
STAA TMSK2+REGS enable RTI interrupt
CLI
RTS
*****
*
* Redirect to the Real time interrupt service routine
* Page 1 routine for service routine located in page 0
*****
*
INTRTI BRSET TFLG2,Y,#RTIF,GOODINT
RTI return if not correct interrupt source
This is an RTI because interrupt vector
only points here when in page 1
*
*
GOODINT cycles
LDX #RTISERV 3 get the interrupt entry point in page 0
JMP CHGPAGE1 3 jump to change page routine
*
*****
*
* CHANGE PAGE ROUTINE
*
* This code must be executed with the I-bit set to prevent interrupts
* during the change if it is a jump for an interrupt routine.
* Otherwise PAGE could be updated and then another interrupt could
* occur before the PAGE was changed causing the first interrupt
* routine to return to the wrong page.
* The PAGE variable is not required for a normal jump and so it does
* not require the I-bit to be set (only the BCLR is important).
*
* This code is repeated for the same position in both pages
*****
*
* jump routine
*
ORG CHANGE Address for this routine is fixed
cycles
CHGPAGE1
LDAA #1 2 set current page number = 1
STAA PAGE 2 store page page number
BCLR PORTD,Y,#NPAGE 8 change page by clearing PD-5
JMP 0,X 3 This code is the same in both pages
*
*****
*
* return from interrupt routine running in page 0
*
*
*
* check if interrupt occurred while code was running in page 1
* and return to page 0 before the RTI command is performed
*
*****

```

```

260
261 0000f80a
262 0000f80a 9600
263 0000f80c 8100
264 0000f80e 2701
265 0000f810 3b
266 0000f811
267 0000f811 181d0820
268 0000f815 3b
269
270
271
272
273
274
275
276 0000ffcc 0200
277 0000ffce 0200
278 0000ffd0 0200
279 0000ffd2 0200
280 0000ffd4 0200
281 0000ffd6 0200
282 0000ffd8 0200
283 0000ffda 0200
284 0000ffdc 0200
285 0000ffde 0200
286 0000ffe0 0200
287 0000ffe2 0200
288 0000ffe4 0200
289 0000ffe6 0200
290 0000ffe8 0200
291 0000ffea 0200
292 0000ffec 0200
293 0000ffee 0200
294 0000fff0 0235
295 0000fff2 0200
296 0000fff4 0200
297 0000fff6 0200
298 0000fff8 0200
299 0000fffa 0200
300 0000fffc 0200
301 0000fffe 0200
302
303

```

```

*
RETRTI1
LDAA PAGE 2 get page the interrupt occurred in
CMPA #0 2 is it page 0
BEQ RTIPAGE1 3 if yes then change page
RTI 12 otherwise, return from interrupt
RTIPAGE1
BCLR PORTD,Y,#NPAGE 8 change page and return from interrupt
RTI 12 This codes is the same in both pages
*
*****
* VECTORS
*****
*
ORG VECTORS
FDB RESET EVENT 2
FDB RESET EVENT 1
FDB RESET TIMER OVERFLOW 2
FDB RESET INPUT CAPTURE 6 / OUTPUT COMPARE 7
FDB RESET INPUT CAPTURE 5 / OUTPUT COMPARE 6
FDB RESET SCI
FDB RESET SPI
FDB RESET PULSE ACC INPUT
FDB RESET PULSE ACC OVERFLOW
FDB RESET TIMER OVERFLOW 1
FDB RESET INPUT CAPTURE 4 / OUTPUT COMPARE 5
FDB RESET OUTPUT COMPARE 4
FDB RESET OUTPUT COMPARE 3
FDB RESET OUTPUT COMPARE 2
FDB RESET OUTPUT COMPARE 1
FDB RESET INPUT CAPTURE 3
FDB RESET INPUT CAPTURE 2
FDB RESET INPUT CAPTURE 1
FDB INTRTI REAL TIME INTRRUPT
FDB RESET IRQ
FDB RESET XIRQ
FDB RESET SWI
FDB RESET ILLEGAL OPCODE
FDB RESET COP
FDB RESET CLOCK MONITOR
FDB RESET RESET
*****
END

```

APPENDIX B – HARDWARE AND SOFTWARE PAGING SCHEME

```

1          ***** EXTENDB.ASC *****
2          *
3          * TESTS EXTENDED MEMORY CONTROL
4          *
5          * for a single 1M bit (128K byte) EEPROM split into 48KB + 5 x 16KB
6          * $4000 - $FFFF    48K    COMMON PAGE
7          * $0200 - $3FFF    16K    PAGES 0,1,2,3,4
8          *
9          * A multiplexer is used to switch between address and port D lines
10         * controlled by PD5 and A16 is controlled by /(PD5+A14+A15)
11         * This ensures that Address A16 is a logic 1 whenever A14 or A15 are
12         * high and that all three lines must be low for the paged memory between
13         * addresses $00000 and $0FFFF.
14         *
15         *
16         *          SOURCE CODE          EPROM
17         *          ADDRESS          ADDRESS
18         *          0000          +-----+ 00000
19         *          |          PAGE 0  |
20         *          4000          +-----+ 04000
21         *          |          |
22         *          |          MAIN PAGE |
23         *          |          |
24         *          |          |
25         *          0000          +-----+ 10000
26         *          |          PAGE 1  |
27         *          0000          +-----+ 14000
28         *          |          PAGE 2  |
29         *          0000          +-----+ 18000
30         *          |          PAGE 3  |
31         *          0000          +-----+ 1C000
32         *          |          PAGE 4  |
33         *          3FFF          +-----+ 1FFFF
34         *

```

(Continued overleaf)


```

81
82 00000000      *
83 00000001      PORTA    EQU    $00
84 00000004      DDRA     EQU    $01          68HC11G5 only
85 00000006      PORTB    EQU    $04
86 00000007      PORTC    EQU    $06
87 00000008      DDRC     EQU    $07
88 00000009      PORTD    EQU    $08
89 00000024      DDRD     EQU    $09
90 00000025      TMSK2    EQU    $24
91 00000040      TFLG2    EQU    $25
92 00000040      RTII     EQU    $40
93 00000026      RTIF     EQU    $40
94 00000080      PACTL    EQU    $26
95 00001000      DDRA7    EQU    $80          68HC11E9 only
96 00001000      REGS     EQU    $1000
97
98
99
100
101
102
103 00000000      *
104
105 00000200      *          RAM definitions
106 00004000      *
107 0000ffcc      *
108
109
110
111
112
113
114
115
116
117
118
119
120 00000000      *          *****
121 00000020      *          PAGE 0 = $00000 - $03FFF (A16=0,A15=0,A14=0) => PAGE0=%00100000
122 00000000      *          MAIN   = $04000 - $0FFFF (A16=0)           => START=%001XX000
123 00000008      *          PAGE 1 = $10000 - $13FFF (A16=1,A15=0,A14=0) => PAGE1=%00000000
124 00000010      *          PAGE 2 = $14000 - $17FFF (A16=1,A15=0,A14=1) => PAGE2=%00001000
125 00000018      *          PAGE 3 = $18000 - $1BFFF (A16=1,A15=1,A14=0) => PAGE3=%00010000
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

128 *+++++*
129 *
130 *       page 0 (1st half of EPROM)
131 *
132 *
133 *+++++*
134       org       ROMBASE0
135 00000200 181c0008 LOOPP0 BSET   PORTA,Y,#$08
136 00000204 181d0008       BCLR   PORTA,Y,$$08       Toggle Port A-3
137 00000208 7e4014       JMP     MAIN0           return to main page
138 *
139 *+++++*
140 *       START OF MAIN PROGRAM
141 *+++++*
142 *
143 *       MAIN ROUTINE NOT UNDER INTERRUPT CONTROL
144 *
145 *+++++*
146 *
147       org       ROMBASE1
148 00004000 8e01ff RESET LDS     #$01FF
149 00004003 bd402e       JSR     SETUP           initialise RTI interrupt and DDRs
150 00004006 181c0840 LOOP  BSET   PORTD,Y,$$40
151 0000400a 181d0840       BCLR   PORTD,Y,$$40       main routine toggles port D-2
152 0000400e bd4062       JSR     CHGPAGE0        select page 0
153 00004011 7e0200       JMP     LOOPP0          Toggle Port A-3
154 00004014 bd406d MAIN0 JSR     CHGPAGE1        select page 1
155 00004017 bd0200       JSR     LOOPP1          Toggle Port A-4
156 0000401a bd4078       JSR     CHGPAGE2        select page 2
157 0000401d bd0200       JSR     LOOPP2          Toggle Port A-5
158 00004020 bd4083       JSR     CHGPAGE3        select page 3
159 00004023 7e0200       JMP     LOOPP3          Toggle Port A-6
160 00004026 bd408e MAIN3 JSR     CHGPAGE4        select page 4
161 00004029 7e0200       JMP     LOOPP4          Toggle Port A-7
162 0000402c 20d8 MAIN4 BRA     LOOP           start loop again
163 *
164 *+++++*
165 *       INITIALISATION ROUTINE
166 *+++++*
167 *
168 0000402e 0f SETUP SEI
169 0000402f 18ce1000 LDY     #$1000       Register address offset
170 00004033 86ff LDAA   #$FF
171 00004035 b71001 STAA  DDRA+REGS     make port A all outputs (68HC11G5)
172 00004038 b71009 STAA  DDRD+REGS     make port D all outputs
173 0000403b 7f0000 CLR   TIME
174 0000403e 7f0001 CLR   TIME+1
175 00004041 4f CLRA
176 00004042 b71000 STAA  PORTA+REGS
177 00004045 b71008 STAA  PORTD+REGS
178 00004048 8640 LDAA  ##01000000
179 0000404a b71025 STAA  TFLG2+REGS     clear the RTI flag
180 0000404d b71024 STAA  TMSK2+REGS     enable RTI interrupt
181 00004050 0e CLI
182 00004051 39 RTS
183 *

```

184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228

```

*****
*
*      Real time interrupt service routine
*
*****
RTISRV LDAA    #01000000
          STAA  TFLG2+REGS      clear RTI flag
          LDAA  TIME+1
          STAA  PORTB+REGS      store counter in port B
          LDX   TIME            get time counter
          INX   increment counter
          STX   TIME            save counter value in RAM
          RTI   Return from interrupt
*
*****
* CHANGE PAGE
* acc B (bits 3-5) contains the 1's complement of new page number address
*
* SOURCE CODE                                EPROM
* ADDRESS                                    ADDRESS
* 0000 +-----+ 00000
* |                |
* |          PAGE 0          |
* |                |
* 4000 +-----+ 04000
* |                |
* |          MAIN PAGE          |
* |                |
* |                |
* 0000 +-----+ 10000
* |                |
* |          PAGE 1          |
* |                |
* 0000 +-----+ 14000
* |                |
* |          PAGE 2          |
* |                |
* 0000 +-----+ 18000
* |                |
* |          PAGE 3          |
* |                |
* 0000 +-----+ 1C000
* |                |
* |          PAGE 4          |
* |                |
* 3FFF +-----+ 1FFFF
*
* PAGE 0 = $0000 - $03FFF (A16=0,A15=0,A14=0) => PAGE0=$00100000
* MAIN   = $04000 - $0FFFF (A16=0)           => START=$001XX000
* PAGE 1 = $10000 - $13FFF (A16=1,A15=0,A14=0) => PAGE1=$00000000
* PAGE 2 = $14000 - $17FFF (A16=1,A15=0,A14=1) => PAGE2=$00001000
* PAGE 3 = $18000 - $1BFFF (A16=1,A15=1,A14=0) => PAGE3=$00010000
* PAGE 4 = $1C000 - $1FFFF (A16=1,A15=1,A14=1) => PAGE4=$00011000
*
*****

```

```

229
230 00004062
231 00004062 b61008
232 00004065 84c7
233 00004067 8b20
234 00004069 b71008
235 0000406c 39
236
237 0000406d
238 0000406d b61008
239 00004070 84c7
240 00004072 8b00
241 00004074 b71008
242 00004077 39
243
244 00004078
245 00004078 b61008
246 0000407b 84c7
247 0000407d 8b08
248 0000407f b71008
249 00004082 39
250
251 00004083
252 00004083 b61008
253 00004086 84c7
254 00004088 8b10
255 0000408a b71008
256 0000408d 39
257
258 0000408e
259 0000408e b61008
260 00004091 84c7
261 00004093 8b18
262 00004095 b71008
263 00004098 39
264
265
266
267
268
269
270 0000ffcc 4000
271 0000ffce 4000
272 0000ffd0 4000
273 0000ffd2 4000
274 0000ffd4 4000
275 0000ffd6 4000
276 0000ffd8 4000
277 0000ffda 4000
278 0000ffdc 4000
279 0000ffde 4000
280 0000ffe0 4000
281 0000ffe2 4000
282 0000ffe4 4000
283 0000ffe6 4000

*
CHGPAGE0
LDAA PORTD+REGS get port D data
ANDAA ##11000111 make middle 3 bits low state
ADDA #PAGE0 add PAGE descriptor to this
STAA PORTD+REGS write back to port D
RTS (only bits 3, 4 and 5 are changed)
*
CHGPAGE1
LDAA PORTD+REGS get port D data
ANDAA ##11000111 make middle 3 bits low state
ADDA #PAGE1 add PAGE descriptor to this
STAA PORTD+REGS write back to port D
RTS (only bits 3, 4 and 5 are changed)
*
CHGPAGE2
LDAA PORTD+REGS get port D data
ANDAA ##11000111 make middle 3 bits low state
ADDA #PAGE2 add PAGE descriptor to this
STAA PORTD+REGS write back to port D
RTS (only bits 3, 4 and 5 are changed)
*
CHGPAGE3
LDAA PORTD+REGS get port D data
ANDAA ##11000111 make middle 3 bits low state
ADDA #PAGE3 add PAGE descriptor to this
STAA PORTD+REGS write back to port D
RTS (only bits 3, 4 and 5 are changed)
*
CHGPAGE4
LDAA PORTD+REGS get port D data
ANDAA ##11000111 make middle 3 bits low state
ADDA #PAGE4 add PAGE descriptor to this
STAA PORTD+REGS write back to port D
RTS (only bits 3, 4 and 5 are changed)
*
*****
* VECTORS
*****
*
ORG VECTORS
FDB RESET EVENT 2
FDB RESET EVENT 1
FDB RESET TIMER OVERFLOW 2
FDB RESET INPUT CAPTURE 6 / OUTPUT COMPARE 7
FDB RESET INPUT CAPTURE 5 / OUTPUT COMPARE 6
FDB RESET SCI
FDB RESET SPI
FDB RESET PULSE ACC INPUT
FDB RESET PULSE ACC OVERFLOW
FDB RESET TIMER OVERFLOW 1
FDB RESET INPUT CAPTURE 4 / OUTPUT COMPARE 5
FDB RESET OUTPUT COMPARE 4
FDB RESET OUTPUT COMPARE 3
FDB RESET OUTPUT COMPARE 2

```

```

284 0000ffe8 4000      FDB      RESET      OUTPUT COMPARE 1
285 0000ffea 4000      FDB      RESET      INPUT CAPTURE 3
286 0000ffec 4000      FDB      RESET      INPUT CAPTURE 2
287 0000ffee 4000      FDB      RESET      INPUT CAPTURE 1
288 0000fff0 4052      FDB      RTISRV     REAL TIME INTRRUPT
289 0000fff2 4000      FDB      RESET      IRQ
290 0000fff4 4000      FDB      RESET      XIRQ
291 0000fff6 4000      FDB      RESET      SWI
292 0000fff8 4000      FDB      RESET      ILLEGAL OPCODE
293 0000fffa 4000      FDB      RESET      COP
294 0000fffc 4000      FDB      RESET      CLOCK MONITOR
295 0000fffe 4000      FDB      RESET      RESET
296
297
298
299
300
301
302
303
304
305 00000200 181c0010  LOOPP1 BSET      PORTA,Y,#$10
306 00000204 181d0010      BCLR      PORTA,Y,#$10      Toggle Port A-4
307 00000208 39          RTS
308
309
310
311
312
313
314
315 00000200 181c0020  LOOPP2 BSET      PORTA,Y,#$20
316 00000204 181d0020      BCLR      PORTA,Y,#$20      Toggle Port A-5
317 00000208 39          RTS
318
319
320
321
322
323
324
325 00000200 181c0040  LOOPP3 BSET      PORTA,Y,#$40
326 00000204 181d0040      BCLR      PORTA,Y,#$40      Toggle Port A-6
327 00000208 7e4026      JMP      MAIN3      return to main page
328
329
330
331
332
333
334
335 00000200 181c0080  LOOPP4 BSET      PORTA,Y,#$80
336 00000204 181d0080      BCLR      PORTA,Y,#$80      Toggle Port A-7
337 00000208 7e402c      JMP      MAIN4      return to main page
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400

```

APPENDIX C - 'C' LANGUAGE ROUTINES FOR METHOD B

```

*      /* CHGPAGE.C
*      C coded extended memory control for 68HC11
*
*      */
*
*****
/*      HC11 structure - I/O registers for MC68HC11 */

struct HC11IO {
  unsigned char  PORTA;      /* Port A - 3 input only, 5 output only */
  unsigned char  Reserved;  /* Motorola's unknown register */
  unsigned char  PIOC;      /* Parallel I/O control */
  unsigned char  PORTC;     /* Port C */
  unsigned char  PORTB;     /* Port B - Output only */
  unsigned char  PORTCL;    /* Alternate port C latch */
  unsigned char  Reserved1; /* Motorola's unknown register 2 */
  unsigned char  DDRC;      /* Data direction for port C */
  unsigned char  PORTD;     /* Port D */
  unsigned char  DDRD;      /* Data direction for port D */
  unsigned char  PORTE;     /* Port E */
};
/*      End of structure HC11IO */
*****
*
*      #define regbase (*(struct HC11IO *) 0x1000)
*      typedef unsigned char byte;
*
*      /* Some arbitrary user defined values */
*      #define page0 0x20
*      #define page1 0x00
*      #define page2 0x08
*      #define pagemask 0xc7
*
*      /* Macro to generate in line code */
*      #define chgpage(a) regbase.PORTD = (regbase.PORTD & pagemask) + a
*
*      /* Function prototype */
*      void func_chgpage(byte p);
*
*      /* Externally defined functions in separate pages */
*      extern void func_in_page0(); /* Dummy function in page 0 */
*      extern void func_in_page2(); /* Dummy function in page 2 */
*
*

```

```

* ----- compiled assembly code ----- C source code -----
*
*
*          main: fbegin                                main()
6 0000
*
*          ldab    $1008                               {
8 0000 f61008          andb    #199                        chgpage(page2);
9 0003 c4c7          addb    #8                          /* Change page using inline code */
10 0005 cb08          stab    $1008
11 0007 f71008
*
*          jsr     func_in_page2                       func_in_page2();
13 000a >bd0000
*
*          ldd     #32                                 func_chgpage(page0);
15 000d cc0020          bsr     func_chgpage                          /* Change page using function call */
16 0010 8d04
*
*          jsr     func_in_page0                       func_in_page0();
18 0012 >bd0000
*
*          rts                                         }
20 0015 39
21 0016          fend
*
*
*          func_chgpage: fbegin                       void func_chgpage(p)
24 0016          pshb                                byte p;
25 0016 37
*
*          ldab    $1008                               {
27 0017 f61008          andb    #199
28 001a c4c7          tsx
29 001c 30            addb    0,x
30 001d eb00          stab    $1008
31 001f f71008
*
*          ins
33 0022 31
34 0023 39          rts
35 0024          fend
36          import func_in_page2
37          import func_in_page0
38          end

```


TV on-screen display using the MC68HC05T1

By Peter Topping
Motorola Ltd., East Kilbride, Scotland

INTRODUCTION

The "T" members of the MC68HC05 family of MCUs provide a convenient and cost effective method of adding on-screen display (OSD) to TVs and VCRs. As well as the OSD capability, they include 8 Kbytes of ROM (adequate for Teletext, frequency-synthesis, stereo and OSD), 320 bytes of RAM, a 16-bit timer and 8 pulse-width-modulated D/A converters. The MC68HC(7)05T7/8 also includes IIC hardware and, by using a 56/64 pin package, 4 ports of I/O independent of the OSD, serial and D/A outputs. It is thus suitable for large full-

feature chassis. The MC68HC05T1 is in the middle of the price/performance range and includes most of the features of the MC68HC05T8 but in a 40-pin package. This is achieved by sharing I/O with the other pin functions (SPI, OSD, D/A). Even if all these features are used there is sufficient I/O for most applications. The low cost MC68HC05T4 has 5 Kbytes of ROM and 96 bytes of RAM making it suitable for simpler (mono, non-Teletext) applications.

68HC05T1 OSD FEATURES

- Programmable display of 10 rows of 18 characters
- 24 byte (18 data + 6 control) single row architecture
- Settable in software to any one of four standards
- Zero inter-row and inter-column spacing
- 64 user-defined mask-programmable 8 x 13 characters
- Programmable horizontal position
- Character colour selectable from 4 colours/row
- Software programmable (start, stop and colour) window
- 4 character sizes (normal, double height and/or width)
- Half-dot character rounding
- Selectable half-dot black outline.

OSD CHARACTERISTICS

The HC05TX series have an OSD capability of 10 rows of 18 characters. Each row can contain characters of four colours selected from the eight available colours (black, blue, green, cyan, red, magenta, yellow and white). The rows can independently select double height and/or double width and the start and stop positions of a background window of any colour. The signals sent to the TV are Red, Green, Blue, fast-blanking and half-tone. Separate horizontal and vertical synchronisation inputs are required.

The OSD architecture employed includes only a single line of display RAM. This makes the software more complicated but reduces the silicon area required to implement the OSD function. The software is required to update the display RAM on a regular basis. When operating in the 625-line PAL standard the updates must occur at 1.66 ms (26 lines) intervals in order to display adjacent lines. The OSD hardware can generate an interrupt when an update is required. There

are 18 data registers (one for each character) and 6 control registers arranged as shown below. The table is for the T1, some of the control bits are different in the T4/7/8.

\$20-\$31	OSD	Data registers.
\$32	CAS	Read: status, Write: colours 1 & 2 and outline enable.
\$33	C34	Colours 3 & 4.
\$34	RAD	Row address, character size, int. enable, RGB invert.
\$35	WCR	OSD & PLL enable, Window enable and start column.
\$36	CCP	Window colour and end column.
\$37	HPD	Horizontal position, standard selection.

The OSD display is timed from an on-chip 14 MHz oscillator which is phase locked to the TV's line synchronisation pulses. The vertical synchronisation depends on the standard in use. Four standards are available (15.75 kHz/60 Hz, 31.5 kHz/120 Hz, 15.625 kHz/50 Hz and 31.25 kHz/100 Hz). The standard is selected by control bits in the T1/2 but is automatic in the T4 and the T7/8.

64 OSD characters are mask programmed along with the user ROM. The spacing and full size of the characters is the same at

8 x 13 (for 625-line standard). This allows continuous graphics. Half-dot interpolation hardware doubles the apparent resolution to produce smoother characters. A software selectable black outline (a half-dot wide) is also implemented in the hardware. Because the half-dot circuitry has to know the information for the next line of pixels, a 14th line is available in the character generator ROM to facilitate look ahead. The vertical height of a character is 26 lines (52 including interlace) and the horizontal width is $2\frac{2}{7}\mu\text{s}$ ($1\frac{1}{7}\mu\text{s}$ per half pixel).

SOFTWARE

There are several approaches to writing OSD software to operate with the single line architecture. The choice will affect the amount of ROM and RAM used. One principle is to have a separate interrupt routine for each type of row to be displayed. This method will use little RAM but will be inefficient in its use of ROM. The other approach is to write a single interrupt routine which transfers display information from a block of normal RAM to the display RAM as it is required for each new line. This method will be more ROM efficient but requires a RAM location for every display character. The amount of RAM used depends on the maximum amount of data which has to be displayed at any one time. The choice between these two methods will depend on the type of data to be displayed. The first method may be better if much of the displayed data is fixed. This could be, for example, a series of menus. The second method will however be more appropriate if the data is mostly variable. This will usually be the case in conventional TV applications.

This application note describes an implementation of the second of the above approaches. A block of RAM is used to contain a copy of all the data to be displayed. The size of this block can be changed to reflect the number of rows and the number of characters per row. The choice made in the example described here is 8 rows of 16 characters. This is slightly less

than the maximum available and was chosen because the total number of characters (128) corresponds to the available page 1 RAM in the MC68HC05T1. The choice of 16 characters per row also slightly simplifies the software. The software allows any eight of the ten available rows to be used but only the first 16 of the 18 available characters. This choice does not prevent access to the right-hand-side of the screen as the display can be moved to the right under software control. The use of page 1 for the RAM does not incur any significant compromise in execution time. It also leaves free the page 0 RAM for the rest of the TV control software, which would be made less efficient if it had to use page 1 RAM, where direct addressing and bit manipulation instructions cannot be used. This choice slightly increases the ROM used by the OSD code, as 3-byte extended store instructions sometimes need to be used to write data to the RAM used for OSD characters.

The 1-byte indexed addressing mode can however be used in page 1. This addressing mode can access up to address \$1FE and is made use of in the example software. For example the OSDCLR routine used to initialise RAM locations used for OSD employs a CLR DRAM-1,X instruction. DRAM is the start of page 1 RAM at \$100 so DRAM-1 evaluates as \$FF a 1-byte offset.

INTERRUPT ROUTINE

The OSD update interrupt routine (NLINE) shown in the program listing transfers data from page 1 RAM to display RAM each time an interrupt occurs. The first operation is to increment the pointer which selects the next row number. This pointer (OSDL) is subsequently used to transfer the appropriate data from page 1 RAM to the OSD RAM. So that any row number can be used the pointer selects the number from a table unique to each type of display. The appropriate table is determined by the value of LIND. The pointer is incremented until the corresponding row number is zero when the pointer is reset to zero. This allows any sub-set of up to 8 of the 10 available rows to be used. The next row number (ORed with the character size information contained in RAM) is written into the appropriate register (\$34). The row number in this register is compared by the OSD hardware with the current position of the raster. When they match, an interrupt is generated and the next interrupt routine is performed. The other control registers are then updated from the page 0 RAM locations, which are used for this purpose.

To save RAM only three (RAD, CAS_ & CCR) of the six control registers are loaded in this way. The pointer OSDL is multiplied by 3 using the table M3, as this is quicker than shifting and adding. In this example the other registers are loaded by the main program and therefore have fixed values for each display. The fixed registers are Colour 3/4 (\$33), Window enable/start column (\$35) and Horizontal position delay (\$37). As this choice would not allow windows to be enabled on individual rows, window enable is controlled by the un-used bit (6) in the RAM byte used to update the Colour 1/2 register (CAS). This choice of fixed registers limits the flexibility of the display but clearly all registers can be updated on a line-by-line basis if more RAM is used. The limitations imposed by this choice are that colours 3 & 4, the window start column and the horizontal position apply to a whole display rather than to individual lines. In practice these constraints were not found to be significant restrictions for the displays required for TV use.

The interrupt routine then transfers the relevant OSD data from page 1 RAM into the OSD data registers. This is done using linear, repetitive code in order to minimise the time taken by the interrupt routine. The code used uses 8 cycles (4 μ s) for each byte transferred. Less ROM space would be utilised if a loop was employed but this would use 28 cycles per byte. The best choice depends on whether time or ROM use is more critical. The example code includes a cycle count to calculate the length of the interrupt routine. The time taken is $121 \pm 4 \mu$ s. This includes the time taken by the interrupt itself. An alternative method of OSD data transfer (TOSD2) using a loop is

included as comments in the listing. It would take an additional 165 μ s.

The last task performed by the interrupt routine is to control any character or window flashing. The software allows one or two characters (on a selected row) and one window (on the same or a different row) to be flashed at a rate determined by the MCU's timer. This function could be performed outwith the interrupt routine in the main program and the time taken to perform it is not included in the figure given above.

MAIN OSD PROGRAM

The remainder of the OSD control program does not write directly to most of the display registers. It simply puts the required display and control information into the blocks of RAM allocated for this purpose, together with supplying the co-ordinates of any required flashing characters or windows. It must, however, write to the display control registers not updated by the interrupt routine; in this example these are \$33, \$35 and \$37. The program has 4 main parts. These are the idle, channel name table, program/channel number and analogue displays. The idle display applies when no transient display (eg program number and channel number or name) is on. The OSD idle condition is selectable between blank and a small program number at the bottom right hand corner of the screen.

The OSD example program (assembler listing included) is just part of the code required to control a TV set. This program was incorporated in HC05T1 software along with four other modules. These were the base module (idle loop, transient control, local keyboard, IR, IIC and reset), the tuning module (PLL, analogue and NVM control), the stereo module (stereoton and Nicam) and the Teletext module (FLOF level 1.5).

The microprocessor in a TV application will usually need to handle the reception of IR commands. Polled methods of IR reception are most effective if the time made unavailable to them by interrupts is minimised. It is for this reason that the illustrated OSD interrupt routine was written to execute as fast as possible. This is, however, not so much of a problem if the TCAP facility is used for IR reception. When a falling edge occurs, the timer value is saved and it does not matter if the interrupt which processes this information is not serviced until several hundred microseconds later. The allowable size of this delay will of course depend on the IR protocol in use. The bi-phase protocol used with the example OSD software (transmitter chip: MC144105) has a minimum spacing of 1 ms between consecutive edges.

The next section describes the OSD features of this software. Some of the data used in the OSD is passed from other modules (particularly the tuning module). The same RAM allocation file was used in all modules so this part of the listing shows the locations used to pass data between them.

OSD FEATURES PROVIDED IN THE EXAMPLE PROGRAM

Program change

When keys 0-9, PC- or PC+ are pressed, the new program number appears (in cyan) at the bottom-right-hand corner of the screen in double height/double width characters and stays for 5 seconds after the last change. Above this display either the channel name (if one has been defined) or the channel number is shown (normal size). After 5 seconds this display times out and there is either no display or a permanent normal size program number display. This is selectable using the Teletext MIX key.

For program numbers of 10 and over, three keys are required. They are selected by first pressing "—". Two flashing dashes are displayed, the first 0-9 key (only 0-4 valid) will be taken as the tens digit and the second as the units digit. If a new program number has not been selected within 30 seconds, the TV returns to the previous display (nothing or the old program number).

Channel mode

When the P/C key is pressed, the program number and channel name (or number) is displayed for 5 seconds as an indication of the current status. If it is pressed again during this period, the TV changes to channel mode. This will remain for 30 seconds after the last key-press. The display (in yellow) shows the program number as in program mode along with the channel number. The channel number flashes to show that it will be changed if a number or PC- or PC+ key is pressed. New channels can be selected. If the STORE key is pressed then the current channel is stored against the current program number. If no key is pressed for 30 seconds, the TV returns to program mode. If the channel has been changed but STORE not pressed then the TV will retune back to the channel stored against the current program number.

Automatic search

When SEARCH is pressed the TV goes into the channel mode and the on screen display is as described above. The channel number is incremented at a rate of 2 per second until a signal is found. The search then stops. A press of STORE returns the TV to program mode, storing the new channel against the current program number.

Analogues

When any of the analogues are selected the appropriate logo is displayed along with a horizontal bar indicating the current value in the D/A convertor (full-scale 63). Display returns to default (nothing or program number) 5 seconds after the last change. If no analogue is selected the volume is shown (and adjusted) when the ANALOGUE +/- keys are used.

Channel name table

Up to 24 channels can have a 4 character name and standard bit associated with them. If the channel number and standard of one of these entries in the table correspond to those selected by the current program number then the name is displayed along with the program number when the program is selected or when P/C is pressed. Entry of names is done using the Teletext INDEX key. When it is pressed a table of six

lines is displayed. Each line (identified by a "station" number in the leftmost column) contains a channel number, standard and the associated name. All of this data is user definable.

One character on the screen flashes to indicate the current position of the cursor. The character at the cursor position can be changed through 0-9, A-Z and space by pressing PC+ or PC- (0-9 for channel number digits and PAL/SECAM for standard). When a character (or the standard) is changed, its colour changes from yellow to red. The cursor is moved to the left and right by the Teletext RED and GREEN keys and up and down by the BLUE and YELLOW keys. The current line appears in a light blue (cyan) window as opposed to the dark blue window used for the other lines. The whole table scrolls when the cursor is required to go beyond the bottom (or top) of the current display.

To save a name the STORE key is pressed. This will save the name and standard on the current line against its channel number. This is indicated by the colour of any changed characters returning to yellow. Any changes which have been made to lines other than the one being stored are lost. Channel 00 cannot have a name. The procedure for removing a name from the table is to set the channel number to zero and then to save the line. Any name left on the line will not be used. The table display is exited by pressing the Teletext INDEX key. The function of each key is shown at the bottom of the display.


```

23 *****
23 *
23 *      General RAM allocation.
23 *
23 *****
23
23 00000049 PLLHI RMB 1 PLL DIVIDE RATIO MSB
23 0000004a PLLOW RMB 1 PLL DIVIDE RATIO LSB
23 0000004b W1 RMB 1 WORKING
23 0000004c W2 RMB 1 "
23 0000004d W3 RMB 1 "
23 0000004e COUNT RMB 1 LOOP COUNTER
23 0000004f KOUNT RMB 1 LOCAL KEYBOARD COUNTER
23 00000050 CNT RMB 1 12.8mS (inc, free running)
23 00000051 CNT1 RMB 1 12.8mS (inc, reset every 1S during transient)
23 *CNT2 RMB 1 3.25 S (inc, store timeout)
23 CNT3 RMB 1 3.25 S (dec, automatic standby timeout)
23 00000053 CNT4 RMB 1 12.8mS (cleared for row24 delay when page arrives)
23 00000054 CNT5 RMB 1 12.8mS (inc, transient mute)
23 00000055 TMR RMB 1 TRANSIENT DISPLAY SECONDS COUNTER
23 00000056 STAT RMB 1 0: TV/TELETEXT
23 * 1: IIC R/W
23 * 2: HOLD
23 * 3: IR REPEAT INHIBIT
23 * 4: TRANSIENT DISPLAY ON
23 * 5: TIME HOLD
23 * 6: SUB-PAGE MODE
23 * 7: IR TASK PENDING
23 00000057 STAT4 RMB 1 0: KEY FUNCTION PERFORMED
23 * 1: LOCAL REPEATING
23 * 2: P/C PROG : 0, CHAN : 1
23 * 3: MUTE (TRANSIENT)
23 * 4: OSD STATUS TRANSIENT
23 * 5: MUTE (BUTTON)
23 * 6: COINCIDENCE MUTE
23 * 7: SEARCH
23 00000058 PWR RMB 1 $55 AT RESET, $AA NORMALLY
23 00000059 PROG RMB 1 CURRENT PROGRAM NUMBER
23 0000005a CHAN RMB 1 CURRENT CHANNEL NUMBER
23 0000005b DISP RMB 1 CURRENT DISPLAY NUMBER
23 0000005c FTUNE RMB 1 FINE TUNING REGISTER
23 0000005d AVOL RMB 1 VOLUME LEVEL
23 0000005e KEY RMB 1 CODE OF PRESSED KEY (LOCAL)
23 0000005f NUM0 RMB 4 LED DISPLAY RAM
23 00000063 IRRRA1 RMB 1 IR INTERRUPT TEMP.
23 00000064 IRRRA2 RMB 1 " " "
23 00000065 IRRRA3 RMB 1 " " "
23 00000066 IRRRA4 RMB 1 " " "
23 00000067 DIFFH RMB 1 IR TIME DIFFERENCE
23 00000068 DIFFL RMB 1 " " "
23 00000069 IRH RMB 1 IR CODE BIT
23 0000006a IRL RMB 1 COLLECTION
23 0000006b IRCODE RMB 1
23 0000006c IRCNT RMB 1
23 0000006d IRCMCT RMB 1
23 0000006e OLDIR RMB 1
23 *****
23 *
23 *      RAM allocation for Stereoton.
23 *
23 *****
23
23 0000006f POLLTM RMB 1 Poll timer
23 00000070 TONEA RMB 1 Tone (unadjusted for loudness)
23 00000071 LBAL RMB 1 Loudspeaker balance variable
23 00000072 LVL RMB 1 Loudspeaker left volume (reg 1)
23 00000073 LVR RMB 1 Loudspeaker right volume (reg 2)
23 00000074 HVL RMB 1 Headphone volume left (reg 3)
23 00000075 HVR RMB 1 Headphone volume right (reg 4)
23 00000076 TONE RMB 1 Tone variable (Bass/Treble) (reg 5)
23 00000077 MATRIX RMB 1 Current matrix (reg 6)
23 00000078 MATNO RMB 1 Present mode (mono/stereo/lan 1/11.12/12.11)
23 00000079 WS1 RMB 1 Workspace 1 (no interrupt useage)
23 0000007a WS2 RMB 1 Workspace 2 for these please..)
23
23 0000007b VAV RMB 1
23
23 0000007c MONCNT RMB 1 Mono ident count Ident detection
23 0000007d STECNT RMB 1 Stereo ident count variables
23 0000007e DULCNT RMB 1 Dual lang ident count
23 0000007f ERRCNT RMB 1 Error ident count
23 00000080 RCOUNT RMB 1 Ident countdown
23 00000081 RANGE RMB 1 Total ident poll number
23
23 00000082 TEMP RMB 1

```

```

23
23 00000083          STAT5  RMB    1      0: LOUDNESS
23                  *          1: VCR
23                  *          2: OSD NAME TABLE
23                  *          3: OSD DEFAULT P/C NUMBER
23                  *          4: ANALOGUE OSD ON
23                  *          5: NAME-TABLE STANDARD
23                  *          6: STANDARD CHANGED
23                  *          7: RE-INITIALISE TELETEXT
23
23 00000084          STAT6  RMB    1      0: 2-DIGIT PROGRAM ENTRY
23
23 00000085          TMPRG  RMB    1      TEMPORARY PROGRAM NUMBER
23                  *****
23                  *
23                  *          OSD RAM allocation.
23                  *
23                  *****
23
23 00000086          CAS1    RMB    1      ROW 1, colour 1/2 & outline enable
23 00000087          RAD1    RMB    1      Row address & character size
23 00000088          CCR1    RMB    1      Window colour & end column
23 00000089          CAS2    RMB    1      ROW 2, colour 1/2 & outline enable
23 0000008a          RAD2    RMB    1      Row address & character size
23 0000008b          CCR2    RMB    1      Window colour & end column
23 0000008c          CAS3    RMB    1      ROW 3, colour 1/2 & outline enable
23 0000008d          RAD3    RMB    1      Row address & character size
23 0000008e          CCR3    RMB    1      Window colour & end column
23 0000008f          CAS4    RMB    1      ROW 4, colour 1/2 & outline enable
23 00000090          RAD4    RMB    1      Row address & character size
23 00000091          CCR4    RMB    1      Window colour & end column
23 00000092          CAS5    RMB    1      ROW 5, colour 1/2 & outline enable
23 00000093          RAD5    RMB    1      Row address & character size
23 00000094          CCR5    RMB    1      Window colour & end column
23 00000095          CAS6    RMB    1      ROW 6, colour 1/2 & outline enable
23 00000096          RAD6    RMB    1      Row address & character size
23 00000097          CCR6    RMB    1      Window colour & end column
23 00000098          CAS7    RMB    1      ROW 7, colour 1/2 & outline enable
23 00000099          RAD7    RMB    1      Row address & character size
23 0000009a          CCR7    RMB    1      Window colour & end column
23 0000009b          CAS8    RMB    1      ROW 8, colour 1/2 & outline enable
23 0000009c          RAD8    RMB    1      Row address & character size
23 0000009d          CCR8    RMB    1      Window colour & end column
23
23 0000009e          OSDL    RMB    1      CURRENT OSD ROW POINTER
23 0000009f          LIND    RMB    1      ROW TABLE INDEX
23 000000a0          BROW    RMB    1      CHARACTER FLASH ROW
23 000000a1          BCOL    RMB    1      CHAACTER FLASH COLUMNS
23 000000a2          WROW    RMB    1      WINDOW FLASH ROW
23                  *ROW1    RMB    1      FIRST ROW No. (NAME TABLE)
23
23 000000a3          ANAL    RMB    1
23 000000a4          ANAF    RMB    1
23
23 000000a5          TEMP2   RMB    1
23
23 000000a6          RMB     3      UNUSED
23
23 000000a9          STACK   RMB    22     23 BYTES USED FOR STACK
23 000000bf          SP      RMB    1      (1 INTERRUPT AND 9 NESTED SUBS)
23
23
23 00000000          KEYI    EQU     $00
23 00000000          KEYO    EQU     $00
23 00000000          KEYIO   EQU     $00
23 00000003          SERO    EQU     $03
23
23 0000000a          VOLU    EQU     $0A    D/A 2   JP08 IN EVB
23 0000000b          CONT    EQU     $0B    D/A 3   JP09 IN EVB
23 0000000c          BRIL    EQU     $0C    D/A 4   JP10 IN EVB
23 0000000d          SATU    EQU     $0D    D/A 5   JP11 IN EVB
23
23 00000005          L1     EQU     $05    Lang. 1 indicator bit      (LEDOUT)
23 00000006          L2     EQU     $06    Lang. 2 indicator bit      (LEDOUT)
23 00000004          WIDE    EQU     $04    Wide matrix bit           (MATRIX+LEDOUT)
23 00000005          PST     EQU     $05    Pseudo-stereo matrix bit  ( " " )
23 00000006          VCR     EQU     $06    VCR active bit            (STAT3+LEDOUT)
23 00000005          LOUD    EQU     $05    LOUDness effect active bit (STAT3)
23 00000003          MUT     EQU     $03    Mute indicator bit        (MATRIX+LEDOUT)
23
23 00000080          STADR   EQU     $80    Stereoton address (IIC)
23 00000019          NORMVOL EQU     &25    normal volume (mid balance)
23

```



```

23
23
23
23
23
23
23 00000000 PORTA EQU $00 Port A address
23 00000001 PORTB EQU $01 Port B "
23 00000002 PORTC EQU $02 Port C "
23 00000003 PORTD EQU $03 Port D "
23 00000004 DDRA EQU $04 Port A data direction reg.
23 00000005 DDRB EQU $05 Port B " " "
23 00000006 DDRD EQU $06 Port C " " "
23 00000007 DDRD EQU $07 Port D " " "
23
23 00000012 TCR EQU $12 Timer control register.
23 00000013 TSR EQU $13 Timer status register.
23 00000014 ICRH EQU $14 Input capture register, high.
23 00000015 ICRH EQU $15 Input capture register, low.
23 00000016 OCRH EQU $16 Output compare register, high.
23 00000017 OCRH EQU $17 Output compare register, low.
23 00000018 TDRH EQU $18 Timer data register, high.
23 00000019 TDRH EQU $19 Timer data register, low.
23 0000001c MISC EQU $1C Misc. register
23
24
25 00000020 OSD EQU $20 18 OSD data registers
26 00000032 CAS EQU $32 Color & status register
27 00000033 C34 EQU $33 Color 3/4 register
28 00000034 RAD EQU $34 Row address & character size
29 00000035 WCR EQU $35 Window/Column register
30 00000036 CCR EQU $36 Column/color register
31 00000037 HPD EQU $37 Horizontal position delay
32
33 00000039 MAD EQU $39 M-bus address register
34 0000003a MFD EQU $3A M-bus frequency divider
35 0000003b MCR EQU $3B M-bus control register
36 0000003c MSR EQU $3C M-bus status register
37 0000003d MDR EQU $3D M-bus data register
38 0000003e TR1 EQU $3E Test 1, OSD/Timer/PLM
39 0000003f TR2 EQU $3F Test 2, EPROM
40
41 SECTION .RAM2
42
43 00000000 DRAM RMB 128
44
45 SECTION .ROM2
46
47 *****
48 *
49 * OSD update routine - row number & data.
50 *
51 *****
52
53 00000000 >b600 NLINE LDA OSDL 3 INCREMENT
54 00000002 4c INCA INCA 3 6 LINE POINTER
55 00000003 >b700 STAG STA OSDL 4 10 27
56 00000005 >bb00 ADD LIND 3 13 30
57 00000007 97 TAX 2 15 32
58 00000008 >d60000 LDA LDA LTAB0,X 5 20 37
59 0000000b 27f6 BEQ STAG 3 23 OR 40 32 +/- 8
60 0000000d >be00 LDX OSDL 3 LINE POINTER
61 0000000f >de0000 IDX M3,X 5 8 MULTIPLY BY 3
62 00000012 >ea00 ORA RAD1,X 4 12 CHARACTER SIZE INFO.
63 00000014 b734 STA RAD 4 16
64 00000016 >e600 LDA CAS1,X 4 20
65 00000018 b732 STA CAS 4 24
66 0000001a 1f35 BCLR 7,WCR 5 29
67 0000001c 49 ROLA 3 32 GET BIT 6
68 0000001d 49 ROLA 3 35 OF CASx
69 0000001e 2402 BCC SKIPW 3 38
70 00000020 1e35 BSET 7,WCR 5 43 USE IT TO ENABLE WINDOW
71 00000022 >e600 SKIPW LDA CCR1,X 4 47
72 00000024 b736 STA CCR 4 51 83 +/- 8
73 00000026 >be00 LLOK LDX OSDL 3 7 LINE POINTER
74 00000028 >de0000 LDX M16,X 5 12 MULTIPLY BY 16
75
76 0000002b >e6f0 TOSD1 LDA <DRAM-16,X 4 GET DATA AND WRITE
77 0000002d b720 STA OSD 4 8 IT TO OSD REGISTER
78 0000002f >e6f1 LDA <DRAM-15,X
79 00000031 b721 STA OSD+1
80 00000033 >e6f2 LDA <DRAM-14,X FAST OSD DATA TRANSFER
81 00000035 b722 STA OSD+2 TAKES ONLY 128 (8x16)
82 00000037 >e6f3 LDA <DRAM-13,X CYCLES.

```

```

83 00000039 b723 STA OSD+3
84 0000003b >e6f4 LDA <DRAM-12,X
85 0000003d b724 STA OSD+4
86 0000003f >e6f5 LDA <DRAM-11,X
87 00000041 b725 STA OSD+5
88 00000043 >e6f6 LDA <DRAM-10,X
89 00000045 b726 STA OSD+6
90 00000047 >e6f7 LDA <DRAM-9,X
91 00000049 b727 STA OSD+7
92 0000004b >e6f8 LDA <DRAM-8,X
93 0000004d b728 STA OSD+8
94 0000004f >e6f9 LDA <DRAM-7,X
95 00000051 b729 STA OSD+9
96 00000053 >e6fa LDA <DRAM-6,X
97 00000055 b72a STA OSD+10
98 00000057 >e6fb LDA <DRAM-5,X
99 00000059 b72b STA OSD+11
100 0000005b >e6fc LDA <DRAM-4,X
101 0000005d b72c STA OSD+12
102 0000005f >e6fd LDA <DRAM-3,X
103 00000061 b72d STA OSD+13
104 00000063 >e6fe LDA <DRAM-2,X
105 00000065 b72e STA OSD+14
106 00000067 >e6ff LDA <DRAM-1,X
107 00000069 b72f STA OSD+15
108
109 *TOSD2 STX TMP1 4 ALTERNATIVE OSD DATA
110 * LDX #16 2 6 TRANSFER USING A LOOP.
111 * STX TMP2 4 10 THIS HAS THE ADVANTAGE
112 * OSDOOP LDX TMP1 3 OF USING 44 BYTES LESS
113 * LDA <DRAM-1,X 4 7 ROM BUT IT USES TWO MORE
114 * DEC TMP1 5 12 TEMPORARY RAM LOCATIONS
115 * LDX TMP2 3 15 AND TAKES 330 CYCLES
116 * STA <OSD-1,X 5 20 (165us) LONGER.
117 * DEC TMP2 5 25
118 * BNE OSDOOP 3 28 28x16+10=458=128+330
119
120 *****
121 *
122 * Character and window flash. *
123 * *
124 *****
125
126 0000006b >09001f BRCLR 4,CNT,WBLK 5 25
127 0000006e >b600 CHBLK LDA BROW 3 28 CHARACTER BLINK
128 00000070 2719 BEQ NCHK 3 31
129 00000072 b634 LDA RAD 3 34
130 00000074 a40f AND #$0F 2 36
131 00000076 >b100 CMP BROW 3 39
132 00000078 2611 BNE NCHK 3 42
133 0000007a >b600 LDA BCOL 3 45
134 0000007c a40f AND #$0F 3 48
135 0000007e 97 TAX 2 50
136 0000007f ad1b BSR SPFL 34 84 1st CHARACTER (LS NIBBLE)
137 00000081 >be00 LDX BCOL 3 87
138 00000083 54 LSRX 3 90
139 00000084 54 LSRX 3 93
140 00000085 54 LSRX 3 96
141 00000086 54 LSRX 3 99
142 00000087 2702 BEQ NCHK 3 102 IF MS NIBBLE ZERO THEN NO
143 00000089 ad11 BSR SPFL 34 136 2nd CHARACTER
144 0000008b 200e NCHK BRA NOBLK 3 139
145 0000008d >b600 WBLK LDA WROW 3 139 WINDOW BLINK
146 0000008f 270a BEQ NOBLK
147 00000091 b634 LDA RAD
148 00000093 a40f AND #$0F
149 00000095 >b100 CMP WROW
150 00000097 2602 BNE NOBLK 371 +/- 8
151 00000099 1f35 BCLR 7,$35 with INT 381 +/- 8 cycles
152 0000009b 80 NOBLK RTI 9 148 ie 190.5 +/- 4 us
153
154 0000009c e620 SPFL LDA OSD,X 6 + 4 10
155 0000009e a43f AND #$3F 2 12
156 000000a0 2609 BNE NTSP 3 15
157 000000a2 e620 LDA OSD,X 4 19
158 000000a4 a4c0 AND #$C0 2 21
159 000000a6 ab0e ADD #$0E 2 23
160 000000a8 e720 STA OSD,X 5 28
161 000000aa 81 RTS 6 34
162 000000ab f620 NTSP CLR OSD,X
163 000000ad 81 RTS

```

```

165
166
167
168
169
170
171 000000ae >060004 OSDEF BRSET 3,STAT5,DOFF
172 000000b1 >1600 BSET 3,STAT5
173 000000b3 2002 BRA OSDLE
174
175 000000b5 >1700 DOFF BCLR 3,STAT5
176
177 000000b7 9b OSDLE SEI
178 000000b8 >1900 BCLR 4,STAT5 NOT ANALOGS
179 000000ba >1500 BCLR 2,STAT5 NOT NAME TABLE
180
181 000000bc ae1d LDZ #29 CLEAR PAGE 0
182 000000be >6fff CLR CAS1-1,X OSD CONTROL
183 000000c0 5a DECX BYTES
184 000000c1 26fb BNE DOOP
185
186 000000c3 >cd0000 JSR CDISP2
187 000000c6 3f30 CLR $30
188 000000c8 3f31 CLR $31
189 000000ca >06000a BRSET 3,STAT5,SKPDEF
190 000000cd >c6000c LDA DRAM+12 PROGRAM NUMBER
191 000000d0 b730 STA $30
192 000000d2 >c6000e LDA DRAM+14
193 000000d5 b731 STA $31
194 000000d7 5f SKPDEF CLRX
195 000000d8 a67f LDA #127
196 000000da ad1b BSR OSDCLR
197
198 000000dc a620 LDA ##00100000 HORIZONTAL POSITION : ZERO
199 000000de b737 STA HPD
200 000000e0 a6a3 LDA ##10100011 COLOR 1,0 = RED, CYAN, EDGE ON
201 000000e2 >050002 BRCLR 2,STAT4,PMD PROGRAM MODE ?
202 000000e5 a6a6 LDA ##10100110 NO, COLOR 0 = YELLOW
203 000000e7 >b700 STA CAS1
204 000000e9 a610 LDA ##00010000 SINGLE WIDTH/HIGHT
205 000000eb >b700 STA RAD1
206
207 000000ed a60c LDA #$0C DEFAULT TO VOLUME
208 000000ef >b700 STA ANAL
209 000000f1 >a600 LDA #AVOL
210 000000f3 >b700 STA ANAF
211 000000f5 9a CLI
212 000000f6 81 RTS
213
214 000000f7 4c OSDCLR INCA
215 000000f8 >b700 STA W1
216 000000fa 5c DCLR INCX
217 000000fb >6fff CLR DRAM-1,X
218 000000fd >b300 CPX W1
219 000000ff 26f9 BNE DCLR
220 0000101 81 RTS

```

```

222
223
224
225
226
227
*****
*
*           Program/Channel/Name display.
*
*****
228 00000102 3334002328003334 BNTAB FCB $33,$34,0,$23,$28,0,$33,$34 ST CH ST
229 0000010a 240e00002e212d25 FCB $24,$0E,0,0,$2E,$21,$2D,$25,$C0 D. NAME
230 00000113 005c009e003c00fe MTAB FCB 0,$5C,0,$9E,0,$3C,0,$FE > < Y ^
231 0000011b 008b008d006d00a9 FCB 0,$8B,0,$8D,0,$6D,0,$A9,$C0 + - M I
232
233 00000124 >040090 PRDSP BRSET 2,STAT5,OSDLE
234 00000127 >1400 BSET 2,STAT5
235 00000129 a61a LDA ##00011010 COLOR 2 = GREEN
236 0000012b b733 STA C34 COLOR 3 = CYAN
237 0000012d a6e1 LDA ##11100001 OSD & PLL ON,
238 0000012f b735 STA WCR WINDOW ON (COLUMN 1)
239 00000131 a621 LDA ##00100001 HORIZONTAL POSITION : ONE
240 00000133 b737 STA HPD
241 00000135 3f30 CLR $30 PUT A SPACE AT 17th AND 18th
242 00000137 3f31 CLR $31 CHARACTERS
243
244 00000139 a6e5 LDA ##11100101 COLOR 1,0 = RED, MAGENTA, EDGE ON
245 0000013b >b700 STA CAS1
246 0000013d a6e6 LDA ##11100110 AND WINDOW ON (USING BIT 6)
247 0000013f >b700 STA CAS8
248
249 00000141 a610 LDA ##00010000 SINGLE WIDTH/HIGHT, INTERRUPTS ON
250 00000143 ae18 LDX #24
251 00000145 >e7fd STLP STA RAD1-3,X
252 00000147 5a DECX
253 00000148 5a DECX
254 00000149 5a DECX
255 0000014a 26f9 BNE STLP
256
257 0000014c a6e6 LDA ##11100110 COLOR 1,0 = RED, YELLOW, EDGE ON
258 0000014e ae12 LDX #18
259 00000150 >e7fd STLP2 STA CAS2-3,X
260 00000152 5a DECX
261 00000153 5a DECX
262 00000154 5a DECX
263 00000155 26f9 BNE STLP2
270
271 00000157 >3f00 CLR OSDL
272 00000159 a609 LDA #LTAB3-LTAB0
273 0000015b >b700 STA LIND THIRD TABLE
274
275 0000015d a603 LDA #3 START AT ROW 3
276 0000015f >b700 STA BROW
277 00000161 a603 LDA #S03 START AT COLUMN 3
278 00000163 >b700 STA BCOL
279 00000165 a600 LDA #0
280 00000167 >b700 STA WROW
281 00000169 a601 LDA #1
282 0000016b >b700 STA COUNT
283
284 0000016d ae80 LDA #128 CLEAR 1st THRU 8th ROWS
285 0000016f >6fff ACLR CLR DRAM-1,X
286 00000171 5a DECX
287 00000172 26fb BNE ACLR
288
289 00000174 5f CLRX
290 00000175 >d60000 BNPL LDA BNTAB,X
291 00000178 alc0 CMP #C0
292 0000017a 2706 BEQ FINBN
293 0000017c >d70000 STA DRAM,X
294 0000017f 5c INCX
295 00000180 20f3 BRA BNPL
296
297 00000182 ae10 FINBN LDX #16
298 00000184 >1d00 BCLR 6,STAT5 CLEAR STANDARD CHANGE FLAG
299 00000186 >b600 LDA COUNT
300 00000188 >b700 STA W2

```

```

302
303
304
305
306
307
308 0000018a >b600          BNLPL   LDA      W2          STATION No.
309 0000018c >bf00          STX      W3
310 0000018e >cd0000        JSR      CBCD
311 00000191 >b700          STA      W1
312 00000193 a40f          AND      #$0F
313 00000195 ab10          ADD      #$10
314 00000197 >be00          LDX      W3
315 00000199 >d70001        STA      DRAM+1,X
316 0000019c >b600          LDA      W1
317 0000019e 44          LSRA
318 0000019f 44          LSRA
319 000001a0 44          LSRA
320 000001a1 44          LSRA
321 000001a2 2602        BNE      NOTZR
322 000001a4 a6f0          LDA      #$F0          LEADING ZERO BLANK
323 000001a6 ab10          ADD      #$10
324 000001a8 >d70000        STLSN   STA      DRAM,X
325 000001ab >b600          LDA      W2          STATION No.
326 000001ad abdf          ADD      #$DF
327 000001af >b700          STA      SUBADR
328 000001b1 a6a0          LDA      #$A0
329 000001b3 >b700          STA      ADDR
330 000001b5 >cd0000        JSR      READ
331 000001b8 >b601          LDA      IOBUF+1      CHANNEL No.
332 000001ba a47f          AND      #$7F
333 000001bc >cd0000        JSR      CBCD
334 000001bf >b700          STA      W1
335 000001c1 a40f          AND      #$0F
336 000001c3 ab10          ADD      #$10
337 000001c5 >be00          LDX      W3
338 000001c7 >d70004        STA      DRAM+4,X      MSD
339 000001ca >b600          LDA      W1
340 000001cc 44          LSRA
341 000001cd 44          LSRA
342 000001ce 44          LSRA
343 000001cf 44          LSRA
344 000001d0 ab10          ADD      #$10
345 000001d2 >d70003        STA      DRAM+3,X      LSD
347
348
349
350
351
352
353 000001d5 >1a00          BSET     5,STAT5
354 000001d7 >0e0102        BRSET   7,IOBUF+1,PALS
355 000001da >1b00          BCLR    5,STAT5
356
357 000001dc >cd0000        PALS    JSR      CHGST
358
359 000001df >b600          LDA      W2
360 000001e1 >cd0000        JSR      GNAME2
361 000001e4 9f          TXA
362 000001e5 ab10          ADD      #16
363 000001e7 97          TAX
364 000001e8 >3c00          INC      W2
365 000001ea a360          CPX      #96
366 000001ec 2203        BHI     NOJMP
367 000001ee >cc0000        JMP     BNLPL
368
369 000001f1 5f          NOJMP   CLRX
370 000001f2 >d60000        MTL     LDA      MTAB,X
371 000001f5 a1c0          CMP     #$C0
372 000001f7 2706          BEQ     ANFIN
373 000001f9 >d70070        STA     DRAM+112,X
374 000001fc 5c          INCX
375 000001fd 20f3        BRA     MTL
376
377 000001ff >cd0000        ANFIN   JSR      WIND
378 00000202 >1800          SEC30  BSET     4,STAT
379 00000204 a61e          LDA     #30
380 00000206 >b700          STA     TMR
381 00000208 81          RTS

```

```

383
384
385
386
387
388
*****
*
*       Look for channel name.
*
*****
389 00000209 >04002f FNAME BRSET 2, STAT4, NONAME CHANNEL MODE
390 0000020c a6a0 LDA # $A0
391 0000020e >b700 STA ADDR
392 00000210 a6e0 LDA # $E0
393 00000212 >b700 STA SUBADR
394
395 00000214 >b600 LDA CHAN
396 00000216 >cd0000 JSR CHEX
397 00000219 >b700 STA COUNT
398 0000021b 030205 BRCLR 1, PORTC, OLOOP 38.9 MHz ?
399 0000021e 0b0202 BRCLR 5, PORTC, OLOOP NO, SECAM ?
400 00000221 >1e00 BSET 7, COUNT NO, PAL
401
402 00000223 >cd0000 OLOOP JSR READ
403 00000226 >b601 LDA IOBUF+1
404 00000228 020202 BRSET 1, PORTC, IF38 38.9 MHz ?
405 0000022b a47f AND # $7F YES, SO IGNORE STANDARD
406 0000022d 2704 IF38 BEQ CH0 CHAN
407 0000022f >b100 CMF COUNT
408 00000231 274d BEQ NOFND
409 00000233 >3c00 CH0 INC SUBADR
410 00000235 >b600 LDA SUBADR
411 00000237 a1f7 CMF # $F7
412 00000239 23e8 BLS OLOOP
413 0000023b >be00 NONAME LDX W3
414 0000023d a623 LDA # $23 NO NAME SO DISPLAY Ch. No.
415 0000023f >d70008 STA DRAM+8, X
416 00000242 a628 LDA # $28
417 00000244 >d70009 STA DRAM+9, X
418 00000247 >b600 LDA CHAN
419 00000249 44 LSRA
420 0000024a 44 LSRA
421 0000024b 44 LSRA
422 0000024c 44 LSRA
423 0000024d ab10 ADD # $10
424 0000024f >d7000a STA DRAM+10, X 3rd CHAR (NAME)
425 00000252 >b600 LDA CHAN
426 00000254 a40f AND # $0F
427 00000256 ab10 ADD # $10
428 00000258 >d7000b STA DRAM+11, X 4th CHAR (NAME)
429
430 0000025b a630 LDA # $30 P
431 0000025d >d7000d STA DRAM+13, X
432 00000260 a621 LDA # $21 A
433 00000262 >d7000e STA DRAM+14, X
434 00000265 a62c LDA # $2C L
435 00000267 >d7000f STA DRAM+15, X
436 0000026a 030212 BRCLR 1, PORTC, SPAL 38.9 MHz ?
437 0000026d 0a020f BRSET 5, PORTC, SPAL NO, PAL ?
438 00000270 a633 LDA # $33 S
439 00000272 >d7000d STA DRAM+13, X
440 00000275 a625 LDA # $25 E
441 00000277 >d7000e STA DRAM+14, X
442 0000027a a623 LDA # $23 C
443 0000027c >d7000f STA DRAM+15, X
444 0000027f 81 SPAL RTS
445
446 00000280 >b600 NOFND LDA SUBADR
447 00000282 a0df SUB # $DF
448 00000284 48 GNAME2 LSLA x2
449 00000285 48 LSLA x4
450 00000286 ab7c ADD # $7C
451 00000288 >b700 STA SUBADR
452 0000028a >cd0000 JSR READ
453 0000028d >be00 LDX W3
454 0000028f >b601 LDA IOBUF+1
455 00000291 >d7000c STA DRAM+12, X 1st CHAR (NAME)
456 00000294 >b600 LDA IOBUF
457 00000296 >d7000d STA DRAM+13, X 2nd CHAR (NAME)
458 00000299 >3c00 INC SUBADR
459 0000029b >3c00 INC SUBADR
460 0000029d >cd0000 JSR READ
461 000002a0 >be00 LDX W3
462 000002a2 >b601 LDA IOBUF+1
463 000002a4 >d7000e STA DRAM+14, X 3rd CHAR (NAME)
464 000002a7 >b600 LDA IOBUF
465 000002a9 >d7000f STA DRAM+15, X 4th CHAR (NAME)
466 000002ac 81 RTS
467

```

```

468
469
470
471
472
473
474 000002ad 03040c0d0e0f CURTAB FCB 3,4,12,13,14,15
475
476 000002b3 ad19 CLFT BSR FCUR
477 000002b5 a305 CPX #5
478 000002b7 2502 BLO NRAP1
479 000002b9 ae00 LDX #FFF
480 000002bb 5c NRAP1 INCX
481 000002bc >d60000 NEWC LDA CURTAB,X
482 000002bf >b700 STA BCOL
483 000002c1 >cc0000 SEC32 JMP SEC30
484
485 000002c4 ad08 CRGT BSR FCUR
486 000002c6 5d TSTX
487 000002c7 2602 BNE NRAP2
488 000002c9 ae06 LDX #6
489 000002cb 5a NRAP2 DECX
490 000002cc 20ee BRA NEWC
491
492 000002ce >b600 FCUR LDA BCOL
493 000002d0 >b700 STA W1
494 000002d2 ae00 LDX #FFF
495 000002d4 5c CRNF INCX
496 000002d5 >d60000 LDA CURTAB,X
497 000002d8 >b100 CMP W1
498 000002da 26f8 BNE CRNF
499 000002dc 81 RTS
500
501 000002dd a631 WIND LDA #800110001 WINDOW BLUE, OFF AT 17
502
503 000002df ae12 LDX #18
504 000002e1 >e7fd STLP3 STA CCR2-3,X
505 000002e3 5a DECX
506 000002e4 5a DECX
507 000002e5 5a DECX
508 000002e6 26f9 BNE STLP3
509
510 000002e8 a611 LDA #800010001 WINDOW BLACK, OFF AT 17
511 000002ea >b700 STA CCR1
512 000002ec >b700 STA CCR8
513 000002ee >be00 LDX BROW
514 000002f0 5a DECX
515 000002f1 5a DECX
516 000002f2 >de0000 LDX M3,X
517 000002f5 >e600 LDA CCR1,X
518 000002f7 >b700 STA W2
519 000002f9 >1c00 BSET 6,W2
520 000002fb >b600 LDA W2
521 000002fd >e700 STA CCR1,X
522 000002ff 81 RTS
523
524
525
526
527
528
529
530 00000300 >b600 CUP LDA BROW
531 00000302 a103 CMP #3
532 00000304 2304 BLS TOOSM
533 00000306 >3a00 DEC BROW
534 00000308 20d3 BRA WIND
535 0000030a >b600 TOOSM LDA COUNT
536 0000030c a101 CMP #1
537 0000030e 27b1 SEC31 BEQ SEC32
538 00000310 >3a00 DEC COUNT
539 00000312 2012 BRA FIN30
540
541 00000314 >b600 CDWN LDA BROW
542 00000316 a108 CMP #8
543 00000318 2404 BHS TOOBG
544 0000031a >3c00 INC BROW
545 0000031c 20bf BRA WIND
546 0000031e >b600 TOOBG LDA COUNT
547 00000320 a113 CMP #19
548 00000322 27ea BEQ SEC31
549 00000324 >3c00 INC COUNT
550 00000326 >cd0000 FIN30 JSR SEC30
551 00000329 >cc0000 JMP FINBN

```

```

553
554
555
556
557
558
559 0000032c >1b00          CHST  BCLR  5,STAT5          DEFAULT TO SECAM
560 0000032e >be00          LDY  BROW
561 00000330          DECX
562 00000331          DECX
563 00000332          DECX
564 00000333 >de0000          LDY  M16,X
565 00000336 >d60006          LDA  DRAM+6,X
566 00000339 a43f          AND  #$3F
567 0000033b a130          CMP  #$30          PAL ?
568 0000033d 2702          BEQ  SZER
569 0000033f >1a00          BSET 5,STAT5          NO, MAKE IT PAL
570 00000341 >1c00          SZER BSET 6,STAT5          STANDARD CHANGED
571 00000343 >cd0000          JSR  SEC30
572
573 00000346 a630          CHGST LDA  #$30
574 00000348 >d70006          STA  DRAM+6,X
575 0000034b a621          LDA  #$21
576 0000034d >d70007          STA  DRAM+7,X
577 00000350 a62c          LDA  #$2C
578 00000352 >d70008          STA  DRAM+8,X
579 00000355 a600          LDA  #0
580 00000357 >d70009          STA  DRAM+9,X
581 0000035a >d7000a          STA  DRAM+10,X
582 0000035d 03021c          BRCLR 1,PORTC,PAL          38.9MHz ?
583 00000360 >0a0019          BRSET 5,STAT5,PAL          NO, PAL ?
584 00000363 a633          SECAM LDA  #$33          NO, SECAM
585 00000365 >d70006          STA  DRAM+6,X
586 00000368 a625          LDA  #$25
587 0000036a >d70007          STA  DRAM+7,X
588 0000036d a623          LDA  #$23
589 0000036f >d70008          STA  DRAM+8,X
590 00000372 a621          LDA  #$21
591 00000374 >d70009          STA  DRAM+9,X
592 00000377 a62d          LDA  #$2D
593 00000379 >d7000a          STA  DRAM+10,X
594 0000037c >0d0012          PAL  BRCLR 6,STAT5,NSTCH
595
596 0000037f 9f          TXA
597 00000380 ab05          ADD  #5
598 00000382 >b700          STA  COUNT
599 00000384 >d60006          XLP  LDA  DRAM+6,X
600 00000387 ab40          ADD  #$40
601 00000389 >d70006          STA  DRAM+6,X
602 0000038c 5c          INCX
603 0000038d >b300          CPX  COUNT
604 0000038f 26f3          BNE  XLP
605
606 00000391 81          NSTCH RTS
607
608
609
610
611
612
613
614 00000392 ad40          PLUS  BSR  GETIT
615 00000394 4c          INCA
616 00000395 a43f          AND  #$3F
617
618 00000397 a119          CMP  #$19          9
619 00000399 2208          BHI  MT9
620 0000039b a110          LTE9 CMP  #$10          0
621 0000039d 2410          BHS  NLT0
622 0000039f a610          LDA  #$10          0
623 000003a1 200c          BRA  NLT0
624 000003a3 a121          MT9  CMP  #$21          A
625 000003a5 2202          BHI  MTA
626 000003a7 a621          LDA  #$21          A
627 000003a9 a13a          MTA  CMP  #$3A          Z
628 000003ab 2302          BLS  NLT0
629 000003ad a600          SPACE LDA  #$00          SPACE
630
631 000003af aa40          NLT0 ORA  #$40
632 000003b1 >d70000          STA  DRAM,X
633 000003b4 >cc0000          JMP  SEC30
634
635 000003b7 ad1b          MINUS BSR  GETIT
636 000003b9 4a          DECA
637 000003ba a43f          AND  #$3F
638

```



```

639 000003bc a121          CMP    #$21      A
640 000003be 2508          BLO    LTA
641 000003c0 a13a          GTEA   CMP    #$3A      Z
642 000003c2 23eb          BLS    NLTO
643 000003c4 a63a          LDA    #$3A      Z
644 000003c6 20e7          BRA    NLTO
645 000003c8 a119          LTA    CMP    #$19      9
646 000003ca 2302          BLS    LTN
647 000003cc a619          LDA    #$19      9
648 000003ce a110          LT9    CMP    #$10      0
649 000003d0 24dd          BHS    NLTO
650 000003d2 20d9          BRA    SPACE
651
652 000003d4 >b600          GETIT  LDA    BROW
653 000003d6 a002          SUB    #2
654 000003d8 48            LSLA                   x2
655 000003d9 48            LSLA                   x4
656 000003da 48            LSLA                   x8
657 000003db 48            LSLA                   x16
658 000003dc >bb00          ADD    BCOL
659 000003de 97            TAX
660 000003df >d60000       LDA    DRAM,X
661 000003e2 81            RTS
662
663 *****
664 *
665 *      Name store.
666 *
667 *****
668
669 000003e3 a6a0          SAVE   LDA    #$A0
670 000003e5 >b700          STA    ADDR
671 000003e7 >b600          LDA    COUNT
672 000003e9 >bb00          ADD    BROW
673 000003eb 48            LSLA
674 000003ec 48            LSLA
675 000003ed ab70          ADD    #$70
676 000003ef >b700          STA    SUBADR
677 000003f1 a603          LDA    #3
678 000003f3 >b700          STA    W1
679 000003f5 >b700          STA    W2
680 000003f7 >be00          LDX    BROW
681 000003f9 5a          DECX
682 000003fa 5a          DECX
683 000003fb 58          LSLX
684 000003fc 58          LSLX
685 000003fd 58          LSLX
686 000003fe 58          LSLX
687 000003ff >bf00          STX    W3
688 00000401 >d6000c       LDA    DRAM+12,X
689 00000404 a43f          AND    #$3F
690 00000406 >b700          STA    IOBUF
691 00000408 >d6000d       LDA    DRAM+13,X
692 0000040b a43f          AND    #$3F
693 0000040d >b701          STA    IOBUF+1
694 0000040f >ae00          LDX    #SUBADR
695 00000411 >cd0000       JSR    WRITE
696
697 00000414 >3c00          INC    SUBADR
698 00000416 >3c00          INC    SUBADR
699 00000418 >be00          LDX    W3
700 0000041a a603          LDA    #3
701 0000041c >b700          STA    W1
702 0000041e >b700          STA    W2
703 00000420 >d6000e       LDA    DRAM+14,X
704 00000423 a43f          AND    #$3F
705 00000425 >b700          STA    IOBUF
706 00000427 >d6000f       LDA    DRAM+15,X
707 0000042a a43f          AND    #$3F
708 0000042c >b701          STA    IOBUF+1
709 0000042e >ae00          LDX    #SUBADR
710 00000430 >cd0000       JSR    WRITE

```

```

712
713
714
715
716
717
718 00000433 >be00          LDX      W3
719 00000435 >d60003      LDA      DRAM+3,X
720 00000438 48          LSLA
721 00000439 48          LSLA
722 0000043a 48          LSLA
723 0000043b 48          LSLA
724 0000043c >b700      STA      W1
725 0000043e >d60004      LDA      DRAM+4,X
726 00000441 a40f        AND      #$0F
727 00000443 >bb00        ADD      W1
728 00000445 >cd0000      JSR      CHEX
729 00000448 >b700        STA      IOBUF
730
731 0000044a >be00          LDX      W3
732 0000044c >d60006      LDA      DRAM+6,X
733 0000044f a43f        AND      #$3F
734 00000451 a133        CMP      #$33
735 00000453 2702        BEQ      STSEC
736 00000455 >1e00        BSET     7,IOBUF
737
738 00000457 >b600          STSEC   LDA      COUNT
739 00000459 >bb00          ADD      BROW
740 0000045b abdc        ADD      #$DC
741 0000045d >b700          STA      SUBADR
742 0000045f a602        LDA      #2
743 00000461 >b700          STA      W1
744 00000463 >b700          STA      W2 -
745 00000465 >ae00        LDX      #SUBADR
746 00000467 >cd0000      JSR      WRITE
747
748 0000046a >cd0000      JSR      SEC30
749 0000046d >cc0000      JMP      FINBN
750
751
752
753
754
755
756
757 00000470 0a00          LTAB0   FCB      10,0          IDLE DISPLAY
758 00000472 090800      LTAB1   FCB      9,8,0        PR/CH DISPLAY
759 00000475 07080a00    LTAB2   FCB      7,8,10,0     ANALOGUE DISPLAY
760 00000479 0203040506070809 LTAB3   FCB      2,3,4,5,6,7,8,9,0 PR/CH/STD/NAME TABLE
761
762 00000482 1020304050607080 M16     FCB      $10,$20,$30,$40,$50,$60,$70,$80  MULT x 16
763 0000048a 000306090c0f1215 M3       FCB      0,3,6,9,12,15,18,21  MULT x 3

```

```

765
766
767
768
769
770
771 00000492 a60c
772 00000494 >b700
773 00000496 >a600
774 00000498 >b700
775 0000049a >1900
776 0000049c a60a
777 0000049e b733
778 000004a0 a670
779 000004a2 b735
780 000004a4 a622
781 000004a6 b737
782 000004a8 3f30
783 000004aa 3f31
784 000004ac 5f
785 000004ad a609
786 000004af >cd0000
787 000004b2 ae10
788 000004b4 a61f
789 000004b6 >cd0000
790 000004b9 a610
791 000004bb >b700
792 000004bd >b600
793 000004bf 2703
794 000004c1 >cd0000
795 000004c4 a601
796 000004c6 >b700
797 000004c8 a602
798 000004ca >b700
799 000004cc a6a3
800 000004ce >050002
801 000004d1 a6a6
802 000004d3 >b700
803 000004d5 >b700
804 000004d7 a6d0
805 000004d9 >b700
806 000004db a610
807 000004dd >b700
808 000004df a612
809 000004e1 >b700
810 000004e3 >b700
811
812 000004e5 >1800
813 000004e7 a61e
814 000004e9 >040005
815 000004ec >000002
816 000004ef a606
817 000004f1 >b700
818 000004f3 81

*****
*
* Bottom corner Program/Channel no. display.
*
*****

PCOSD LDA #S0C
STA ANAL
LDA #AVOL
STA ANAF
BCLR 4,STAT5 NOT ANALOGS
LDA #%00001010 COLOR 2 = GREEN
STA C34 COLOR 3 = BLUE
LDA #%01110000 OSD & PLL ON,
STA WCR WINDOW OFF (COLUMN 16)
LDA #%00100010 HORIZONTAL POSITION : TWO
STA HPD
CLR $30 PUT A SPACE AT 17th AND 18th
CLR $31 CHARACTERS
CLR#X
LDA #9
JSR OSDCLR CLEAR UNUSED CHARACTERS
LDX #16
LDA #31
JSR OSDCLR CLEAR UNUSED CHARACTERS
PNAME LDA #16
STA W3
LDA PROG
BEQ SKPGN
JSR FNAME
SKPGN LDA #1
STA OSDL START AT 1 TO PREVENT
LDA #LTAB1-LTAB0 DOUBLE-HIGHT-LINE-SHIFT FLASH
STA LIND FIRST TABLE
LDA #%10100011 COLOR 1,0 = RED, CYAN, EDGE ON
BRCLR 2,STAT4,PMD2 PROGRAM MODE ?
LDA #%10100110 NO, COLOR 0 = YELLOW
PMD2 STA CAS1
STA CAS2
LDA #%11010000 DOUBLE WIDTH/HIGHT
STA RAD1
LDA #%00010000 SINGLE WIDTH/HIGHT
STA RAD2
LDA #%00010010 WINDOW CYAN
STA CCR1
STA CCR2
SEC5 BSET 4,STAT
LDA #30
BRSET 2,STAT4,S30 CHANNEL MODE ?
BRSET 0,STAT6,S30 NO, 2-DIGIT PROG No. ENTRY ?
LDA #6 NO, SO 6 SECONDS ONLY
S30 STA TMR
RTS

```

```

820
821
822
823
824
825
826 000004f4 0e121113 CHAR FCB $0E,$12,$11,$13 BARGRAPH CHARACTERS
827
828 000004f8 636f6e74a2b2a9ac ANCH FCB $63,$6F,$6E,$74,$A2,$B2,$A9,$AC ANALOG
829 00000500 63b321f4f6efecf5 FCB $63,$B3,$21,$F4,$F6,$EF,$EC,$F5 LOGOS
830
831 00000508 >b700 ANOSD STA W3
832 0000050a >080041 BRSET 4,STAT5,LOGO ANALOGS
833 0000050d >1800 BSET 4,STAT5 SET-UP SKIP FLAG
834 0000050f 5f CLRX
835 00000510 a67f LDA #127
836 00000512 >cd0000 JSR OSDCLR CLEAR ALL CHARACTERS
837 00000515 ae1d LDX #29
838 00000517 >6fff COOP CLR CAS1-1,X
839 00000519 5a DECX
840 0000051a 26fb BNE COOP
841
842 0000051c a60a LDA #80001010 COLOR 2 = GREEN
843 0000051e b733 STA C34 COLOR 3 = BLUE
844 00000520 a6e1 LDA #811100001 OSD & PLL ON,
845 00000522 b735 STA WCR WINDOW ON (COLUMN 1)
846 00000524 a622 LDA #800100010 HORIZONTAL POSITION : TWO
847 00000526 b737 STA HPD
848 00000528 3f30 CLR S30
849 0000052a 3f31 CLR S31
850 0000052c a6a6 LDA #811100110 COLOR 1,0 = RED, YELLOW, EDGE ON
851 0000052e >b700 STA CAS1 AND WINDOW ON (USING BIT 6)
852 00000530 >b700 STA CAS2
853 00000532 a6a6 LDA #810100110 COLOR 1,0 = RED, YELLOW, WINDOW OFF
854 00000534 >b700 STA CAS3
855 00000536 a610 LDA #800010000 SINGLE WIDTH/HIGHT, INTERRUPTS ON
856 00000538 >b700 STA RAD1
857 0000053a >b700 STA RAD2
858 0000053c >b700 STA RAD3
859 0000053e a6e3 LDA #811100011 WINDOW WHITE, OFF AT 3
860 00000540 >b700 STA CCR1
861 00000542 >b700 STA CCR2
862 00000544 a611 LDA #800010001 WINDOW BLACK, OFF AT 17
863 00000546 >b700 STA CCR3
864
865 00000548 >3f00 CLR OSDL
866 0000054a a605 LDA #LTAB2-LTAB0
867 0000054c >b700 STA LIND SECOND TABLE
*****
869
870
871
872
873
874
875 0000054e >be00 LOGO LDX ANAL
876 00000550 >d60000 LDA ANCH,X
877 00000553 >c70000 STA DRAM
878 00000556 5c INCX
879 00000557 >d60000 LDA ANCH,X
880 0000055a >c70001 STA DRAM+1
881 0000055d 5c INCX
882 0000055e >d60000 LDA ANCH,X
883 00000561 >c70010 STA DRAM+16
884 00000564 5c INCX
885 00000565 >d60000 LDA ANCH,X
886 00000568 >c70011 STA DRAM+17
887

```

```

888
889
890
891
892
893
894 0000056b >b600          LDA      W3
895 0000056d >3f00          CLR      W2
896 0000056f 44             LSR     LSR
897 00000570 >3900          ROL     W2
898 00000572 44             LSR     LSR
899 00000573 >3900          ROL     W2
900 00000575 >b700          STA     W3
901 00000577 ae10          LDX     #16
902 00000579 5a             LSR     LSR
903 0000057a >b300          CPX     W3
904 0000057c 270a          BEQ     STAR
905 0000057e 2204          BHI     DOT
906 00000580 a614          LDA     #S14
907 00000582 200d          BRA     SKST
908 00000584 a60e          DOT     LDA     #S0E
909 00000586 2009          BRA     SKST
910 00000588 >bf00          STAR   STX     W1
911 0000058a >be00          LDX     W2
912 0000058c >d60000          LDA     CHAR,X
913 0000058f >be00          LDX     W1
914 00000591 >d70020          SKST   STA     DRAM+32,X
915 00000594 5d             TSTX
916 00000595 26e2          BNE     LSR
917
918 00000597 >cc0000          JMP     SEC5
919
920
          END

```

Serial bootstrap for the RAM and EEPROM1 of the MC68HC05B6

By Jeff Wright,
Motorola Ltd., East Kilbride

INTRODUCTION

The MC68HC05B6 has 256 bytes of on chip EEPROM, called EEPROM1, which can be used to store variable data in a non-volatile manner. In many applications this EEPROM1 will be used to hold a look-up table or system set up variables. In these cases it is usually a requirement that the EEPROM1 be initialised during

the manufacture of the application. In addition, loading small programs into RAM and executing them is an easy way of trying out new software routines. This application note describes one method for serially loading (bootstrapping) the EEPROM1 via a program executing in the RAM of the MC68HC05B6.

BUILT IN BOOTSTRAP

The MC68HC05B6 has a built in RAM serial bootstrap program contained in the mask ROM of the device that uses the SCI. It would therefore seem a simple task to load programs into RAM; however, as ROM space on the device is obviously critical, a very simple protocol has been implemented. This means that the boot-loader on the 'B6' does not accept S-records which are the normal output from an assembler; instead, the protocol expects pure binary data preceded by a count byte that holds the size of the program to be downloaded. No address information is contained in the download; instead, the boot-loader always starts the program load at address \$50 in RAM. The first byte (the count byte) is stored here and then as the subsequent bytes are received via the SCI they are stored at incrementing RAM locations and the count byte is

decremented for each byte received. When the count byte reaches zero the bootstrap program jumps to address \$51 and starts to execute the program that has just been loaded. No built in bootstrap routine is provided for the EEPROM1 array.

These restrictions present two problems:

- i) How to convert assembler output to the format accepted by the 68HC05B6 RAM bootstrap routine?
- ii) How to bootstrap the EEPROM1 of the 68HC05B6?

This application note provides a solution for each of these problems.

1) CONVERTING S-RECORDS FOR RAM BOOTSTRAP

To use the built in RAM bootstrap program on the MC68HC05B6 the device must be configured as shown in Figure 1. If these conditions are met when the reset pin is released, then the serial bootstrap program described above will start to execute and a program can be downloaded via a 9600 baud RS-232 source. Personal computers usually have one or more RS-232 ports referred to as COM ports. To overcome the format difference between S-records and that accepted by the boot-loader, a conversion program is required. There is also an additional problem when using a PC – when a file is copied to a COM port to transfer it, it is the ascii characters that are transmitted, not the binary data. This means for example that if a file containing the typed data byte \$A5 was copied via the COM port to the B6, the B6 would in fact receive two bytes: \$41 and \$35, which represent the ascii characters A and 5 respectively.

This means that the conversion program has to strip out the S-record format and convert the resultant data to binary format for transfer to the HC05B6. It must also insert the count byte at the beginning of the output file.

The pascal program BINCONV performs these three tasks; a listing of the source code is given at the end of this application note. A flow diagram of BINCONV can be seen in Figure 2. The inclusion of the count byte has been left as an option to increase the flexibility of the program, but it could easily be standardised to include the count byte for the B6 RAM boot-loader. When BINCONV is invoked it prompts for the name of the S-record input file and the name required for the binary

output file. After this each S-record in the input file is read and converted to binary data and stored in a temporary file. As each S-record is read it is echoed to the screen; when they have all been processed a message prompts the user and asks if a count byte is required. When used with the 68HC05B6 RAM boot-loader the answer will always be yes, in which case the count value is written to the output file before the rest of the data is copied from the temporary file to the output file. Finally the value of the count byte is displayed for user confirmation – remember that the count byte is equal to the number of bytes in the program being converted plus 1 for the count byte itself. The program will only accept standard S-record format and will trap and abort if any non-valid character or format is detected.

With the PC COM port set for 9600 baud and the 68HC05B6 configured as in Figure 1 the binary file can be transferred and executed as follows:

- i) Release Reset on the HC05B6
- ii) Enter the command "COPYXXXX.YYY COM1\B" on the PC.

The program will then be transferred to the B6 and execution started automatically. Note that the B option is used to denote a binary file transfer so that the copy procedure does not abort if it finds an end of file (EOF) character in the middle of the file.

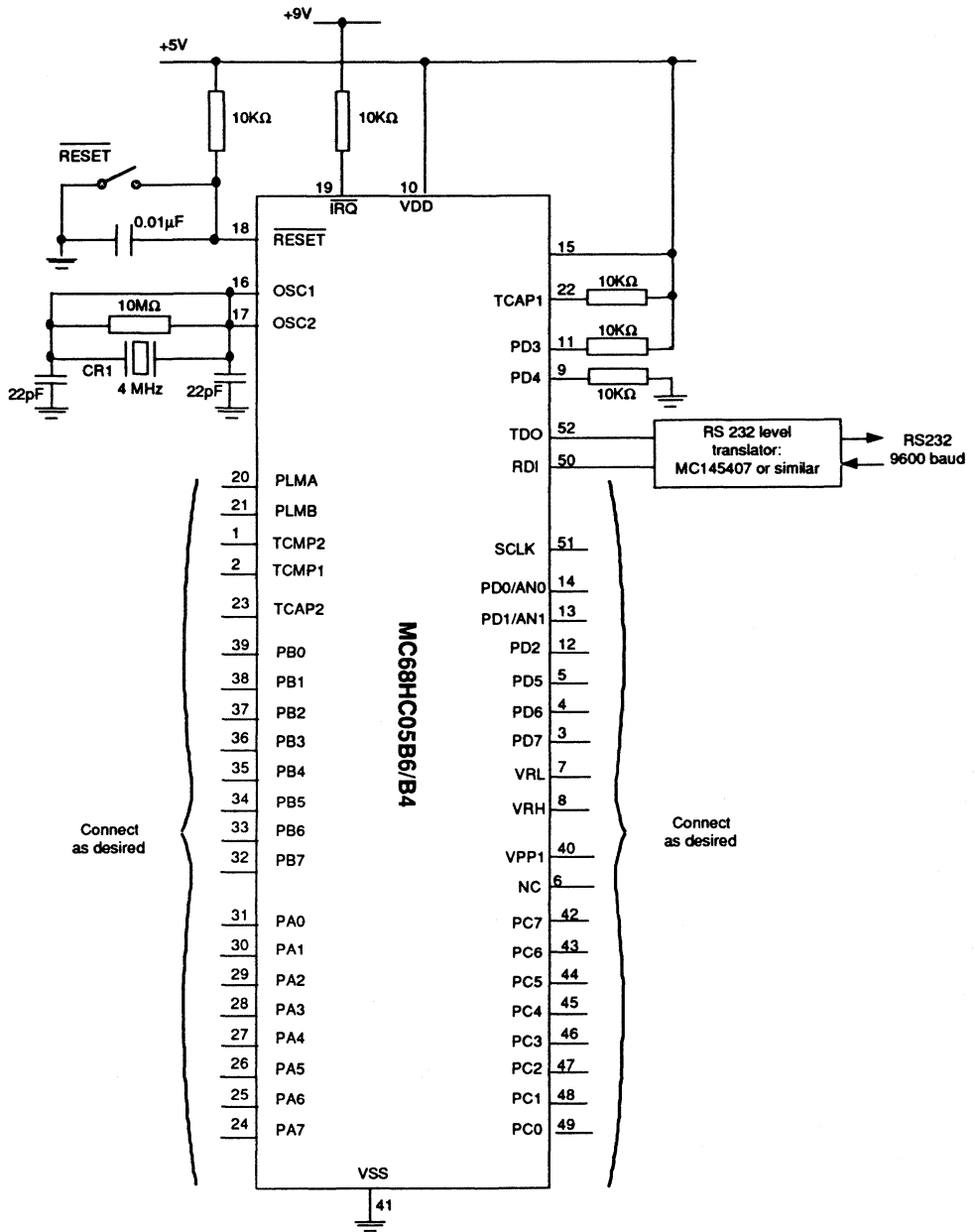


Figure 1. RAM bootstrap schematic

2) BOOTSTRAPPING THE EEPROM1

To bootstrap the EEPROM1 on the MC68HC05B6 in the absence of a built in loader program, use must be made of the RAM boot-loader described above. The idea is that an EEPROM1 loader can be written to the users exact requirements then assembled and downloaded into the RAM of the HC05B6 where it will execute and in turn download data and program it into the EEPROM1.

The 6K EEPROM emulation part, the MC68HC805B6, does have a built in EEPROM boot-loader in place of the RAM boot-loader and there is an accompanying PC program available from Motorola called E2B6 that downloads S-records to the device for programming.

The following is an explanation of an example EEPROM1 bootstrap program for the B6 that has been written to be compatible with the 805B6 PC program E2B6 thus eliminating the need to develop another PC program.

A listing of this program (EE1BOOT) is given at the end of this application note. The MC68HC05B6 has 176 bytes of RAM that can be used for the EEPROM1 bootstrap program, so the protocol must be kept simple and the code written efficiently. The format of the E2B6 program is a transfer of 2 address bytes followed by the data byte that is to be programmed at that location. At the same time the B6 returns the data from the previously programmed location for verification by E2B6. The program EE1BOOT has 4 main sections: a main loop, an erase routine, a program routine and an SCI service routine. The core of both the erase and program subroutines is the extended addressing subroutine EXTSUB which is used to access the EEPROM1 array. This subroutine is built in RAM by the main loop as the address information for the next byte to be programmed is received from the SCI. E2B6 always sends a null character during initialisation which could throw the EE1BOOT program out of synchronisation, as it is already executing before E2B6 is invoked. For this reason EE1BOOT ignores the first character received and treats the second as the first address byte.

The EXTSUB routine is first called as an "LDA \$aaaa" to retrieve the last byte programmed for verification.

Then the address in the routine is modified as the next address to be programmed is received. When the data byte is received the opcode of EXTSUB is incremented so that it becomes "STA bbbb" before the erase and program routines are called. After programming the opcode is decremented back to LDA before the main loop is repeated.

Note that the EEPROM1 location is always erased before programming. The timer output compare function is used to provide a 10ms delay for erasing and programming and the programming step is skipped to save time if the data presented to that location is \$FF. The sequence of events to bootstrap the EEPROM1 of the 68HC05B6 is therefore as follows:

- 1) Configure the 68HC05B6 as in Figure 1.
- 2) Assemble the program EE1BOOT and convert it to binary using BINCONV as described in section 1.
- 3) Set up PC COM port to 9600 baud then release Reset on the HC05B6.
- 4) Use the command "COPY EE1BOOT.BIN COM1/B" to download EE1BOOT into the RAM of the HC05B6. EE1BOOT will now start to execute.
- 5) Start the program E2B6 on the PC and follow the instructions to download the desired S-records to the EEPROM1 of the 68HC05B6.

Note:

- i) Only the download procedure of E2B6 will work in conjunction with EE1BOOT.
- ii) Once the EEPROM1 security bit has been set, the RAM boot-loader on the 68HC05B6 will no longer operate. This means that after the device has been reset it will be impossible to download any more data into the EEPROM1 until selfcheck has been executed - selfcheck performs an erase of the entire EEPROM1 array. This means that if the EEPROM1 is to be programmed in several steps, the one that will set the security bit should be done last.

FURTHER POSSIBILITIES

This application note has shown a method for initialising the EEPROM1 on the 68HC05B6 by using the RAM boot-loader. It would of course be much simpler to incorporate a EEPROM1 boot-loader in the ROM space of the user program, but often there is not

enough space. If enough space is available (117 bytes), then EE1BOOT could be incorporated in the application software, thus saving steps 2, 3 & 4 in the procedure above.

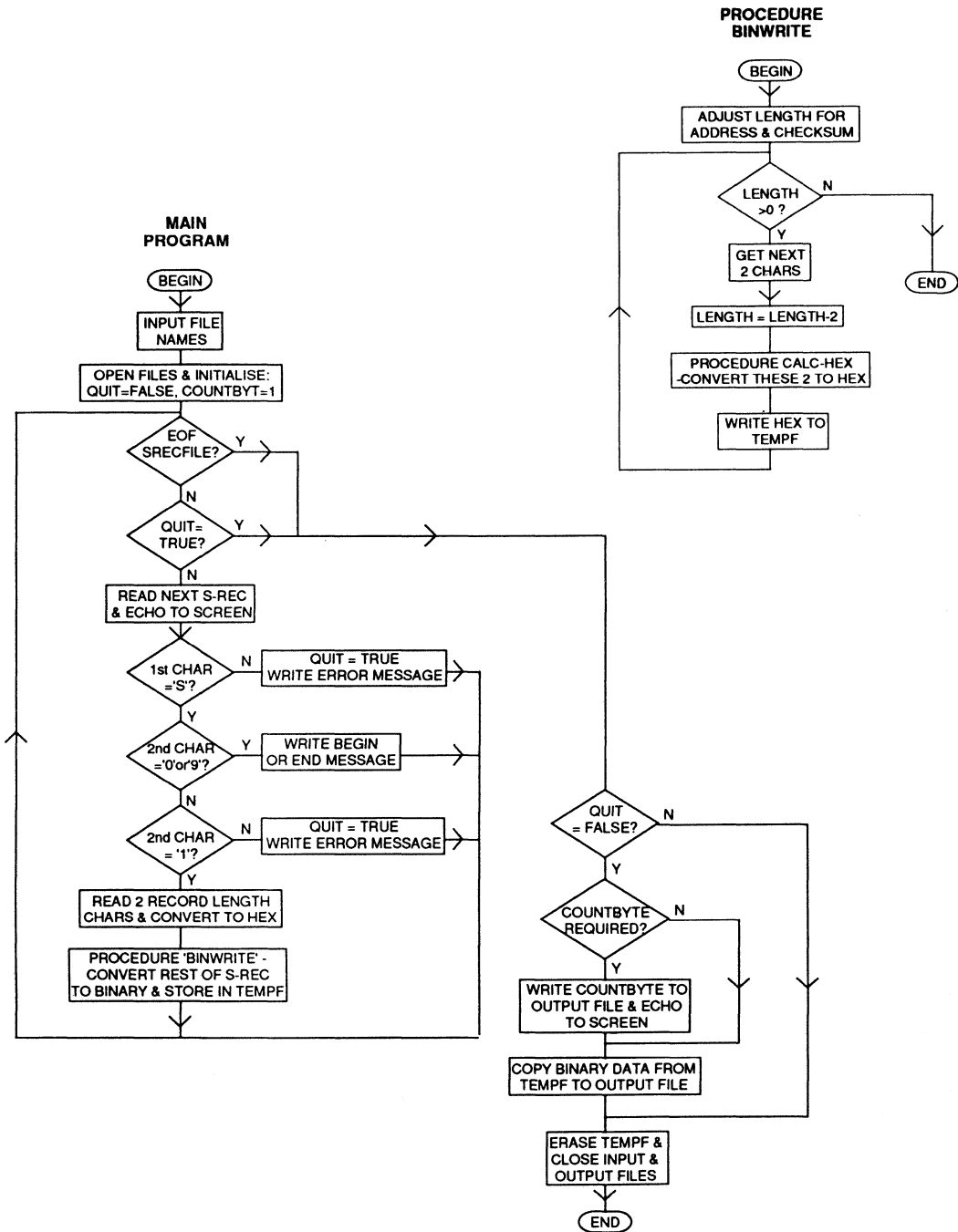


Figure 2. Flow diagram of BINCONV

```

0001      *
0002      *
0003      *%
0004      *%      EE1BOOT - 68HC05B6 EEPROM1 Serial bootloader      *%
0005      *%
0006      *%      - This prog. is loaded into the RAM of the HC05B6 via the RAM      *%
0007      *%      bootloader. The program will then start to execute. The format      *%
0008      *%      has been selected to be the same as that on the 805B6 so that      *%
0009      *%      the program E2B6 can be used to program the EEPROM1.      *%
0010      *% Note: E2B6 sends a null character during initialisaton so this prog      *%
0011      *%      ignores the first character received on the SCI.      *%
0012      *%
0013      *%
0014      *%      Jeff Wright      Last Updated 10/5/90      *%
0015      *%
0016      *
0017      *
0018      *
0019      *
0020      *
0021      *
0022      *
0023      *      I/O registers
0024      *
0025 0000      PORTA      EQU      $00      port A.
0026 0001      PORTB      EQU      $01      port B.
0027 0002      PORTC      EQU      $02      port C.
0028 0003      PORTD      EQU      $03      port D.
0029 0004      DDRA      EQU      $04      port A DDR.
0030 0005      DDRB      EQU      $05      port B DDR.
0031 0006      DDRC      EQU      $06      port C DDR.
0032
0033 0007      EECONT      EQU      $07
0034 0002      E1ERA      EQU      2
0035 0001      E1LAT      EQU      1
0036 0000      E1PGM      EQU      0
0037
0038 000d      BAUD      EQU      $0D
0039 000e      SCCR1      EQU      $0E
0040 0004      M8BIT      EQU      4
0041 000f      SCCR2      EQU      $0F
0042 0010      SCSR      EQU      $10
0043 0005      RDRF      EQU      5
0044 0011      SCDAT      EQU      $11
0045      *
0046      *      TIMER registers
0047      *
0048 0012      TCR      EQU      $12      Timer control register.
0049 0005      TOIE      EQU      5      Timer overflow interrupt enable.
0050 0006      OCIE      EQU      6      Timer output compares interrupt enable.
0051 0007      ICIE      EQU      7      Timer input captures interrupt enable.
0052

```

```

0053 0013          TSR          EQU    $13      Timer status register.
0054 0003          OCF2         EQU    3        Timer output compare 2 flag.
0055 0004          ICF2         EQU    4        Timer input capture 2 flag.
0056 0005          TOF          EQU    5        Timer overflow flag.
0057 0006          OCF1         EQU    6        Timer output compare 1 flag.
0058 0007          ICF1         EQU    7        Timer input capture 1 flag.
0059
0060 0016          TOC1HI        EQU    $16      Timer output compare register 1 (16-bit).
0061 0017          TOC1LO        EQU    $17
0062 0018          TIMHI         EQU    $18      Timer free running counter (16-bit).
0063 0019          TIMLO         EQU    $19
0064
0065          ***** MISC DEFINITIONS -----
0066
0067 00c6          LDAEXT         EQU    $C6      OP-Code for LDA extended.
0068 0014          MS10          EQU    $14      10mS delay constant.
0069
0070          *
0071
0072          *****
0073          *                               *
0074          *          START OF CODE          *
0075          *                               *
0076          *****
0077
0078 0051          ORG    $51
0079
0080 0051 a6 00          RESET          LDA    #$00
0081 0053 b7 04          STA    DDRA      All Ports inputs.
0082 0055 b7 05          STA    DDRB
0083 0057 b7 06          STA    DDRC
0084
0085 0059 19 0e          SCIINT         BCLR   MBIT,SCCR1  Initialise SCI - 8 data bits.
0086 005b a6 c0          LDA    #$C0
0087 005d b7 0d          STA    BAUD      9600 baud at 4MHz.
0088 005f a6 0c          LDA    #$0C      Enable transmit and receive.
0089 0061 b7 0f          STA    SCCR2
0090 0063 b7 10          STA    SCSR      Clear pending flags.
0091 0065 a6 c6          LDA    #LDAEXT    Init extended addressing subroutine to LDA.
0092 0067 c7 00 8f          STA    OPCDE
0093 006a ad 1d          BSR    SCREAD      Wait here and ignore 1st char (E2B6 init).
0094
0095 006c ad 21          LOOP          BSR    EXTSUB    Load Acc with data from last programmed addr
0096 006e b7 11          STA    SCDAT      Send it back for host to verify.
0097 0070 ad 17          BSR    SCREAD      Get high address
0098 0072 c7 00 90          STA    ADDHI      - and store it.
0099 0075 ad 12          BSR    SCREAD      Get low address
0100 0077 c7 00 91          STA    ADDLO      - and store it.
0101 007a ad 0d          BSR    SCREAD      Get the data to be programmed
0102 007c c7 00 93          STA    DATA      Store it temporarily.
0103 007f 3c 8f          INC    OPCDE      Change the ext addr subroutine to STA aaaa.
0104 0081 ad 11          BSR    ERASEE      Erase the selected address for 10ms.
0105 0083 ad 27          BSR    PROGEE      Now prog the data for 10mS.
0106 0085 3a 8f          DEC    OPCDE      Restote ext addr subroutine to LDA aaaa.
0107 0087 20 e3          BRA    LOOP
0108

```

```

0109          ***** SUBROUTINE TO SERVICE SCI *****
0110
0111 0089 0b 10 fd      SCREAD      BRCLR  RDRF,SCSR,*
0112 008c b6 11        LDA      SCDAT
0113 008e 81          RTS
0114
0115
0116          ***** EXTENDED ADDRESSING SUBROUTINE TO ACCESS FULL MEMORY MAP *****
0117
0118 008f          EXTSUB      EQU      *
0119 008f 00          OPCDE      FCB      0
0120 0090 00          ADDHI      FCB      0
0121 0091 00          ADDLO      FCB      0
0122 0092 81          RTS
0123
0124 0093 00          DATA      FCB      0                      Reserved Byte for data during erasing.
0125
0126          ***** EEL ERASING SUBROUTINE *****
0127
0128 0094 12 07      ERASEE      BSET      E1LAT,EECONT
0129 0096 14 07      BSET      E1ERA,EECONT
0130 0098 ad f5      BSR      EXTSUB
0131 009a a6 14      LDA      #MS10
0132 009c 10 07      DEL1      BSET      E1PGM,EECONT
0133 009e b7 19      STA      TIMLO          Set up timer for a 10ms count
0134 00a0 b7 16      STA      TOC1HI
0135 00a2 b7 13      STA      TSR           - using output compare 1 function.
0136 00a4 b7 17      STA      TOC1LO
0137 00a6 0d 13 fd  BRCLR      OCF1,TSR,*      Wait here for end of erase time
0138 00a9 3f 07      CLR      EECONT        - erase finished.
0139 00ab 81          RTS
0140
0141          ***** EEL PROGRAMMING SUBROUTINE *****
0142
0143 00ac 12 07      PROGEE      BSET      E1LAT,EECONT
0144 00ae b6 93      LDA      DATA
0145 00b0 ad dd      BSR      EXTSUB
0146 00b2 4c          INCA
0147 00b3 27 0f      BEQ      SKIP          Skip programming if data = 5FF
0148 00b5 a6 14      LDA      #MS10
0149 00b7 10 07      DEL      BSET      E1PGM,EECONT
0150 00b9 b7 19      STA      TIMLO          Set-up timer for 10mS count
0151 00bb b7 16      STA      TOC1HI
0152 00bd b7 13      STA      TSR           - using output compare 1 function.
0153 00bf b7 17      STA      TOC1LO
0154 00c1 0d 13 fd  BRCLR      OCF1,TSR,*      Wait here for programming to finish.
0155 00c4 3f 07      SKIP      CLR      EECONT
0156 00c6 81          RTS

```

```

{*****}

program BINCONV; { Program to convert Motorola S-record files to
                 binary format. Optional inclusion of a count byte for
                 HC05B6 RAM bootloader etc}
                 { Programmer - Jeff Wright, MCU applications
                   Motorola
                   East Kilbride}

{
                 Last Updated 10/5/90}

{*****}

var
  SrecFile : text;
  BinFile  : file;
  Tempf    : file;
  srec     : string[100];
  Transfer : array[1..20000] of char;
  numread, numwritten : word;
  answer   : char;
  fnamei   : string[15];
  fnameo   : string[15];
  bytout   : char;
  countbyt : integer;
  datcnt   : integer;
  datval   : integer;
  point    : integer;
  cnt1     : integer;
  cnt2     : integer;
  quit     : boolean;
  Count    : boolean;

(-----)

Procedure Calc_hex(chr1,chr2 : integer);

{Combines 2 characters into a single byte value i.e A5->165, error
signaled if non hex character detected}

Begin
Case chr1 of
48..57 : chr1 := chr1 - 48;
65..70 : chr1 := chr1 - 55; { Is this a valid hex character?}
else
begin
writeln ('invalid data - conversion aborted');
quit := true
end
end;
Case chr2 of
48..57 : chr2 := chr2 - 48;
65..70 : chr2 := chr2 - 55;
else
begin
writeln ('invalid data - conversion aborted');
quit := true
end
end
end

```

```

end;
datval := chr1*16 + chr2;    {Convert to single byte}
end;

{-----}

Procedure Binwrite(length,dpoint : integer);

{Converts an S-record line to hex and stores it in a temporary file}

begin
length := length-3;        {Allow for address and checksum bytes}
countbyt := countbyt+length;    {Update running byte total}
length := length*2;        {Twice as many characters as bytes}
while length > 0 do
begin
cnt1 := Ord(srec[dpoint]);    {Get the next two characters}
cnt2 := Ord(srec[dpoint+1]);
dpoint := dpoint+2;          {Update pointer and length}
length := length-2;

Calc_hex(cnt1,cnt2);        {Convert two characters into single byte}
bytout := Chr(datval);      {- now convert that single byte into a }
blockwrite (tempf,bytout,1)  {character and save it in temporary file}

end
end;

{***** MAIN PROGRAM STARTS BELOW *****)

begin
writeln ('S-record to Binary conversion utility');
writeln;
writeln;
write('Input S-record file name? -> ');
readln(fnamei);
assign(SrecFile, fnamei);
write(' Binary output file name? -> ');
readln(fnameo);
assign(BinFile, fnameo);
assign(tempf, 'temp.tmp');
quit := false;
countbyt := 1;
Reset(SrecFile);           {open the two }
Rewrite(BinFile,1);       { -selected files}
Rewrite(tempf,1);        { + a temporary file}

```

```

while not Eof(SrecFile) and not quit do
begin
  readln(SrecFile, srec);  {read S-rec into char string srec}
  writeln(srec);

  If srec[1]='S'then      {If string does not start with S then quit}
  begin
    CASE srec[2] of
      '1' :              {If not S1 record then loop back}
      begin
        cnt1 := Ord(srec[3]);  {get the 2 record length}
        cnt2 := Ord(srec[4]);  {characters}
        calc_hex(cnt1,cnt2); {func to produce hex in
                               datcnt from cnt1 & 2}

        datcnt := datval;
        point := 9;           {point to first data character}
        binwrite(datcnt,point) { convert the data in this s-rec
                                line to binary and store in temp file}
      end;
      '0' : writeln ('Conversion started');
      '9' : writeln ('last S-record done');
    else
      begin {If not S0,S1orS9 record then abort}
        quit := true;
        writeln ('Non standard S-record detected - Conversion aborted')
      end
    end
  end
  end
  else
  begin {If 1st char not an S then abort}
    quit := true;
    writeln ('Non standard S-record detected - Conversion aborted')
  end
end;

If quit = false then

{If no errors then copy the temporary file to the output file and add in
a count byte if required}

begin
  Reset (tempf,1);
  writeln;
  write ('Do you want a count byte added to start of output file? -> ');
  readln (answer);
  If upcase(answer) = 'Y' then
  Begin
    writeln ('Total size including count byte = ',countbyt);
    bytout := chr(countbyt);
    blockwrite (binfile,bytout,1)
  end;
  repeat
    blockread (tempf,transfer,sizeof(transfer),numread);
    blockwrite (binfile,transfer,numread,numwritten);
  until (numread=0) or (numwritten <> numread)
end;

close(tempf);
erase(tempf);  {Finished with temporary file so erase it}
close(SrecFile);
close(BinFile) {Close files before quitting}
end.

```


Error Detection and Correction Routines for M68HC05 devices containing EEPROM

By Ken Terry
MCU Applications Group
Motorola Ltd
East Kilbride

INTRODUCTION

An increasing number of applications involving MC68HC05 MCUs require large amounts of critical data to be stored in EEPROM memory. This application note describes software routines, generated for the HC05, which allow stored data to be encoded so that single bit errors existing in retrieved data may be corrected and two bit errors detected. The routines use a simple Linear Block Code for the encoding of stored data.

SINGLE BIT ERROR CORRECTION

All methods of error detection/correction involve the use of extra check bits added to the data bits to produce some form of codeword. To allow the detection of a single bit error in a specific codeword it is necessary that each word differs from any other word by at least two digits. A one bit error will then produce an invalid word. The number of digits by which two words, of the same length, differ is defined as the Hamming Distance. For the correction of up to t errors a minimum Hamming Distance of $2t + 1$ is required between each codeword. Single bit error correction and double bit error detection requires a minimum distance of 3. The problem is to decide what an original codeword was if an invalid codeword has been detected. One means of doing this is to use a Linear Block Code, as described below. Linear Block Codes for the correction of single bit errors are referred to as Hamming Codes. The following describes a systematic method for single bit error correction.

A codeword consists of k data digits to which are added r check digits to produce an n digit codeword ($n = k+r$). The r data digits are redundant, in that they carry no additional data, and the code efficiency is defined as k/n . This is an indication of the amount of information transferred, relative to the total number of bits.

For a linear block code the general codeword can be written in the form:

$$a_1 a_2 a_3 \dots a_k c_1 c_2 \dots c_r$$

where a_1 to a_k are the k data digits and c_1 to c_r are the r check digits.

The check digits are chosen to satisfy the r linear equations:

$$0 = h_{11}a_1 \oplus h_{12}a_2 \oplus \dots \oplus h_{1k}a_k \oplus c_1$$

$$0 = h_{21}a_1 \oplus h_{22}a_2 \oplus \dots \oplus h_{2k}a_k \oplus c_2$$

.

.

.

$$0 = h_{r1}a_1 \oplus h_{r2}a_2 \oplus \dots \oplus h_{rk}a_k \oplus c_r$$

Each element in the above equations is either a one or a zero and all addition is modulo 2.

These equations can be more conveniently expressed in terms of the matrix equation:

$$[H] [T] = 0$$

where [T] is an $n \times 1$ column vector representing the stored codeword:

$$[T] = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_k \\ c_1 \\ c_2 \\ \vdots \\ c_r \end{bmatrix}$$

and [H] is an $r \times n$ matrix, referred to as the parity check matrix.

$$[H] = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1k} & 1 & 0 & \dots & 0 \\ h_{21} & h_{22} & \dots & h_{2k} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ h_{r1} & h_{r2} & \dots & h_{rk} & 0 & 0 & \dots & 1 \end{bmatrix}$$

A second column vector [R], with same dimensions as [T], is used to represent the retrieved codeword. This may or may not be equal to the original stored codeword [T], depending on whether or not an error exists. If [H][R] = 0, then [R] is most likely to be the original stored codeword. If [H][R] gives a non zero value then at least one error has occurred. If an $n \times 1$ error matrix [E] is introduced, then the retrieved codeword [R] can be written as:

$$[R] = [T] + [E]$$

If [E] consists totally of zeros then no error has occurred. For any error that does occur in [R], [E] will contain a '1' in the corresponding position. The problem is then to determine where in [E] the non zero elements are, once the codeword [R] has been retrieved. A matrix [S], referred to as the syndrome, is defined such that:

$$[S] = [H][R]$$

This can be expanded to

$$[S] = [H][T] + [H][E]$$

giving

$$[S] = [H][E]$$

[S] is an $r \times 1$ column matrix and can consist of any one of 2^r sequences. [E] is an $n \times 1$ matrix and can consist of any one of 2^n sequences. As $n > r$ there is no unique solution to the above equation. However, in this case it is assumed that only one error has occurred and therefore [E] contains only one non zero element. Multiplying [E] by [H] yields a syndrome which will be equal to one column within [H]. The position of this column will indicate where the non zero element exists in [E] and hence the position of the single bit error in [R]. In the case of two or more non-zero elements in [E] error correction is not possible.

HAMMING BOUND AND CODE EFFICIENCY

The Hamming Bound is defined as: $2^r \geq k + r + 1$

where k is the number of data bits and r is the number of check bits.

This must be satisfied for single bit error correction. To allow double bit error detection a further check bit must be added. Table 1 shows the number of check bits required, along with the corresponding code rate, for single bit error correction and double bit error detection in different numbers of data bits.

It can be seen from the table that, in general, the greater the number of data bits the greater the code efficiency. However as the size of the codeword increases the calculations involved in detecting an error become increasingly more cumbersome. It can also be seen that for both 8 and 11 data bits, the number of check bits required is 5. By using 11 data bits and 5 check bits the Hamming bound can be satisfied exactly. There is no exact solution when 8 data bits are used. However, this is a more convenient data size for an 8-bit MCU and is therefore used in this application, despite the lower code efficiency.

No. of Data Bits (k)	No. of Check Bits	Code Efficiency
4	4	50%
8	5	61.5%
11	5	68.7%
26	6	83.9%

Table 1. Check Bit Requirements and Code Efficiency for Single Error Correction

CODEWORD GENERATION AND STORAGE

For one byte of data, 4 check bits are required for single bit error correction. The parity check matrix will consist of 12 columns of 4 bits and can be simply generated by taking the binary values \$1 to \$C (represented as binary column vectors) to generate 12 columns as shown. The check bits, c1 to c4, are assigned to the columns containing a single non zero entry and the data bits, b7 to b0, are assigned to the remaining columns. The order of assignment is completely arbitrary.

$$[H] = \begin{bmatrix} c1 & c2 & b7 & c3 & b6 & b5 & b4 & c4 & b3 & b2 & b1 & b0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The following equations can then be derived from the parity check matrix and used to calculate c1 to c4.

$$c1 = b7 \oplus b6 \oplus b4 \oplus b3 \oplus b1$$

$$c2 = b7 \oplus b5 \oplus b4 \oplus b2 \oplus b1$$

$$c3 = b6 \oplus b5 \oplus b4 \oplus b0$$

$$c4 = b3 \oplus b2 \oplus b1 \oplus b0$$

At no time is the application software required to carry out any matrix multiplication. This is done implicitly by the use of the above equations. A fifth check digit, c5, is used to detect the occurrence of a double bit error and is a simple parity check (even parity) for the 12 bit codeword formed by concatenating b7–b0 and c1–c4.

Figure 1 shows the data organisation in memory. The data bytes (b0–b7) and the corresponding check bits (c1–c5) are stored separately in adjoining blocks of EEPROM. This allows executable code to be stored in the EEPROM and protected using error checking. The data EEPROM block is 256 bytes long. It is immediately followed by the check EEPROM block. The minimum size possible for the check EEPROM is 160 bytes (256 x 5 bits). In order that all check bits can be accommodated within this, software routines are required for the 'packing' and 'unpacking' of check bits.

DATA

DATA + \$100

DATA + \$190

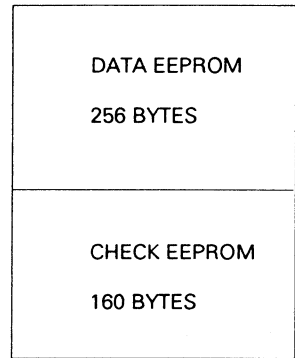


Figure 1. Data Organisation in Memory

DATA RETRIEVAL AND CORRECTION

To allow a retrieved codeword to be checked it is necessary to generate the syndrome [S]. To do this the retrieved data byte is used to generate a new set of check bits, c1'–c4', using the same set of equations as above. The syndrome is then generated by exclusive ORing c1' to c4' with c1 to c4. A non zero result will indicate the presence of an error. The parity check c5' is calculated from the retrieved data and check bits and compared with c5. If they are the same, and the syndrome indicates the presence of an error, then it is assumed that a double error has occurred and can therefore not be corrected. If the error is correctable then the syndrome can be compared with values corresponding to the columns of [H] (in this case, a simple lookup table in ROM) to determine the error position.

SOFTWARE

An assembled listing of the software is included at the end of this application note.

The software has been written to run on the MC68HC05SC21 but can be easily modified to run on any HC05 MCU with EEPROM. It comprises 2 main routines. The first routine is CHECKPROG and this generates the codeword from the data and programs the data and the appropriate check bits into EEPROM.

The second routine, GETCHECK, retrieves the data and check bits from the EEPROM, calculates the syndrome and, if any error is detected, returns with the error position indicated in the accumulator. The detection of a double bit error by GETCHECK is indicated by the carry bit being set on return from the routine. The data and check EEPROM blocks can be placed anywhere within the device EEPROM memory, the start address of the data EEPROM being determined by an address held in RAM registers EPSTHI and EPSTLO.

Four further subroutines are called by the the main routines. PAKCHK and UNPAKCHK are used for the packing and unpacking of the check bits in the check EEPROM block. CHECKBIT is used to calculate the check bits c_1 to c_4 and c_1' to c_4' . CALC5 calculates the parity checks c_5 and c_5' .

The total ROM requirement for the routines is 301 bytes with a further 56 bytes required for the EEPROM write/erase routines. Execution time for the routine GETCHECK is approximately 0.6 ms (with 2 MHz internal bus frequency). The execution time for CHECKPROG is dependant on the EEPROM programming time. The time required for the calculation and packing of the check bits amounts to approximately 0.6 ms.

REFERENCES

Carlson, 'Communication Systems', McGraw Hill.

```

0001 *****
0002 *****
0003 *          MC68HC05SC21 - EEPROM ERROR CHECK CODING ROUTINES
0004 *
0005 *****
0006 *          This software was developed by Motorola Ltd. for demonstration
0007 *          purposes only. Motorola does not assume liability arising out of
0008 *          the application or use of this software and does not guarantee
0009 *          its functionality.
0010 *          Original software copyright Motorola - all rights reserved.
0011 *****
0012 *
0013 *          16/10/90
0014 *
0015 *****
0016 *          These routines use a modified (12,8) Hamming code to provide
0017 *          single bit error correction and double bit error detection for
0018 *          data stored in EEPROM. The data is segmented into blocks of 256
0019 *          bytes. Each 256 block of 'data' EEPROM is immediately followed
0020 *          by 160 bytes of 'check' EEPROM which contains the parity check
0021 *          bits.
0022 *****
0023 *          BYTE EQUATES
0024 *****
0025 *
0026 0004 DDRA      EQU    $04
0027 0000 PORTA      EQU    $00
0028 0001 PORTB      EQU    $01          PORTB
0029 0005 DDRB      EQU    $05          PORT B DATA DIRECTION REGISTER
0030 0008 MISC       EQU    $08          MISC register
0031 0009 PCR       EQU    $09          Program Control Register
0032 *****
0033 *          USER EQUATES
0034 *****
0035 *
0036 0080 ADSTA      EQU    $80          Start add. of RAM subroutine area for STA inst.
0037 *          or LDA inst.
0038 0081 EPRADH     EQU    $81          Adr. EEPROM high for EEPROM write routine
0039 0082 EPRADL     EQU    $82          Adr. EEPROM low for EEPROM write routine
0040 0090 SAVA       EQU    $90          General purpose RAM reg. to store acc.
0041 0091 SAVX       EQU    $91          General purpose RAM reg. to store x-reg.
0042 *
0043 0092 DATA      EQU    $92          Data reg. contains data word to be encoded
0044 0093 INDEX      EQU    $93
0045 0094 CHECK0      EQU    $94          Holds check bits c1 to c5 for byte in DATA
0046 0095 CHECK1      EQU    $95          Used to generate check bits
0047 0096 CHECK3      EQU    $96
0048 0097 CHECK4      EQU    $97
0049 0098 REM       EQU    $98
0050 0099 EPSTHI     EQU    $99
0051 009a EPSTLO     EQU    $9A
0052 009b SYNDROME   EQU    $9B
0053 *
0054 1100 ERRCOR     EQU    $1100
0055 *
0056 *****
0057 *          BIT EQUATES
0058 *****
0059 *
0060 *          PORT A SERIAL I/O PORT
0061 0000 SERIO      EQU    0          Serial i/o port - port A bit 0
0062 *
0063 *          MISC Register
0064 0007 ROMPG      EQU    7
0065 0006 INTFF      EQU    6
0066 0004 DCTST      EQU    4
0067 *

```

```

0068          *          Program Control Register
0069 0007      WE          EQU      7
0070 0002      VPON       EQU      2
0071 0001      PGE        EQU      1
0072 0000      PLE        EQU      0
0073          *
0074          *
0075          *****
0076          *
0077          *
0078 1100      ORG          ERRCOR
0079          *
0080          *
0081          *****
0082          *          CHKPROG - This routine programs a byte of data held in acc. into a
0083          *          block of data EEPROM. Data EEPROM is 256 bytes long and starts from
0084          *          an address held in EPSTHI and EPSTLO. Location of data byte within
0085          *          data EEPROM is determined by X-reg value. Check bits C1 to C4 are
0086          *          calculated for the data byte, using a (12,8) block code, to allow
0087          *          the correction of a single bit error by the routine GETCHECK. A
0088          *          further simple parity check bit, C5, is generated to allow the
0089          *          detection of double bit errors. The check bits are programmed as a
0090          *          5 bit block into the check EEPROM, which is 160 byte long and starts
0091          *          from location EPSTHI,EPSTLO + $100.
0092          *
0093          *          Enter with data to be programmed in acc., start add. of data EEPROM
0094          *          in EPSTHI and EPSTLO and index value for data address in X-reg.
0095          *
0096          *          Returns with X-reg. value saved.
0097          *
0098          *****
0099 1100 b7 92  CHKPROG   STA      DATA      Store data byte
0100 1102 bf 93          STX      INDEX      Save data address index value
0101          *
0102 1104 b6 99          LDA      EPSTHI     Set up address offset for EPRWRT
0103 1106 b7 81          STA      EPRADH    - (EEPROM write routine)
0104 1108 b6 9a          LDA      EPSTLO
0105 110a b7 82          STA      EPRADL
0106 110c b6 92          LDA      DATA      Restore data byte into acc.
0107          *
0108 110e cd 12 33      JSR      EPRWRT     Store data byte in EEPROM at address
0109          *          specified by EPSTHI,EPSTLO + x reg.
0110          *
0111 1111 cd 11 c1      JSR      CHECKBIT    Calculate check bits C1 to C4.
0112          *          Returns C1 to C4 in CHECK0 (bits 0 - 3)
0113          *
0114 1114 cd 11 e4      JSR      CALC5      Calculate C5 and return with C5 in CHECK0(4)
0115          *
0116 1117 cd 12 09      JSR      PAKCHK     Calculate offset req'd to give byte location
0117          *          for C1 to C5 and store in X-reg. and rotate
0118          *          CHECK0 and CHECK1 so that C1 to C5 will be
0119          *          programmed into appropriate part of check
0120          *          EEPROM
0121          *
0122 111a 3c 81          INC      EPRADH     Increment start add. of data EEPROM by $100
0123          *          to get start add. of check EEPROM
0124          *
0125 111c b6 94          LDA      CHECK0
0126 111e cd 12 33      JSR      EPRWRT     Program CHECK0 into Check EEPROM
0127 1121 b6 95          LDA      CHECK1
0128 1123 5c          INCX
0129 1124 cd 12 33      JSR      EPRWRT     INC X-reg. to get add. for CHECK1
0130          *          Program CHECK1 into EEPROM
0131 1127 be 93          LDX      INDEX     Restore X-reg. value
0132 1129 81          RTS
0133          *
0134          *
0135          *****

```

```

0136 * GETCHECK - Retrieves a data byte from location EPSTHI,EPSTLO + x
0137 * along with corresponding check bits C1 to C5. The data, and C1 to C4,
0138 * are used to calculate the SYNDROME value which is used to indicate
0139 * the position of a single bit error. The SYNDROME value is generated
0140 * by calculating new check bit values C1' to C4' from the retrieved
0141 * data and adding these (modulo 2) to the original check bit values
0142 * retrieved from the Check EEPROM. C5 and C5' are simple parity check
0143 * bits used to indicate the occurrence of a 2 bit error.
0144 *
0145 * Enter with start address of data block in EPSTHI and EPSTLO and index
0146 * value for data byte in X-reg.
0147 *
0148 * Returns with the uncorrected data byte in RAM location DATA. Error
0149 * status is indicated by the following:
0150 *
0151 * No errors - carry = 0, acc. = 0.
0152 *
0153 * Single bit error in data byte - carry = 0, acc. has one bit set to
0154 * indicate the position of the error in the data byte.
0155 *
0156 * Single bit error in check bits - carry = 0, acc has upper nybble = $F
0157 * and one bit set in lower nybble to indicate check bit error position.
0158 * (b0 indicates error in C1, b3 indicates error in C4).
0159 *
0160 * Double bit error - carry = 1.
0161 *
0162 * X-reg. contents are saved.
0163 * *****
0164 *
0165 112a GETCHECK EQU *
0166 *
0167 112a bf 93 STX INDEX Store data address offset
0168 *
0169 112c b6 99 LDA EPSTHI Set up address offset for GETBYTE
0170 112e b7 81 STA EPRADH
0171 1130 b6 9a LDA EPSTLO
0172 1132 b7 82 STA EPRADL
0173 *
0174 1134 cd 11 71 JSR GETBYTE Get data byte from loc'n EPSTHI,EPSTLO + X
0175 1137 b7 92 STA DATA Store retrieved data byte
0176 *
0177 1139 cd 11 88 JSR UNPAKCHK Get check bits and return with C1 to C5
0178 * in CHECK3 (0 to 4)
0179 113c b7 96 STA CHECK3
0180 *
0181 113e cd 11 c1 JSR CHECKBIT Calculate new check bits (C1' to C4') from
0182 * retrieved data byte and return with them in
0183 * CHECK0 (0:3)
0184 *
0185 1141 b6 94 LDA CHECK0
0186 1143 b7 97 STA CHECK4 Store C1' to C4'
0187 *
0188 *
0189 1145 b6 96 LDA CHECK3
0190 1147 b7 94 STA CHECK0
0191 *
0192 *
0193 *
0194 1149 cd 11 e4 JSR CALC5 Calculate new parity check bit C5' from
0195 * retrieved data and check bits (C1 to C4)
0196 * and return with C1 to C4 in CHECK0 (0 to 3)
0197 * and C5' in CHECK0 (bit 4)
0198 *
0199 114c b6 96 LDA CHECK3 Load C1 to C5
0200 114e b8 97 EOR CHECK4 Generate Syndrome
0201 1150 a4 0f AND #$0F Mask out C5 from syndrome
0202 1152 27 15 BEQ NOERR If syndrome = 0 then no error
0203 1154 b7 9b STA SYNDROME Store syndrome

```



```

0204      *
0205      *
0206      LDA      CHECK0      Load C1 to C4 and C5' into acc.
0207      AND      #$10      Mask out C1 to C4 - leave C5'
0208      BNE      CORR1      Branch if C5' = 1
0209      BRCLR   4,CHECK3,DOUBLERR  If C5 = C5' = 0 then 2 bit error exists
0210      BRA      FNDERR      If C5 = 1 and C5' then correctable
0211      *
0212      CORR1   BRSET   4,CHECK3,DOUBLERR  If C5 = C5' = 1 then 2 bit error exists
0213      *
0214      *
0215      *
0216      *
0217      FNDERR  LDX      SYNDROME
0218      LDA      BITPNT-1,X  Set bit in acc. to indicate pos. of error
0219      NOERR    CLC          Clear carry to indicate that double bit
0220      *
0221      LDX      INDEX      Restore X-reg. value
0222      RTS
0223      *
0224      DOUBLERR SEC          Set carry to indicate double error
0225      *
0226      LDX      INDEX      Restore X-reg. value
0227      RTS
0228      *
0229      *
0230      GETBYTE  EQU      *
0231      *
0232      LDA      #$D6      RAM Subroutine to load acc. with byte held
0233      STA      ADSTA      in EPRADH,EPRADL + x
0234      LDA      #$81      Indexed 2 byte offset LDA inst.
0235      STA      ADSTA+3    RTS inst.
0236      JSR      ADSTA
0237      RTS
0238      *
0239      BITPNT  FCB      $F1      Error in C1
0240      FCB      $F2      Error in C2
0241      FCB      $80      Error in b7
0242      FCB      $F4      Error in C3
0243      FCB      $40      Error in b6
0244      FCB      $20      Error in b5
0245      FCB      $10      Error in b4
0246      FCB      $F8      Error in C4
0247      FCB      $08      Error in b3
0248      FCB      $04      Error in b2
0249      FCB      $02      Error in b1
0250      FCB      $01      Error in b0
0251      *
0252      *
0253      *
0254      *
0255      *
0256      *
0257      *
0258      *
0259      *
0260      *
0261      *
0262      *
0263      *
0264      *
0265      *
0266      *
0267      UNPAKCHK LDX      INDEX
0268      LDA      #$05
0269      MUL
0270      *
0271      *

```

```

0272 118d 3f 98          CLR      REM      Initialise remainder RAM reg.
0273                    *
0274                    *      Divide result in X-reg. and acc. by 8 and store remainder in
0275                    *      RAM byte REM
0276                    *
0277 118f 56            RORX
0278 1190 46            RORA
0279 1191 24 02         BCC      UPCHKAD1
0280 1193 10 98         BSET    0,REM
0281 1195 56            UPCHKAD1 RORX
0282 1196 46            RORA
0283 1197 24 02         BCC      UPCHKAD2
0284 1199 12 98         BSET    1,REM
0285 119b 56            UPCHKAD2 RORX
0286 119c 46            RORA
0287 119d 24 02         BCC      UPCHKAD3
0288 119f 14 98         BSET    2,REM
0289                    *
0290 11a1 97            UPCHKAD3 TAX      Store index for checkbits in X-reg.
0291 11a2 3c 81         INC      EPRADH   Inc. GETBYTE address offset by $100
0292 11a4 cd 11 71     JSR      GETBYTE  Get 1st byte of C1 to C5 string
0293 11a7 b7 94         STA      CHECK0
0294 11a9 5c            INCX
0295 11aa cd 11 71     JSR      GETBYTE  Inc Index by 1
0296 11ad b7 95         STA      CHECK1  Get 2nd byte of C1 to C5 string
0297                    *
0298                    *
0299 11af 3d 98         TST      REM
0300 11b1 27 09         BEQ      UPNOREM
0301 11b3 98            UPCHKAD4 CLC
0302 11b4 36 95         ROR      CHECK1   Rotate CHECK0 and CHECK1 left
0303                    *      - the no. of left shifts is equal
0304                    *      to the value held in REM
0305 11b6 36 94         ROR      CHECK0   - msb of CHECK0 is shifted into lsb
0306 11b8 3a 98         DEC      REM      of CHECK1.
0307 11ba 26 f7         BNE      UPCHKAD4
0308                    *
0309 11bc b6 94         UPNOREM LDA    CHECK0
0310 11be a4 1f         AND     #$1F      Mask out non valid check bits
0311 11c0 81            RTS
0312                    *
0313                    *      CHECKBIT - Calculates the checkbits C1 to C4
0314                    *
0315                    *      Enter with data in RAM reg. DATA
0316                    *
0317                    *      Returns with C1 to C4 in CHECK0 (0 - 3).
0318                    *      Acc., X-reg. contents not saved.
0319                    *
0320                    *      *****
0321                    *
0322                    *
0323 11c1 3f 94         CHECKBIT CLR    CHECK0   Clear CHECK0 reg.
0324                    *
0325 11c3 b6 92         LDA     DATA      Load data byte into acc.
0326 11c5 a4 0f         AND     #00001111  c4 = b3 + b2 + b1 + b0 (mod 2)
0327                    *      (mask out other bits)
0328 11c7 ad 31         BSR    CALCHK      Calculate c4 (result returned in carry)
0329 11c9 39 94         ROL    CHECK0      rotate c4 into CHECK0
0330                    *
0331                    *
0332 11cb b6 92         LDA     DATA      Load data byte
0333 11cd a4 71         AND     #01110001  c3 = b6 + b5 + b4 + b0 (mod 2)
0334                    *      (mask out other bits)
0335 11cf ad 29         BSR    CALCHK      Calculate c3 (result returned in carry)
0336 11d1 39 94         ROL    CHECK0      rotate c3 into CHECK0
0337                    *
0338                    *
0339 11d3 b6 92         LDA     DATA      Load data byte

```

```

0340 11d5 a4 b6      AND      %%10110110      c2 = b7 + b5 + b4 + b2 + b1 (mod 2)
0341                *                               (mask out other bits)
0342 11d7 ad 21      BSR      CALCHK      Calculate c2 (result returned in carry)
0343 11d9 39 94      ROL      CHECK0      rotate c2 into CHECK0
0344
0345                *
0346 11db b6 92      LDA      DATA      Load data byte
0347 11dd a4 da      AND      %%11011010      c1 = b7 + b6 + b4 + b3 + b1 (mod 2)
0348                *                               (mask out other bits)
0349 11df ad 19      BSR      CALCHK      Calculate c1 (result returned in carry)
0350 11e1 39 94      ROL      CHECK0      rotate c1 into CHECK0
0351 11e3 81          RTS
0352                *
0353                *
0354                *   CALC5 - Calculates the parity checkbit C5
0355                *
0356                *   Enter with data in acc. and C1 to C4 in CHECK0 (0 to 3)
0357                *   Returns with C5 in CHECK0 (bit 4).
0358                *   Acc., X-reg. contents not saved.
0359                *
0360                *
0361                *
0362 11e4 b6 92      CALC5   LDA      DATA
0363 11e6 19 94      BCLR     4,CHECK0      Clear C5
0364 11e8 ad 10      BSR     CALCHK      Calc. C' = b7+b6+b5+b4+b3+b2+b1+b1+b0 (mod 2)
0365 11ea 24 02      BCC     NOSETC5
0366 11ec 18 94      BSET     4,CHECK0      Set C5 if C = 1
0367 11ee b6 94      NOSETC5 LDA     CHECK0
0368 11f0 ad 08      BSR     CALCHK      Calc. c5 = C'+c1+c2+c3+c4
0369 11f2 25 03      BCS     SETC5
0370 11f4 19 94      BCLR     4,CHECK0      Clear C5 if total no. of 1s in DATA and
0371                *                               c1 to c4 is even ((DATA,c1 to c4) parity
0372                *                               is even)
0373                *
0374 11f6 81          RTS
0375                *
0376 11f7 18 94      SETC5   BSET     4,CHECK0      Set C5 if (DATA,c1 to c4) parity is odd.
0377 11f9 81          RTS
0378                *
0379 11fa          CALCHK  EQU      *
0380                *   Calculates the modulo 2 sum of all bits
0381                *   in acc. (b7 + b6 + ... + b0) and returns
0382                *   with result in carry.
0382 11fa 3f 95      CLR     CHECK1
0383 11fc ae 08      LDX     #508
0384 11fe 46          CALCHK1 RORA
0385 11ff 24 02      BCC     CALCHK2
0386 1201 3c 95      INC     CHECK1
0387 1203 5a          CALCHK2 DECX
0388 1204 26 f8      BNE     CALCHK1
0389 1206 36 95      ROR     CHECK1
0390 1208 81          RTS
0391                *
0392                *
0393                *   PAKCHK - Calculates the address of the the 1st byte location
0394                *   for the check data C1 to C5 and then rotates the RAM registers
0395                *   CHECK0 and CHECK1 so that C1 to C5 are in the correct bit positions
0396                *   for programming into the Check EEPROM.
0397                *
0398                *   Enter with C1 to C5 precalculated and stored CHECK0 (0-4), Start
0399                *   address of DATA EEPROM in EPSTHI and EPSTLO and address offset
0400                *   for data byte in RAM reg. INDEX.
0401                *
0402                *   Returns with C1 to C5 rotated into correct position within CHECK0
0403                *   and CHECK1 (all other bit positions = 0) and address offset for
0404                *   C1 to C5 in X-reg.
0405                *
0406                *
0407                *

```

0408 1209 be 93	PAKCHK	LDX	INDEX	
0409 120b a6 05		LDA	#\$05	
0410 120d 42		MUL		Multiply DATA offset by 5
0411	*			
0412	*			
0413 120e 3f 98		CLR	REM	Initialise remainder RAM reg.
0414	*			
0415	*			
0416	*			Divide result in X-reg. and acc. by 8 and store remainder in RAM byte REM
0417	*			
0418 1210 56		RORX		
0419 1211 46		RORA		
0420 1212 24 02		BCC	CHKAD1	
0421 1214 10 98		BSET	0,REM	
0422 1216 56	CHKAD1	RORX		
0423 1217 46		RORA		
0424 1218 24 02		BCC	CHKAD2	
0425 121a 12 98		BSET	1,REM	
0426 121c 56	CHKAD2	RORX		
0427 121d 46		RORA		
0428 121e 24 02		BCC	CHKAD3	
0429 1220 14 98		BSET	2,REM	
0430	*			
0431	*			
0432	*			
0433 1222 3f 95	CHKAD3	CLR	CHECK1	
0434 1224 3d 98		TST	REM	
0435 1226 27 09		BEQ	NOREM	
0436 1228 98	CHKAD4	CLC		
0437 1229 39 94		ROL	CHECK0	Rotate CHECK0 and CHECK1 left
0438	*			- the no. of left shifts is equal
0439	*			to the value held in REM
0440 122b 39 95		ROL	CHECK1	- msb of CHECK0 is shifted into lsb
0441 122d 3a 98		DEC	REM	of CHECK1.
0442 122f 26 f7		BNE	CHKAD4	
0443	*			
0444 1231 97	NOREM	TAX		Store CHECK EEPROM offset for C1 to C5
0445	*			in X reg.
0446 1232 81		RTS		
0447				*****
0448	*			Subroutine EPRWRT/EPRERA - Writes one byte of data to EEPROM.
0449	*			or erases 4 byte block within which, specified address is located.
0450	*			The address to be written to is determined by EPRADH and EPRADL
0451	*			added to the contents of the index register.
0452	*			
0453	*			Subroutines used: EXSTAI
0454	*			
0455				*****
0456	*			
0457 1233 cd 12 4f	EPRWRT	JSR	EXSTAI	Load ext sta inst subroutine in RAM
0458 1236 1e 09		BSET	WE,PCR	Select write mode
0459 1238 20 05		BRA	EWRT1	
0460	*			
0461 123a cd 12 4f	EPRERA	JSR	EXSTAI	Load ext sta inst subroutine into RAM
0462 123d 1f 09		BCLR	WE,PCR	Select erase mode
0463 123f 11 09	EWRT1	BCLR	PLE,PCR	Latch address
0464 1241 bd 80		JSR	ADSTA	write to EEPROM
0465 1243 13 09		BCLR	PGE,PCR	Activate charge pump
0466 1245 cd 12 60		JSR	EPRDEL	Delay 10 ms at 2.5 MHz internal bus speed
0467 1248 10 09		BSET	PLE,PCR	
0468 124a be 91		LDX	SAVX	Restore X reg contents
0469 124c b6 90		LDA	SAVA	Restore acc. contents
0470 124e 81		RTS		
0471	*			
0472 124f b7 90	EXSTAI	STA	SAVA	Save acc
0473 1251 bf 91		STX	SAVX	Save X reg
0474 1253 a6 d7		LDA	#\$D7	Indexed 2 byte offset STA inst
0475 1255 b7 80		STA	ADSTA	

0476	1257	a6	81		LDA	#\$81	
0477	1259	b7	83		STA	ADSTA+3	
0478	125b	b6	90		LDA	SAVA	
0479	125d	be	91		LDX	SAVX	
0480	125f	81			RTS		
0481				*			
0482	1260	a6	d8	EPRDEL	LDA	#\$D8	10 ms delay at 2.5 MHz internal bus freq.
0483	1262	ae	10	EDEL1	LDX	#\$10	
0484	1264	cd	12 6b		JSR	EDEL2	
0485	1267	4a			DECA		
0486	1268	26	f8		BNE	EDEL1	
0487	126a	81			RTS		
0488				*			
0489	126b	5a		EDEL2	DECX		
0490	126c	26	fd		BNE	EDEL2	
0491	126e	81			RTS		
0492				*			
0493					END		
0494				*			

MC68HC805B6 and MC68HC705B5 Serial/Parallel Programming Module

By Ross A Mitchell and Mark Maiolani
Motorola Ltd, East Kilbride

INTRODUCTION

The MC68HC05B serial/parallel programmer module (Figure 1) enables the user to program MC68HC805B6 and MC68HC705B5 MCU devices. This application note describes the various operating modes of the module and gives details of its use and construction.

PROGRAMMING MODES

Two programming modes are available via jumper selection: parallel mode and serial mode.

In parallel programming mode, the user program contained in an external EPROM is copied into the internal EEPROM or EPROM of the MCU device, whereas in the serial programming mode the MCU EEPROM or EPROM can be programmed or read via the serial port on the programmer module.

Note: If the security bit is active on the 68HC705B5 device, no programming operations are possible. On the 68HC805B6 device, the device will be initially erased if programming is attempted with the security bit active.

Table 1 describes the 4 modes of operation for the 68HC805B6 and 68HC705B5 devices. The markings for jumpers J2 and J3 are on the programmer board.

PARALLEL PROGRAMMING MODE

This mode enables programming of the MCU EEPROM (68HC805B6) or EPROM (68HC705B5) directly from an external 27C64 EPROM device, with the programming module operating as a stand-alone unit. The main functions of erasure (68HC805B6 only), programming, and verification are all implemented in this mode.

In this mode, all the internal EEPROM of the 68HC805B6 is automatically erased before being programmed. Any failure of the EEPROM to erase results in illumination of the red LED and a re-attempt of erasure. EEPROM erasure is normally complete within 500 mS. The erased state of the 68HC805B6 is \$FF and \$00 for the 68HC705B5.

The 27C64 EPROM should contain the data to be programmed with the same addresses as the 6 Kbyte EEPROM of the 68HC805B6 and so the EPROM should only have data between addresses \$800 and \$1FFF inclusive. Note: The smaller 256 byte EEPROM array of the 68HC805B6 is not programmable from external EPROM.

When programming 68HC805B6 devices, locations of the external EPROM not in the internal EEPROM address range are omitted, as are locations containing the data \$FF, thus speeding up the programming operation. With 68HC705B5 devices, a similar technique is used, with the exception that areas with the data \$00 are omitted.

Jumper J2	Jumper J3	Device 68HC805B6	Device 68HC705B5
SERIAL	BOOT ONLY	SERIAL LOAD (NO ERASE)	RAM/EPROM SERIAL BOOT
SERIAL	ERASE + BOOT	SERIAL LOAD WITH ERASE	EPROM ERASE CHECK
PARALLEL	BOOT ONLY	PARALLEL RAM BOOT	PARALLEL RAM BOOT
PARALLEL	ERASE + BOOT	PARALLEL EEPROM BOOT	PARALLEL EPROM BOOT

Table 1. Programming modes for 68HC805B6 and 68HC705B5 devices

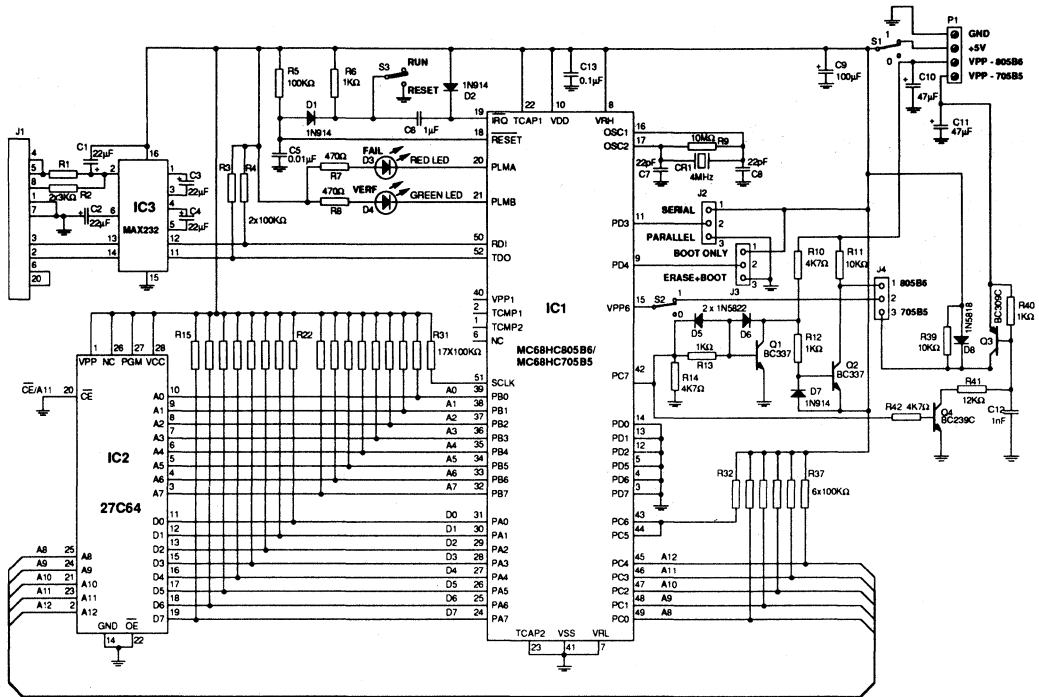


Figure 1. MC68HC805B6 and MC68HC705B5 Serial/Parallel Bootstrap Programmer

PARTS LIST

RESISTORS

- R1 3K
- R2 3K
- R3-R5 100K
- R6 1K
- R7,R8 470
- R9 10M
- R10 4K7
- R11 10K
- R12,R13 1K
- R14 4K7
- R15-R37 100K
- R39 10K
- R40 1K
- R41 12K
- R42 4K7

CAPACITORS

- C1-C4 22µF
- C5 0.01µF
- C6 1.0µF
- C7,C8 22pF
- C9 100µF
- C10,C11 47µF
- C12 1.0nF
- C13 0.1µF

DIODES

- D1,D2 1N914
- D3 LR3160
- D4 LG3160
- D5,D6 1N5822
- D7 1N914
- D8 1N5818

INTEGRATED CIRCUITS AND SOCKETS

- IC1 68HC805B6 52 Pin PLCC ZIF
- IC2 INT27C64 28 Pin DIL ZIF
- IC3 MAX232CPE 16 Pin DIL LIF

CONNECTORS

- J1 25 Way AMP Female
- J2,J3,J4 3 Way Jumper
- P1 4 Way Terminal Connector

SWITCHES

- S1,S2,S3 2 Way, Toggle Switch (SPDT)

MISC

- CR1 MK04000A 4MHz Crystal Package HC-18U

TRANSISTORS (All in T092 Package)

- Q1,Q2 BC337-25
- Q3 BC239C
- Q4 BC309C

During the programming operation the green LED should flash with a period of approximately 1 second to indicate normal programming mode.

After the programming operation has been completed, the programmed contents of the MCU are verified against the external EPROM. Any failure to verify will result in illumination of the red LED. Successful verification will result in illumination of the green LED.

The 68HC705B5 device can be checked for the EPROM in the erased state by placing the jumpers in the SERIAL and ERASE+BOOT positions with +5 volts on the Vpp supply for 68HC705B5. In this case follow the instructions below for parallel programming ignoring steps 4 and 5, but there is no need for a 27C64 EPROM in socket IC2. The green LED turned on indicates success, the red LED indicates that the EPROM is not in the erased state.

PARALLEL PROGRAMMING OPERATION

To program the MCU from an EPROM using parallel mode, perform the following steps:

1. With power to the module removed install MCU and EPROM devices into the programming module.
2. With the power switches S1 and S2 both off, and switch S3 in the RESET position, connect both the +5V supply and appropriate Vpp supply (68HC705B5 or 68HC805B6) to the module.
3. Set jumper J4 to the appropriate setting for the MCU being programmed (705B5 or 805B6).
4. Set jumper J3 to the 'ERASE + BOOT' position.
5. Set jumper J2 to the 'PARALLEL' position.
6. Turn the +5V power supply switch, S1, ON.
7. Turn the Vpp power supply switch, S2, ON.
8. Place switch S3 in the RUN position.
9. Once the green LED has stopped flashing, and remains continuously illuminated, place switch S3 to the RESET position.
10. Place the Vpp power supply switch, S2, in the OFF position.
11. Place the +5V power supply switch, S1, in the OFF position.

Note: To avoid possible damage to the MCU it is essential that power to the programming module is applied and removed in the sequence specified above.

SERIAL PROGRAMMING MODE

This mode allows the user to program and read the MCU EEPROM or EPROM via the serial port on the programmer module. By using a host computer and a control program such as E2B6, data can be downloaded and programmed onto the MCU, or uploaded from the MCU back to the host computer.

Programming in serial mode consists of the MCU reading a byte of data from the serial port, programming it into the internal 6 Kbyte EEPROM or EPROM array, reading the data back from programmed location and sending it to the serial port. The host computer should verify programming by checking the data returned from the programming module for differences from the programmed data, which would indicate incorrect programming or erasure.

As in parallel mode, bytes to be programmed with \$FF for EEPROM devices and \$00 for EPROM devices are skipped, reducing the overall programming time and allowing the memory upload feature to be implemented. This involves the host computer reading the data programmed in the MCU by attempting to program these values and examining the returned verification data.

MC68HC805B6

A program called E2B6 is available for the IBM PC and similar machines that communicate via RS-232 with the programmer board serial connector. This program allows upload (data transfer from 68HC805B6 to IBM PC) to read the EEPROM and can also program the EEPROM by downloading S1 record files to the 68HC805B6 device.

As in the parallel programming mode, the internal EEPROM areas can be automatically erased before programming. In serial mode, however, this operation is optional, and is selected by setting jumper J3 to the ERASE + BOOT position. An exception is if the EEPROM security bit is active, in which case the erase will be carried out regardless of the setting on J3. A 'read' or 'upload' of the EEPROM will also cause the EEPROM to be erased if J3 is set to 'ERASE + BOOT' or if the security bit is active.

MC68HC05B6

The 68HC05B6 (ROM device) 256 byte EEPROM may also be programmed using this board as described in

application note AN434. In this case the jumpers should be set as for the 68HC805B6, ERASE+BOOT and SERIAL but the Vpp supply for the 805B6 power socket should be connected to +5 volts.

MC68HC705B5

A program called EPB5 for the IBM PC communicates via RS-232 with the programmer board serial connector. This program allows upload (data transfer from 68HC705B5 to IBM PC) to read the EPROM and can also program the EPROM by downloading S1 record files to the 68HC705B5 device.

The 68HC705B5 EPROM means that the jumpers J2 and J3 have slightly different meaning. See table 1 for details of operating modes.

SERIAL PROGRAMMING OPERATION

1. Run the program E2B6 (68HC805B6) or EPB5 (68HC705B5) on an IBM PC to communicate with the device to be programmed.
2. With power to the module removed install the MCU and connect the serial line between the host computer and the serial port on the module.
3. With the power switches S1 and S2 both off and switch S3 in the RESET position, connect both the +5V supply and appropriate Vpp supply (705B5 or 805B6) to the module.

4. Set jumper J4 to the appropriate setting for the MCU being programmed (705B5 or 805B6).
5. Jumper J3 should be set to the desired setting, e.g., BOOT ONLY if reading data from the MCU or ERASE + BOOT if re-programming a device (68HC805B6 only).
6. Set jumper J2 to the 'SERIAL' position.
7. Turn the +5V power supply switch, S1, ON.
8. Turn the Vpp power supply switch, S2, ON.
9. Place switch S3 in the RUN position when prompted by the host computer control program.
10. Follow the instructions of the upload/download program to initiate the data transfer.
11. When the operation has been completed, place switch S3 to the RESET position.
12. Place the Vpp power supply switch, S2, in the OFF position.
13. Place the +5V power supply switch, S1, in the OFF position.

For programming several devices, leave the IBM PC program running and repeat instructions 2 to 13 inclusive.

Note: The documentation of the host computer control program being used should be consulted for further details on the use of serial programming mode.

MC68HC05E0 EPROM Emulator

By Peter Topping
MCU Applications Group
Motorola Ltd, East Kilbride

INTRODUCTION

The MC68HC05E0 is a versatile member of the M6805 family of microprocessors. Unlike most other versions it has no on-chip ROM but instead can address a full 64K of external memory. This memory could simply be a ROM or EPROM containing the required program but can also include RAM and/or additional hardware. In addition to the external busses required to support this capability, the MC68HC05E0 has the usual I/O, timers etc. found on single-chip microprocessors.

The EPROM emulator described here illustrates a typical application of this type of microprocessor. In addition to the program EPROM it employs a keyboard, LCD, serial communication and 64K of paged RAM. The emulator can replace with RAM the program EPROM or ROM (up to 64K x 8) in a microprocessor based target system. This is done by connecting the emulator to the target system via a cable to its EPROM socket.

The object code, which can be loaded serially or from an EPROM, can be inspected and modified with the use of a local keyboard and LCD display. The new or modified code can then be used by the target system without having to go through the procedure of erasing and re-programming an EPROM after each software change. A selectable offset in \$0100 steps is available in order to position the code correctly in the target system's memory map.

The emulator facilitates the debugging of hardware and software for any system whose control program is to be contained in a 27(C)16/32/64/128/256/512 type EPROM. The control software includes branch offset calculation for 6805 code and is thus particularly suitable for debugging systems using one of the microprocessors from the M68(HC)05/01/11 ranges.

Two basic methods of loading a program are available. The first is applicable when the code is available in an existing EPROM. The contents of this EPROM can be transferred by the microprocessor into the RAM. This method requires an existing EPROM but will prove useful in applications where a small change to an existing program has to be checked before committing to an updated EPROM. This can be done without access to the source or object code. An EPROM can be read from the target system interface (through the emulator's buffers) or from a separate socket wired directly to the microprocessor. The former method allows one socket to be used for both EPROM reading and the target cable. The second method saves having to remove the target cable to read an EPROM but requires an additional socket.

Alternatively, data can be serially loaded in the form of Motorola S-records via an RS232 link. This code can come from a "COM" port of a PC (using the COPY command) or by tapping into the link between a computer and its terminal on a system using an RS232 connection between terminal and host. In this case a TYPE or LIST to the terminal should be used. A verify facility which compares the contents of RAM with serial S-records is also available, as is a routine to dump the current contents of the emulation RAM out on the RS232 interface.

PRINCIPLE OF OPERATION

Figure 1 shows a block diagram of the emulator in each of its three main modes of operation. The data/address flow is controlled by MC74HC245 bidirectional tri-statable 8-bit buffers. They constitute two 18-bit buffers for address and control signals and two 8-bit buffers for data. The enabling and direction control signals are supplied by the MC68HC05E0 microprocessor.

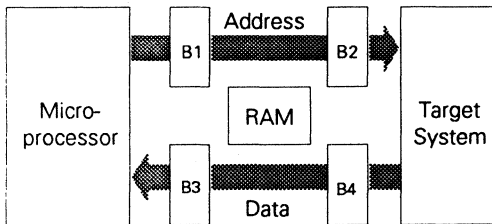


Figure 1a. Mode A:
direct access to the target system's interface

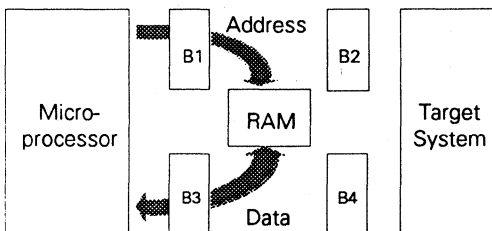


Figure 1b. Mode B:
access to the emulator's RAM

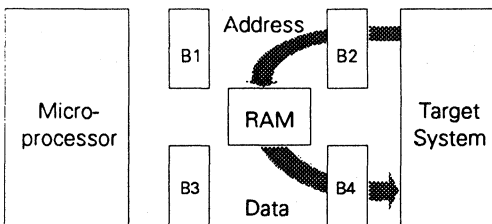


Figure 1c. Mode C:
gives the target system access to the RAM

There are three modes of operation:

- Mode A allows the microprocessor to read the contents of an EPROM on the target system interface (this is most easily arranged by connecting the cable to the target system via a zero or low insertion force socket) by enabling buffers B1 and B2 to drive from left to right to supply addresses to the socket (the RAM also receives these addresses but its data outputs are disabled). Buffers B3 and B4 are enabled from right to left to return the data from the EPROM to the MC68HC05E0. The RAM is disabled via its chip-enable pin and so does not affect the data bus between buffers B3 and B4.
- Mode B enables the buffers in such a way that the microprocessor can read from and write to the RAM. B1 is enabled to supply addresses to the RAM from the microprocessor (MC74HC245s were used throughout although a bidirectional buffer is not strictly necessary in this position as B1, if enabled, always drives from left to right). B3 is enabled to allow data to be written to or read from the RAM. The direction control for B3 is by the R/W signal from the MC68HC05E0 (gated with the RAM's chip enable). This mode is used during use of the memory modify facilities. Buffer B4 is disabled so that there is not a bus contention on either of its busses even if the target system is still connected. Buffer B2 is also disabled. The routine (L1) which loads RAM from an EPROM on the target system interface switches between modes A and B for each byte transferred.
- In the emulation mode (C) the target system plugged into the socket is required to have access to the RAM so buffers B2 and B4 are enabled. B2 passes the addresses from right to left and B4 the data from left to right (as the emulation is for an EPROM, the target system is not allowed to write to the RAM). Buffers B1 and B3 are disabled.

Control line	MODE		
	A	B	C
4,PortB	1	1	0
5,PortB	0	0	1
6,PortB	0	1	0
7,PortB	1	0	0

CIRCUIT

Figure 2 shows the main circuit. An MC74HC138 is used to provide the chip enables. The emulator hardware is enabled in the address range \$4000 – \$7FFF and the EM64K program EPROM (27(C)64) at \$C000 – \$FFFF. If the EM64K program is contained in a 27(C)16 its pin 21 (Vpp) should be held high.

An additional socket is shown at address \$8000 – \$BFFF. This is for the optional LOAD2 facility which allows code to be loaded from EPROM without having to disconnect the cable to the target system. The emulation RAM occupies the address range \$4000 – \$7FFF. As this is only a 16K address space the RAM is paged. The four pages are selected by I/O lines (port B, bits 0 and 1) from the microprocessor. The memory map of the emulator is shown in figure 7.

The control lines (port B, bits 4–7) are biased by resistors. This holds the system in mode B if the MC68HC05E0 is held in reset and prevents bus contention resulting from an illegal combination of control signals. During hardware debug of the emulator it is advisable to use a current limited power supply (in the range 50–100 mA) as a bus contention can cause sufficiently high currents to damage the buffers.

The display is a 6-digit 4-backplane LCD (eg Hamlin type 4200 or the 8-digit GE type LXD69D3F09KG) which is driven by an MC145000 display driver. The driver is controlled by a 2-line serial link from the microprocessor. A single-backplane (or "static") display can be used as an alternative as shown in Figure 3. Three MC144115 driver chips are used. This circuit requires many more connections to the LCD but allows the use of a more readily available display. A third line (port B bit 3) from the microprocessor is used to supply the enable pins of the MC144115s. The single-backplane display drivers can be supplied directly from the main 5 volt supply but the multiplexed display requires a lower voltage. Figure 2 shows the MC145000 supplied via a 20k potentiometer which serves as a contrast control.

The keyboard uses an MC14028 decoder to minimise the number of I/O pins used. Note that port A bit 6 is used for both the keyboard and the display driver. The LOAD1 and LOAD2 keys overwrite the contents of emulation RAM and should thus not be pressed accidentally. It may therefore be useful for only those LOAD keys actually required to be fitted (usually LOAD1 and LOAD2 will not both be required) and for

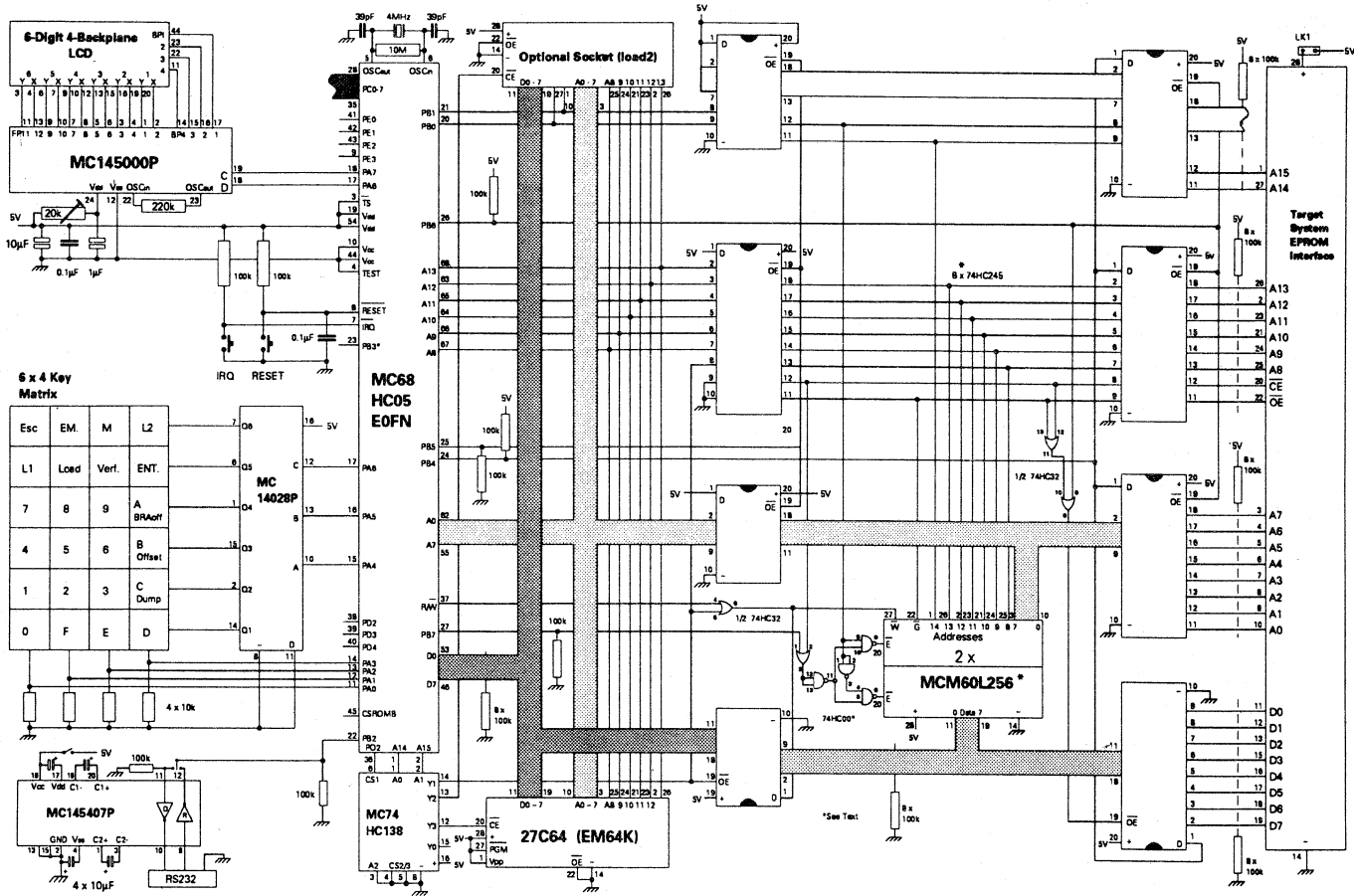
any parallel LOAD keys fitted to be placed away from the front panel or protected by requiring two keys, connected in series, to be pressed. An accidental press of the serial LOAD key can be aborted by pressing RESET.

As the circuit, except for the RS232 interface, is all CMOS the supply current is very low when the microprocessor is in STOP mode. This is a low power mode in which all processing, even the clock, is stopped. In the emulation mode (C) the MC68HC05E0 is in stop mode. In this mode and with no bus activity from the target system (or its interface open circuit) the supply current should be less than 1 μA (this does not include the current taken by the RS232 interface which, if present, can be switched off when not in use, or the 70–80 μA taken by the LCD driver).

It is worth checking that a low supply current is achieved as any excess can be a useful pointer to a wiring fault, particularly open circuit pins. The supply current may be affected by the choice of RAM but the MCM60L256 selected has a specified standby ICC of 2 μA and is typically well below this figure. In many applications the full 64K of RAM will not be required. If this is the case, only the required RAM need be included. 6116 2K RAMs could be used for 2–4K applications and MCM60L64s or equivalents for 8–16K. If using 6064s, their second chip enable pin (E2) should be held high. One MCM60L256 provides 32K. The serial load routine includes a read-back check on each byte sent to RAM so an attempt to write to non-existent RAM will generate an error message indicating the first faulty address. If 16K or less is required then the two 74HC245s handling addresses, A14 and A15, can also be omitted. These buffers have unused pins. The simplest way to ensure that no pins are left open circuit is to wire up the buffers in a manner similar to those actually used. Pins 2–7 of the left-hand buffers are held high while pins 13–18 are connected to the right-hand buffers whose other pins have pull-ups. This arrangement means that there will be no open circuits or bus contentions regardless of the levels of the control lines. If only one memory chip is used, the 74HC00 can be omitted (connect pin 3 of the 74HC32 directly to the RAM's chip enable).

The optional RS232 interface can most easily be implemented using the single-supply MC145407 driver-receiver chip. If outputting of S-records is not required then a simple transistor inverter with a pull-up resistor and a reverse polarity protection diode can be used. This interface is shown in Figure 4.

Figure 2. EPROM emulator circuit diagram



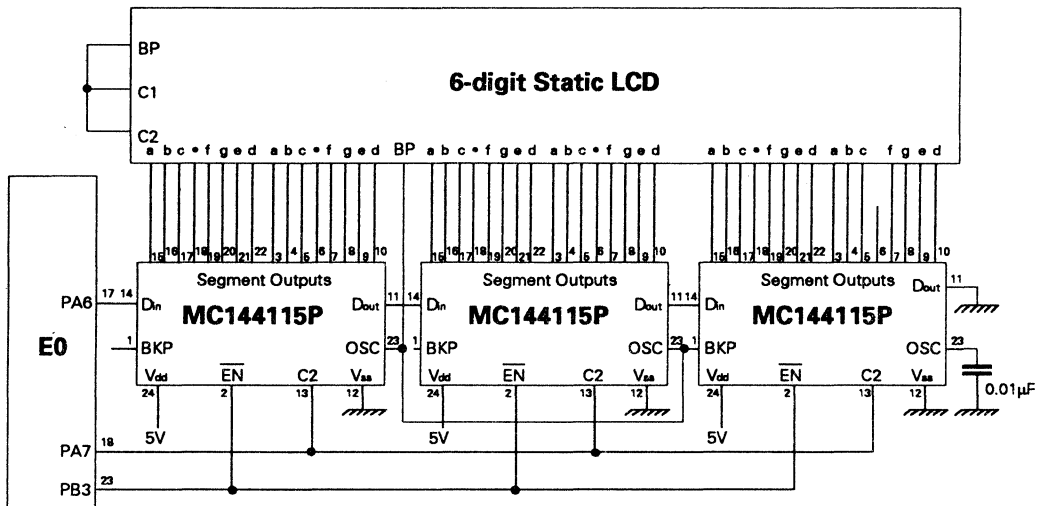


Figure 3. Alternative static LCD display

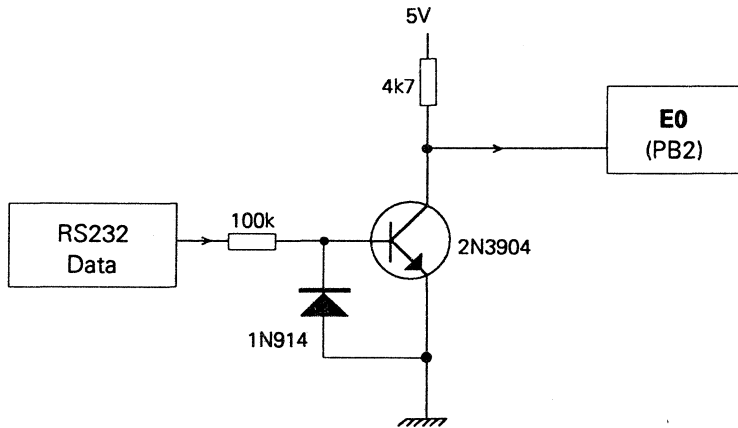


Figure 4. Simple input-only RS232 interface

IDD Monitor

It is often useful with CMOS circuits to provide a simple I_{DD} monitor which shows via an LED whether or not the I_{DD} is above or below a set value. In this application it shows whether or not the microprocessor is in the STOP mode. The required circuit is shown in Figure 5. The current threshold can be chosen by selecting the value of R1. A value of $1k\Omega$ sets the limit at about $500\mu A$ which means the LED should be off in the emulation mode but on otherwise. The $500\mu A$ limit allows the LCD and perhaps an emulator-supplied CMOS target system to be supplied without switching on the LED. When the microprocessor is not in STOP (emulator not in mode C), its I_{DD} is several milliamps and the LED should be lit. In a battery application this circuit would also serve as a useful reminder that the RS232 interface has been left on. If a multiplexed LCD is used it may be preferable not to supply it via this type of monitor circuit as a significant change in contrast may occur when the microprocessor goes into its STOP mode (see Figure 5). The monitor drops about $600mV$ when the microprocessor is running so the supply voltage should be chosen accordingly; four zinc-carbon or five Ni-Cad cells were found to be satisfactory.

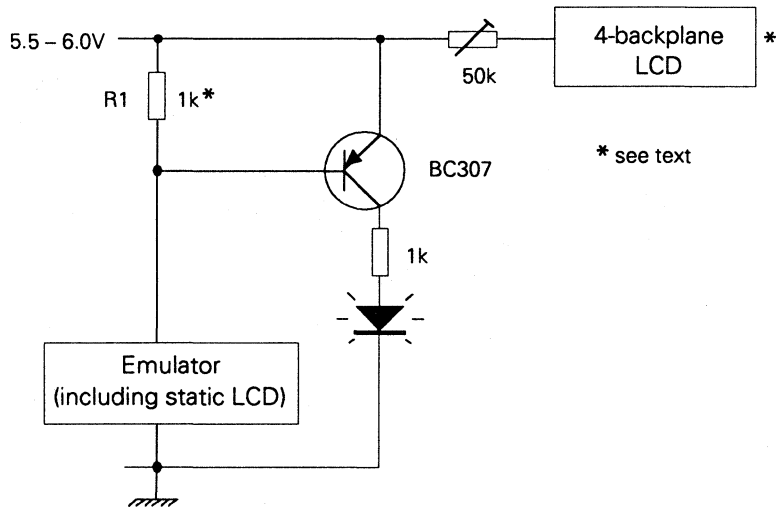


Figure 5. Simple I_{DD} monitor

Address trap

The emulator allows memory locations to be examined and changed, but does not provide the breakpoint and trace features normally found in development systems. A limited capability can be made available if address comparators of the type shown in Figure 6 are added. This circuit gives an LED indication if the address selected on the bank of switches is encountered by the program running in the target system. An address coincidence is latched by the 74HC74. To indicate the occurrence of a repetitive event, a one-shot chip could be added.

SERIAL LOAD

To load external Motorola S-records the serial load key (LOAD) should be pressed. The LCD will display "LOAD". S-records should then be supplied at 9600 baud (8-bit, no parity) on the RS232 interface. When an S9 termination record is received, the prompt returns. If an error is detected during a serial load, the load routine stops and displays the address at which the error occurred and the error type.

The following error types are possible:

- 1: Checksum error, transmitted data or interface faulty.
- 2: RAM read-back error, RAM faulty or non-existent.
- 3: ASCII character less than \$30 (0) received.
- 4: ASCII character between \$39 (9) and \$41 (A) received.
- 5: ASCII character more than \$46 (F) received.
- 7: Verify error when comparing S-records with emulation RAM.

If, when using the emulator, the target system ceases to function properly, then the verify function can be used to check that the emulation RAM has not been corrupted. The VERIFY function is used exactly like LOAD except that RAM is compared with, rather than loaded by, the S-records.

The address at the start of each S1-record determines the address at which the code will reside in the target system. This address will sometimes be different from that at which the code is required to be loaded into the emulation RAM so an offset may need to be used. The offset byte is entered using the appropriate key and allows an offset of any multiple of \$0100. The offset is subtracted from the MSB of the S-record address and this modified address is the physical address at which the data is loaded into the emulation RAM. The S-record output routine adds the offset before transmitting the records. At reset or power-up the offset is initialised to zero.

All addresses entered while using the MEMORY-MODIFY, BRAOFF and DUMP routine use the actual address in the target system. These addresses will only be the same as the physical RAM address if the offset is zero.

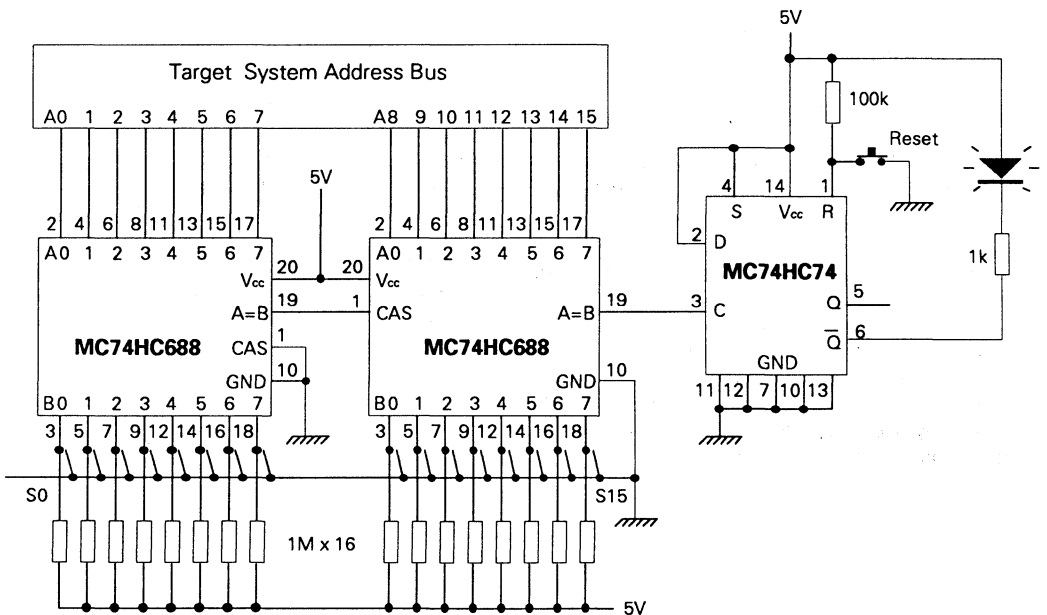


Figure 6. Address Trap

PARALLEL LOAD

When the parallel load functions (LOAD1 and LOAD2) are used, OFFSET has no effect on the transfer of code into RAM. It can, however, still be used to offset the RAM addresses to correspond with the actual address in the target system program when using the memory-modify facilities.

27(C)64/128/256 EMULATION

When emulating a 27(C)512 EPROM, all the RAM is used. For emulation of smaller EPROMS, less RAM is required. The memory used will be at the beginning of RAM (starting at address zero) only if the unused high-order address lines are held low. It may, however, be more convenient to allow one or more of these addresses to be high. The pull-ups included in Figure 2 will hold any uncommitted lines high. For example, a 27(C)64 can be emulated with no hardware change as long as the code is loaded between \$E000 and \$FFFF. This will often be appropriate as it allows the vectors at the top of the target system's memory to be included. It makes little difference if the target microprocessor has an address space smaller than 64K as the high order addresses will not be present and will be held high. Clearly, the code must still be assembled at the appropriate addresses and the emulator's offset feature used to load the S-records between \$E000 and \$FFFF. Alternatively, the S-records can be loaded lower in the emulator's address space and the relevant high-order addresses held low. When loading from a smaller EPROM, with a VPP or PRG pin, these pins should be configured correctly for reading (high) and not driven by the emulator. See Figure 9 for the industry-standard EPROM pinouts. As the software does not behave differently for smaller EPROMs, a full 64K transfer will still be made, copying the EPROM several times into the 64K RAM. The actual copy used depends on the levels of the high order addresses as outlined above.

TARGET SYSTEM INTERFACE

Vdd can be connected to the interface by the link shown. The simplest method of use is to make this connection and to use a common supply for the emulator and the target system. If, however, separate supplies are used, then pin 28 should not be connected. If separate supplies are used, care should be taken that they do not differ by more than 0.5V. A delta greater than this may cause a malfunction as a result of the logic level on an input pin being in excess of the chip's Vdd.

In emulation mode the target system has total control of the RAM except for its R/W line. It can thus use the RAM exactly as if it were a ROM or EPROM. Before IRQ (or RESET) is pressed to exit from the emulation mode the target system should be stopped so that it no longer expects the "EPROM" to be there. This will normally be done by holding the target system in reset. If the target system is an M68(HC)05 (eg MC68HC05E0 or MC146805E2) or M68HC11, then it can alternatively be put into its STOP mode. If this is its normal idle condition, then nothing need be done prior to exiting emulation.

EM64K PROGRAM

The EM64K control program is less than 2K bytes long and can thus reside in a 27C64 or 27C16. The circuit is shown for a 27C64 and assumes that the program starts at the beginning of the EPROM. This EPROM is enabled at \$C000 (and \$E000 as A13 is not used). An assembled listing of the control program is included at the end of this application note.

EM64K KEY FUNCTIONS

Function	KEY	Description of function
LOAD1	L1	Load RAM from target system interface (\$4000-\$7FFF).
LOAD2	L2	Load RAM from secondary socket (\$8000-\$BFFF).
SERIAL LOAD	LOAD	Load emulation RAM with S-records via the RS232 interface, during loading LCD shows "LOAD".
VERIFY	Verf	Compare emulation RAM with S-records via the RS232 interface, LCD displays "UErIFy".
EMULATE	EM	Emulator mode. Prompts: "EP ?" for the removal of an EPROM (if present) and connection of the target system (press again if OK) and put micro into EMULATE mode.
MEMORY MODIFY	M	Display/change a RAM location. When pressed the last address is displayed. Press ENTER to display the contents of this address or input a new address followed by ENTER. To change, input new data followed by ENTER. ENTER moves to next address, M moves to previous address, ESCAPE exits.
ENTER	Ent	Enter keyed-in address or data (and move to next address in MEMORY MODIFY).
ESCAPE	Esc	Exit from current function (OFFSET, BRAOFF, DUMP or MEMORY MODIFY).
BRAOFF	A	Calculate branch offset. The address of the branch instruction and of the destination are requested. If a valid branch is calculated it is written into memory and displayed. If not valid then "or" for out of range is displayed. A branch of -128 through +127 relative to the start address of the next instruction is allowed. Esc returns to the normal prompt.
OFFSET	B	Allows entry of an offset to the emulation RAM address. It is subtracted from the most significant byte of the address specified by the incoming S-records. The offset is added to the address by the DUMP function.
DUMP	C	Output emulation RAM contents as S-records via RS232 interface. RAM start and finish addresses are requested. They should be entered followed by ENTER. After the second ENTER, the S-record output starts.
IRQ	IRQ	Abort emulation and return to emulator monitor.
RESET	RESET	Resets emulator, displays prompt (□). Should be used after power-up or if the emulator malfunctions. Can be used instead of IRQ, with the difference that the OFFSET is reset to zero. RESET provides the only exit from a LOAD or VERIFY which has not been terminated correctly by the reception of an S9 record.

MC68HC05E0 I/O timers	0000 001F
MC68HC05E0 RAM (480 bytes including stack at 00FF)	0020 01FF
Not used.	0200 3FFF
Emulator RAM (64k in 4 x 16k Pages)	4000 7FFF
Load2 EPROM Socket	8000 BFFF
EM64k Control Program	C000 FFF5
MC68HC05E0 Vectors	FFF6 FFFF

Figure 7. Memory Map

The only other register used (apart from the I/O data and DDR registers) is the interrupt control register (\$0E). It is written to \$01 on lines 82 and 83 of the listing. This operation clears the interrupt flag (bit 3) but keeps the INTMX bit set. This bit enables external interrupts. The registers associated with unused on-chip resources are left at their reset conditions. An important bit in the MC68HC05E0 is the XROM bit (2,\$0C). It defaults to a 1 which is appropriate in this application. When it is cleared it constrains the data bus to be input only thus preventing any unnecessary activity in sensitive applications when writing to external memory is not required.

The MC68HC05E0 has the 8-bit index register common to all M6805 microprocessors. It is thus not able to contain a 16-bit extended address. For this reason, loading and storing in the emulator's RAM is carried out using a small program in the micro's RAM. This program consists of an extended LDA or STA instruction followed by a two byte address which can be built in software and an RTS instruction. The four-byte program resides in RAM at locations W2, ADDEH, ADDRL and W3. It allows the full 64K map to be accessed using addresses generated within the program. Address generation is further complicated by the requirement that the emulation RAM is in four 16K pages. The two most significant addresses thus have to be transferred to port lines PB0 and PB1.

SOFTWARE

A listing of the control program used in the emulator is included in this application note. Some points specific to the MC68HC05E0 are discussed below.

Port D on the MC68HC05E0 can be used as a normal I/O port or can selectively supply special signals. In this application five of the special function are used. These function are selected using the register at address \$12. The function used are P02, R/W, A13, A14 and A15. By default only addresses A0 through A12 are available as this will be sufficient in many applications. In this application, however, all the addresses are required. The clock (P02) is used to qualify the chip selects generated by the MC74HC138 and R/W for control of the emulation RAM. The other three pins are left as I/O pins but are not used in this application. The initialisation of \$12 can be seen on lines 93 and 94 of the software listing.

SERIAL INTERFACE

Figure 8 shows a suggested method of wiring up the RS232 sockets in an emulator with both loading and dumping capabilities. This arrangement facilitates use of the serial LOAD and DUMP routines of the emulator either via a PC COM port or between a host and terminal connected by an RS232 link. When using a PC the "host" socket should be used. As only one pin on the MC68HC05E0 is used, switching is required to make the required connections. S2 can be eliminated (or left at "L") if only loading is required, as will often be the case. To save power in battery applications, the RS232 interface chip can be switched off using S1. The following table shows possible methods of use.

Set-up	Function	S1	S2	Comments
Host & terminal	Load	On	L	Terminal and host connected. Micro looks at data sent from host to terminal (pins 3).
	Dump	On	D	Connection between terminal and host broken. S-records sent to both host (2) and terminal (3).
PC "COM" port	Load	On	L	S-records loaded from pin 3.
	Dump	On	D	S-records sent to pin 2 on "host" socket (and pin 3 on "terminal" socket).

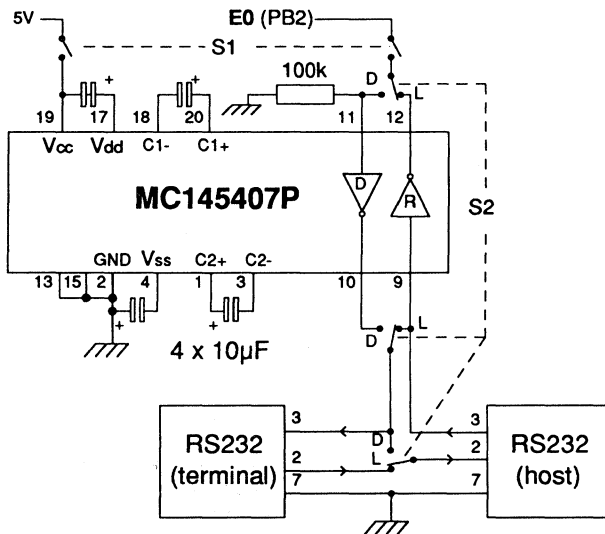


Figure 8. RS232 circuit with LOAD/DUMP switching

Pin	27512	27256	27128	2764
1	A15	Vpp	Vpp	Vpp
2	A12	.	.	.
3	A7	.	.	.
4	A6	.	.	.
5	A5	.	.	.
6	A4	.	.	.
7	A3	.	.	.
8	A2	.	.	.
9	A1	.	.	.
10	A0	.	.	.
11	D0	.	.	.
12	D1	.	.	.
13	D2	.	.	.
14	Vss	.	.	.
15	D3	.	.	.
16	D4	.	.	.
17	D5	.	.	.
18	D6	.	.	.
19	D7	.	.	.
20	Chip enable	.	.	.
21	A10	.	.	.
22	Output enable	.	.	.
23	A11	.	.	.
24	A9	.	.	.
25	A8	.	.	.
26	A13	A13	A13	NC
27	A14	A14	PGM	PGM
28	Vcc	.	.	.

**Figure 9. 27(C) 512, 256, 128 and 64 pin-outs
 (Table shows the standard 28-pin EPROMs.
 Blank entries indicate that the pin is the same as for the 27(C) 512.)**

ASSEMBLED LISTING OF THE EM64K CONTROL PROGRAM

```

1          *****
2          *
3          *          MC68HC05E0 EPROM Emulator.          *
4          *
5          *          A 68HC05E0 is used to emulate an EPROM          *
6          *          of up to 64K (27(C)512) with SRAM which          *
7          *          can be loaded from an EPROM or serially          *
8          *          by S-records via an RS232 interface and          *
9          *          changed if required for de-bug etc.          *
10         *
11         *          P. Topping          11-Jan-91          *
12         *
13         *****
14
15 00000000          PORTA EQU          $00          PORT A ADDRESS
16 00000001          PORTB EQU          $01          " B "
17 00000002          PORTC EQU          $02          " C "
18 00000003          PORTD EQU          $03          " D "
19 00000004          PORTE EQU          $04          " E "
20 00000005          PORTAD EQU          $05          PORT A DATA DIRECTION REG.
21 00000006          PORTBD EQU          $06          " B " " "
22 00000007          PORTCD EQU          $07          " C " " "
23 00000008          PORTDD EQU          $08          " D " " "
24 00000009          PORTED EQU          $09          " E " " "
25 0000000e          ICR EQU          $0E          INTERRUPT CONTROL REGISTER
26 00000012          PORTDSF EQU          $12          PORTD ALTERNATIVE FUNCTION REGISTER
27
28          ORG          $0020          RAM ALLOCATION
29
30 00000020          DTABL RMB          6          LCD BUFFER
31 00000026          TEMP RMB          2
32 00000028          W1 RMB          1
33 00000029          W2 RMB          1          RAM SUBROUTINE LDA or STA
34 0000002a          ADDEH RMB          1          " " ADDRESS MSB
35 0000002b          ADDR1 RMB          1          " " " LSB
36 0000002c          W3 RMB          1          " " RTS
37 0000002d          W4 RMB          1
38 0000002e          W5 RMB          1
39 0000002f          W6 RMB          1
40 00000030          ADDRH RMB          1
41 00000031          STAT RMB          1          STATUS BYTE :-
42         *          2: REAL ADDRESS (OFFSET)
43         *          4: INVALID ADDRESS (BLDRNG)
44         *          5: VERIFYING (TLOAD)
45         *          6: INDIVIDUAL REG. (MEMEX)
46         *          7: PUNCH END
47 00000032          CHKSUM RMB          1          CHECKSUM
48 00000033          COUNT RMB          1          BIT COUNTER
49 00000034          TMP1 RMB          1
50 00000035          TMP2 RMB          1
51 00000036          BCNT RMB          1          S-RECORD BYTE COUNT
52 00000037          ERTYP RMB          1          ERROR TYPE
53 00000038          ERDAT RMB          1          ERROR DATA
54 00000039          OFF RMB          1          S-RECORD OFFSET
55 0000003a          RMB          185          UNUSED
56 000000f3          STACK RMB          12          13 BYTES USED (1 INTERRUPT
57 000000ff          SP RMB          1          AND 4 NESTED SUBROUTINES)

```

em64k.as5

```

59
60
61
62
63
64
65
66
67
68 0000e000 cde05f SCAN JSR KEYSCHN KEY FOUND ?
69 0000e003 24fb BCC SCAN NO, TRY AGAIN
70 0000e005 5f CLRX
71 0000e006 b728 STA W1 CODE OF PRESSED KEY
72 0000e008 d6e09f RJ LDA CTAB.X FETCH KEYCODE
73 0000e00b b128 CMP W1 THIS ONE ?
74 0000e00d 270b BEQ PJ YES
75 0000e00f c1e0bf CMP LAST NO, LAST CHANCE ?
76 0000e012 273a BEQ GETCMD YES, ABORT
77 0000e014 5c INCX NO
78 0000e015 5c INCX TRY
79 0000e016 5c INCX THE
80 0000e017 5c INCX NEXT
81 0000e018 20ee BRA RJ KEY
82 0000e01a a601 PJ LDA #1
83 0000e01c b70e STA ICR CLEAR IRQX FLAG
84 0000e01e 5c INCX
85 0000e01f dce09f JMP CTAB.X

```

em64k.as5

```

87
88
89
90
91
92
93 0000e022 a6e3 START LDA #$E3 ENABLE PORTD SPECIAL FUNCTIONS
94 0000e024 b712 STA PORTDSF P02, R/W, A13, A14 & A15
95
96 0000e026 3f00 CLR PORTA
97 0000e028 a6f0 LDA #$F0 DISPLAY/KEYBOARD
98 0000e02a b705 STA PORTAD I/O
99
100 0000e02c a658 LDA #$58 MODE 2, ENABLE (144115) HIGH
101 0000e02e b701 STA PORTB
102 0000e030 a6fb LDA #$FB BITS 0, 1, 3-7 OUTPUTS
103 0000e032 b706 STA PORTBD BIT 2 INPUT
104
105 0000e034 3f02 CLR PORTC
106 0000e036 a6ff LDA #$FF ALL OUT, NOT USED
107 0000e038 b707 STA PORTCD
108
109 0000e03a 3f03 CLR PORTD
110 0000e03c a61c LDA #$1C BITS 2, 3 & 4 OUT, NOT USED
111 0000e03e b708 STA PORTDD
112
113 0000e040 3f04 CLR PORTE
114 0000e042 a60f LDA #$0F BITS 0 - 3 OUT, NOT USED
115 0000e044 b709 STA PORTED
116
117 0000e046 3f31 CLR STAT
118 0000e048 3f2b CLR ADDRLL INITIALISE
119 0000e04a 3f30 CLR ADDRHH ADDRESS
120 0000e04c 3f39 CLR OFF
121
122 0000e04e a658 GETCMD LDA #$58 MODE 2, ENABLE (144115) HIGH

```

123	0000e050	b701	STA	PORTB	
124	0000e052	cde235	JSR	CLRTAB	
125	0000e055	a633	LDA	#\$33	PRINT
126	0000e057	b720	STA	DTABL	PROMPT
127	0000e059	cde1ec	DSCN JSR	DISTAB	
128	0000e05c	9c	RSP		
129	0000e05d	20a1	BRA	SCAN	

em64k.as5

131			*****		
132			*		*
133			*	The keyboard routine	returns the code
134			*	of the pressed key	in the accumulator.
135			*		*
136			*****		
137					
138	0000e05f	4f	KEYSCN CLRA		
139	0000e060	ae06	LDX	#6	SETUP
140	0000e062	ab10	KEY1 ADD	#\$10	ROW
141	0000e064	b700	STA	PORTA	
142	0000e066	b600	COLUMN LDA	PORTA	READ KEYBOARD
143	0000e068	b72c	STA	W3	STORE IT
144	0000e06a	a50f	BIT	#\$0F	KEY CLOSED ?
145	0000e06c	2719	BEQ	COLRET	NO GET OUT
146	0000e06e	ad1d	BSR	DBOUNC	ELSE DEBOUNCE
147	0000e070	b600	LDA	PORTA	RE-READ KEYPAD
148	0000e072	b12c	CMP	W3	SAME KEY CLOSED ?
149	0000e074	2611	BNE	COLRET	NO, GET OUT
150	0000e076	99	SEC		
151	0000e077	b600	COL1 LDA	PORTA	KEY
152	0000e079	a50f	BIT	#\$0F	RELEASED ?
153	0000e07b	26fa	BNE	COL1	NO TRY AGAIN
154	0000e07d	ad0e	BSR	DBOUNC	YES DEBOUNCE
155	0000e07f	b600	LDA	PORTA	STILL
156	0000e081	a50f	BIT	#\$0F	RELEASED ?
157	0000e083	26f2	BNE	COL1	NO TRY AGAIN
158	0000e085	b62c	LDA	W3	RETURN CHAR IN A-REG
159	0000e087	2503	COLRET BCS	KEY2	IF VALID GET OUT
160	0000e089	5a	DECX		ELSE TRY
161	0000e08a	26d6	BNE	KEY1	NEXT ROW
162	0000e08c	81	KEY2 RTS		
163					
164	0000e08d	a60a	DBOUNC LDA	#10	40mS
165	0000e08f	b72f	STA	W6	
166	0000e091	a6ff	DLP LDA	#\$FF	PAUSE
167	0000e093	21fe	DLOOP BRN	*	256X12
168	0000e095	21fe	BRN	*	CYCLES
169	0000e097	4a	DECA		
170	0000e098	26f9	BNE	DLOOP	
171	0000e09a	3a2f	DEC	W6	
172	0000e09c	26f3	BNE	DLP	
173	0000e09e	81	RTS		

em64k.as5

175
176
177
178
179
180

```
*****
*                                     *
*      Keyboard tables.               *
*                                     *
*****
```

181	0000e09f	51	CTAB	FCB	\$51	P	LOAD FROM EPROM (\$4000)
182	0000e0a0	cce27b		JMP	DUMP1		
183	0000e0a3	68		FCB	\$68	S	LOAD FROM EPROM (\$8000)
184	0000e0a4	cce2db		JMP	DUMP9		
185	0000e0a7	28		FCB	\$28	C	S-RECORD OUTPUT
186	0000e0a8	cce3f8		JMP	PUNCH		
187	0000e0ab	52		FCB	\$52	L	LOAD S-RECORD
188	0000e0ac	cce321		JMP	TLOAD		
189	0000e0af	54		FCB	\$54	V	VERIFY (S-RECORD)
190	0000e0b0	cce31b		JMP	VERIFY		
191	0000e0b3	62		FCB	\$62	G	GO INTO EMULATOR MODE
192	0000e0b4	cce23d		JMP	MODE3		
193	0000e0b7	64		FCB	\$64	M	READ/CHANGE MEMORY
194	0000e0b8	cce4fd		JMP	MEMEX		
195	0000e0bb	38		FCB	\$38	B	ADDRESS OFFSET
196	0000e0bc	cce2c0		JMP	OFFSET		
197	0000e0bf	48	LAST	FCB	\$48	A	BRANCH OFFSET CALC.
198	0000e0c0	cce13e		JMP	BRAOFF		
199							
200	0000e0c3	11	STABL	FCB	\$11	0	
201	0000e0c4	21		FCB	\$21	1	
202	0000e0c5	22		FCB	\$22	2	
203	0000e0c6	24		FCB	\$24	3	
204	0000e0c7	31		FCB	\$31	4	
205	0000e0c8	32		FCB	\$32	5	
206	0000e0c9	34		FCB	\$34	6	
207	0000e0ca	41		FCB	\$41	7	
208	0000e0cb	42		FCB	\$42	8	
209	0000e0cc	44		FCB	\$44	9	
210	0000e0cd	48		FCB	\$48	A	BRANCH OFFSET
211	0000e0ce	38		FCB	\$38	B	LOAD OFFSET
212	0000e0cf	28		FCB	\$28	C	OUTPUT S-RECORDS
213	0000e0d0	18		FCB	\$18	D	
214	0000e0d1	14		FCB	\$14	E	
215	0000e0d2	12		FCB	\$12	F	
216	0000e0d3	61		FCB	\$61	10	Esc CANCEL COMMAND
217	0000e0d4	58		FCB	\$58	11	E ENTER COMMAND
218	0000e0d5	68		FCB	\$68	12	S LOAD FROM \$8000
219	0000e0d6	64		FCB	\$64	13	M MEMORY EXAMINE/CHANGE
220	0000e0d7	62		FCB	\$62	14	G EMULATE
221	0000e0d8	54		FCB	\$54	15	V VERIFY RAM
222	0000e0d9	52		FCB	\$52	16	L LOAD RAM
223	0000e0da	51		FCB	\$51	17	P LOAD FROM \$4000

em64k.as5

225
 226
 227
 228
 229
 230
 231
 232 0000e0db 1931
 233 0000e0dd 1531
 234 0000e0df cde235
 235 0000e0e2 a6f4
 236 0000e0e4 b724
 237 0000e0e6 a677
 238 0000e0e8 b725
 239 0000e0ea cde1ec
 240 0000e0ed cde5be
 241 0000e0f0 2423
 242 0000e0f2 b630
 243 0000e0f4 b726
 244 0000e0f6 b62b
 245 0000e0f8 b727
 246 0000e0fa cde570
 247 0000e0fd b72f
 248 0000e0ff cde235
 249 0000e102 a6f1
 250 0000e104 b724
 251 0000e106 a677
 252 0000e108 b725
 253 0000e10a cde1ec
 254 0000e10d cde5be
 255 0000e110 2403
 256 0000e112 b630
 257 0000e114 81
 258 0000e115 1831
 259 0000e117 81
 260
 261
 262
 263
 264
 265
 266
 267 0000e118 bf2f
 268 0000e11a 3f33
 269 0000e11c be2f
 270 0000e11e d6e132
 271 0000e121 be33
 272 0000e123 e720
 273 0000e125 3c2f
 274 0000e127 3c33
 275 0000e129 b633
 276 0000e12b a106
 277 0000e12d 25ed
 278 0000e12f cce1ec
 279
 280 0000e132 000d0d777e6
 281 0000e138 d6f1600671b6

```
*****
*
*   Build a beginning and ending address   *
*   in TEMP,TEMP+1 & ADDRH,ADDRL resp.   *
*
*****
```

```
BLDRNG  BCLR  4,STAT
        BCLR  2,STAT      EMULATION ADDRESS
        JSR   CLRTAB      PRINT
        LDA   #$F4        'BA'
        STA   DTABL+4
        LDA   #$77
        STA   DTABL+5
        JSR   DISTAB
        JSR   BLDADR      GET SOURCE ADDR.
        BCC   BLDRN1     VALID?
        LDA   ADDRH      YES
        STA   TEMP      SAVE IT
        LDA   ADDRL
        STA   TEMP+1
        JSR   LOAD       FETCH OPCODE OF INSTR.
        STA   W6         SAVE IT
        JSR   CLRTAB
        LDA   #$F1      PRINT 'EA'
        STA   DTABL+4
        LDA   #$77
        STA   DTABL+5
        JSR   DISTAB
        JSR   BLDADR     GET DESTINATION ADDR
        BCC   BLDRN1     VALID?
        LDA   ADDRH     YES
        RTS
BLDRN1  BSET  4,STAT     INVALID
        RTS
```

```
*****
*
*   Display message.                       *
*
*****
```

```
DISP   STX   W6
        CLR   COUNT
DISLP  LDX   W6
        LDA   DLOAD,X
        LDX   COUNT
        STA   DTABL,X
        INC   W6
        INC   COUNT
        LDA   COUNT
        CMP   #6
        BLO  DISLP
        JMP   DISTAB
DLOAD  FCB   0,0,$D0,$D7,$77,$E6
VERF   FCB   $D6,$F1,$60,$06,$71,$86
```

```

283
284
285
286
287
288
289 0000e13e ad9b
290 0000e140 08313e
291 0000e143 b62b
292 0000e145 a002
293 0000e147 b72b
294 0000e149 b630
295 0000e14b a200
296 0000e14d b730
297 0000e14f b62b
298 0000e151 b027
299 0000e153 b72b
300 0000e155 b630
301 0000e157 b226
302 0000e159 b730
303 0000e15b b62f
304 0000e15d a11f
305 0000e15f 234e
306 0000e161 b630
307 0000e163 a1ff
308 0000e165 270b
309 0000e167 4d
310 0000e168 2674
311 0000e16a b62b
312 0000e16c a17f
313 0000e16e 226e
314 0000e170 200a
315
316 0000e172 b62b
317
318 0000e174 a1ff
319 0000e176 2766
320
321 0000e178 a180
322 0000e17a 2562
323 0000e17c ad06
324 0000e17e cce000
325
326 0000e181 cce04e

```

```

*****
*                                     *
*      Calculate branch offset.      *
*                                     *
*****
BRAOFF BSR      BLORNG
        BRSET   4,STAT,ORET
        LDA     ADDR1      NO FIND APPARENT
        SUB     #2
        STA     ADDR1
        LDA     ADDR2
        SBC     #0
        STA     ADDR2
        LDA     ADDR1
        SUB     TEMP+1      OFFSET
        STA     ADDR1
        LDA     ADDR2
        SBC     TEMP
        STA     ADDR2
        LDA     W6          CHECK OP CODE
        CMP     #$1F        FOR BIT BRANCH
        BLS     OFFST1
        LDA     ADDR2
        CMP     #$FF        + OR - OFFSET?
        BEQ     OFFST2
        TSTA
        BNE     OVRERR      CHECK OFFSET
        LDA     ADDR1        FOR +/- 0
        CMP     #$7F
        BHI     OVRERR
        BRA     OK1
OFFST2 LDA     ADDR1
        CMP     #$FF
        BEQ     OVRERR
        CMP     #$80
        BLO     OVRERR
OK1    BSR     USE          PRINT IT IF VALID
        JMP     SCAN
ORET  JMP     GETCMD

```

em64k.as5

```

328
329 0000e184 cde235
330 0000e187 a6d6
331 0000e189 b720
332 0000e18b a6b5
333 0000e18d b721
334 0000e18f a6f1
335 0000e191 b722
336 0000e193 a6e6
337 0000e195 b723
338 0000e197 b62b
339 0000e199 cde5f2
340 0000e19c 97
341 0000e19d b627
342 0000e19f ab01
343 0000e1a1 b72b
344 0000e1a3 b626
345 0000e1a5 a900
346 0000e1a7 b730
347 0000e1a9 9f
348 0000e1aa 1531

```

```

USE   JSR     CLR1TAB
        LDA     #$D6        PRINT 'USED'
        STA     DTABL
        LDA     #$B5
        STA     DTABL+1
        LDA     #$F1
        STA     DTABL+2
        LDA     #$E6
        STA     DTABL+3
        LDA     ADDR1      PRINT OFFSET
        JSR     PR1DAT
        TAX
        LDA     TEMP+1
        ADD     #1
        STA     ADDR1
        LDA     TEMP
        ADC     #0          PUT INTO
        STA     ADDR2      INSTRUCTION
        TXA
        BCLR   2,STAT

```

```

349 0000e1ac cce562          JMP     STORE
350
351 0000e1af b62b          OFFST1 LDA   ADDR1      ADJUST FOR
352 0000e1b1 a001          SUB     #1           BIT BRANCH
353 0000e1b3 b72b          STA   ADDR1
354 0000e1b5 b630          LDA   ADDR1H
355 0000e1b7 a200          SBC   #0
356 0000e1b9 b730          STA   ADDR1H
357 0000e1bb a1ff          CMP   #FF           NEG OFFSET?
358 0000e1bd 270b          BEQ   OFFST3        YES
359 0000e1bf 4d          TSTA          CHECK FOR
360 0000e1c0 261c          BNE   OVRERR       +/- 0 AND -1
361 0000e1c2 b62b          LDA   ADDR1
362 0000e1c4 a17f          CMP   #7F
363 0000e1c6 2216          BHI   OVRERR
364 0000e1c8 200a          BRA   OK2
365
366 0000e1ca b62b          OFFST3 LDA   ADDR1
367 0000e1cc a1fe          CMP   #FE
368 0000e1ce 240e          BHS   OVRERR
369 0000e1d0 a180          CMP   #80
370 0000e1d2 250a          BLO   OVRERR
371
372 0000e1d4 3c27          OK2    INC     TEMP+1
373 0000e1d6 2602          BNE   OFFITS
374 0000e1d8 3c26          INC   TEMP
375 0000e1da ada8          OFFITS BSR   USE      PRINT IF VALID
376 0000e1dc 200b          BRA   SCJMP
377
378 0000e1de a6d7          OVRERR LDA   #D7       PRINT "OR"
379 0000e1e0 b724          STA   DTABL+4
380 0000e1e2 a660          LDA   #60
381 0000e1e4 b725          STA   DTABL+5
382 0000e1e6 cde616        JSR   PRTADR
383 0000e1e9 cce000        SCJMP  JMP   SCAN

```

em64k.as5

```

385 *****
386 *
387 *      Display table contents.
388 *
389 *****
390
391 0000e1ec 1701          DISTAB BCLR  3,PORTB   ENABLE (144115) LOW
392 0000e1ee ae05          LDX   #5
393 0000e1f0 e620          DISCHR LDA   DTABL,X  LOAD DISPLAY
394
395 0000e1f2 bf28          NT1    STX   W1       SAVE INDEX
396 0000e1f4 1d00          BCLR  6,PORTA      CLEAR DATA
397 0000e1f6 ae08          LDX   #8
398 0000e1f8 48          DIS1   LSLA          SET UP
399 0000e1f9 2402          BCC   DIS2         BIT OF
400 0000e1fb 1c00          BSET  6,PORTA      ACCUMULATOR
401 0000e1fd 1e00          DIS2   BSET  7,PORTA  CLOCK
402 0000e1ff 1f00          BCLR  7,PORTA      IT
403 0000e201 1d00          BCLR  6,PORTA      CLEAR DATA
404 0000e203 5a          DECX          COMPLETE?
405 0000e204 26f2          BNE   DIS1         NO
406 0000e206 be28          LDX   W1          RESTORE INDEX
407 0000e208 5a          DECX
408 0000e209 2ae5          BPL   DISCHR
409 0000e20b 1601          BSET  3,PORTB     ENABLE (144115) HIGH
410 0000e20d 81          RTS
411

```

```

412
413
414
415
416
417
418 0000e20e 1201 RAMACC BSET 1,PORTB 5 5 XFER A14 & A15 TO PORTB
419 0000e210 0e3002 BRSET 7,ADDRH,A15H 5 10 A15 HIGH ?
420 0000e213 1301 BCLR 1,PORTB 5 15 NO
421 0000e215 1001 A15H BSET 0,PORTB 5 20 YES
422 0000e217 0c3002 BRSET 6,ADDRH,A14H 5 25 A14 HIGH ?
423 0000e21a 1101 BCLR 0,PORTB 5 30 NO
424 0000e21c be30 A14H LDX ADDRH 3 33 YES
425 0000e21e bf2a STX ADDEH 4 37
426 0000e220 1c2a BSET 6,ADDEH 5 42 A14 HIGH
427 0000e222 1f2a BCLR 7,ADDEH 4 47 A15 LOW
428
429 0000e224 0a3108 BRSET 5,STAT,L3 5 52 READING ?
430 0000e227 aec7 LDX #$C7 2 54 NO, WRITING (STA)
431 0000e229 bf29 STX W2 4 58 STA IN
432 0000e22b bd29 JSR W2 16 74 RAM SUBROUTINE
433 0000e22d b72d STA W4 4 78 SAVE FOR READBACK CHECK
434 0000e22f aec6 L3 LDX #$C6 2 80 READING (LDA)
435 0000e231 bf29 STX W2 13 93 LDA IN RAM
436
437 0000e233 bc29 JMP W2 14 107 121 (89 FOR READ) WITH JSR

```

em64k.as5

```

439
440
441
442
443
444
445 0000e235 ae05 CLR TAB LDX #5
446 0000e237 6f20 CLRLOC CLR DTABL,X CLEAR SIX
447 0000e239 5a DECX LOCATIONS IN
448 0000e23a 2afb BPL CLRLOC DISPLAY TABLE
449 0000e23c 81 RTS
450
451
452
453
454
455
456
457 0000e23d cde235 MODE3 JSR CLR TAB
458 0000e240 a6f1 LDA #$F1 E
459 0000e242 b720 STA DTABL
460 0000e244 a673 LDA #$73 P
461 0000e246 b721 STA DTABL+1
462 0000e248 a663 LDA #$63 ?
463 0000e24a b723 STA DTABL+3
464 0000e24c cde1ec JSR DISTAB
465 0000e24f cde05f KSC JSR KEYSCHN WAIT UNTIL EPROM REMOVED
466 0000e252 24fb BCC KSC
467 0000e254 a162 CMP #$62 EMULATION CONFIRMED ?
468 0000e256 2703 BEQ CONF
469 0000e258 cce04e JMP GETCMD
470
471 0000e25b a628 CONF LDA #$28 MODE 3, ENABLE (144115) HIGH
472 0000e25d b701 STA PORTB
473 0000e25f a6f1 LDA #$F1 E
474 0000e261 b720 STA DTABL
475 0000e263 a6d6 LDA #$D6 U
476 0000e265 b721 STA DTABL+1
477 0000e267 a6d0 LDA #$D0 L

```

```

478 0000e269 b722          STA  DTABL+2
479 0000e26b a677          LDA  #$77          A
480 0000e26d b723          STA  DTABL+3
481 0000e26f a6f0          LDA  #$F0          T
482 0000e271 b724          STA  DTABL+4
483 0000e273 a6f1          LDA  #$F1          E
484 0000e275 b725          STA  DTABL+5
485 0000e277 cde1ec        JSR  DISTAB
486
487 0000e27a 8e          STP  STOP

```

em64k.as5

```

489
490
491
492
493
494
495
*****
*
*
*      Xfer EPROM contents to RAM from emulation *
*      socket ($4000).
*
*****
496 0000e27b cde235        DUMP1 JSR  CLRTAB
497 0000e27e cde1ec        JSR  DISTAB
498 0000e281 a658          LDA  #$58          MODE 2, ENABLE (144115) HIGH
499 0000e283 b701          STA  PORTB
500 0000e285 1431          BSET 2,STAT        REAL ADDRESS (NO OFFSET)
501 0000e287 3f2b          CLR  ADDR1
502 0000e289 3f30          CLR  ADDRH
503 0000e28b 1201          LLP1 BSET 1,PORTB    5 5  XFER A14 & A15 TO PORTB
504 0000e28d 0e3002        BRSET 7,ADDRH,AD15H 5 10 A15 HIGH ?
505 0000e290 1301          BCLR 1,PORTB     5 15 NO
506 0000e292 1001          AD15H BSET 0,PORTB   5 20 YES
507 0000e294 0c3002        BRSET 6,ADDRH,AD14H 5 25 A14 HIGH ?
508 0000e297 1101          BCLR 0,PORTB     5 30 NO
509 0000e299 be30          AD14H LDX  ADDRH    3 33 YES
510 0000e29b bf2a          STX  ADDEH        4 37
511 0000e29d 1f2a          BCLR 7,ADDEH     5 42 A15 LOW
512 0000e29f 1c2a          BSET 6,ADDEH     5 47 A14 HIGH
513 0000e2a1 1d01          BCLR 6,PORTB
514 0000e2a3 1e01          BSET 7,PORTB    READ FROM EMULATOR SOCKET
515 0000e2a5 cde570        JSR  LOAD          LOAD BYTE
516 0000e2a8 1c01          BSET 6,PORTB
517 0000e2aa 1f01          BCLR 7,PORTB    WRITE TO RAM
518 0000e2ac cde562        JSR  STORE        STORE BYTE
519 0000e2af 3c2b          INC  ADDR1
520 0000e2b1 2602          BNE  SKPH
521 0000e2b3 3c30          INC  ADDRH
522 0000e2b5 b630          SKPH LDA  ADDRH
523 0000e2b7 26d2          BNE  LLP1         MSB ZERO ?
524 0000e2b9 b62b          LDA  ADDR1        YES, LSB ZERO ?
525 0000e2bb 26ce          BNE  LLP1         IF SO, FINISHED
526 0000e2bd cce04e        JMP  GETCMD
527

```

```

528
529
530
531
532
533
534 0000e2c0 1c31
535 0000e2c2 1431
536 0000e2c4 a6d7
537 0000e2c6 b720
538 0000e2c8 a671
539 0000e2ca b721
540 0000e2cc b722
541 0000e2ce 3f23
542 0000e2d0 3f2a
543 0000e2d2 3f30
544 0000e2d4 a639
545 0000e2d6 b72b
546 0000e2d8 cce506

```

```

*****
*                                     *
*      Offset for S-record load/send.  *
*                                     *
*****
OFFSET  BSET      6.STAT      NO ADDRESS INC/DEC
          BSET      2.STAT      REAL ADDRESS
          LDA      #$D7          0
          STA      DTABL
          LDA      #$71          F
          STA      DTABL+1
          STA      DTABL+2
          CLR      DTABL+3
          CLR      ADDEH
          CLR      ADDRH
          LDA      #OFF
          STA      ADDRl
          JMP      MEMEX3

```

em64k.as5

```

548
549
550
551
552
553
554
555 0000e2db cde235
556 0000e2de cde1ec
557 0000e2e1 a658
558 0000e2e3 b701
559 0000e2e5 1431
560
561 0000e2e7 ad0f
562 0000e2e9 3c01
563 0000e2eb b601
564 0000e2ed a403
565 0000e2ef a103
566 0000e2f1 25f4
567 0000e2f3 ad03
568 0000e2f5 cce04e
569
570 0000e2f8 3f2b
571 0000e2fa 3f30
572
573 0000e2fc be30
574 0000e2fe bf2a
575 0000e300 1e2a
576 0000e302 1d2a
577 0000e304 cde570
578 0000e307 1f2a
579 0000e309 1c2a
580 0000e30b cde562
581 0000e30e 3c2b
582 0000e310 2602
583 0000e312 3c30
584 0000e314 b630
585 0000e316 a140
586 0000e318 26e2
587
588 0000e31a 81

```

```

*****
*                                     *
*      Xfer EPROM contents to RAM from auxiliary *
*      socket ($8000).                          *
*                                     *
*****
DUMP9   JSR      CLRTAB
          JSR      DISTAB
          LDA      #$58          MODE 2. ENABLE (144115) HIGH
          STA      PORTB
          BSET     2.STAT      REAL ADDRESS (NO OFFSET)
TLOP    BSR      T19
          INC     PORTB          NEXT PAGE
          LDA     PORTB
          AND     #3
          CMP     #3          LAST PAGE ?
          BLO    TLOP
          BSR     T19          YES
          JMP     GETCMD
T19     CLR      ADDRl
          CLR      ADDRH
LLP9    LDX      ADDRH          YES
          STX     ADDEH
          BSET    7.ADDEH      A15 HIGH
          BCLR    6.ADDEH      A14 LOW
          JSR     LOAD          READ FROM AUXILIARY SOCKET
          BCLR    7.ADDEH      A15 LOW
          BSET    6.ADDEH      A14 HIGH
          JSR     STORE        WRITE TO EMULATION RAM
          INC     ADDRl
          BNE    SKPH9
          INC     ADDRH
SKPH9   LDA      ADDRH
          CMP     #$40          LAST ADDRESS $3FFF
          BNE    LLP9          FINISHED ?
          RTS

```

em64k.as5

```

590
591
592
593
594
595
596 0000e31b 1a31 VERIFY BSET 5,STAT SERIAL VERIFY
597 0000e31d ae06 LDX #6
598 0000e31f 2003 BRA L4
599 0000e321 1b31 TLOAD BCLR 5,STAT SERIAL LOAD
600 0000e323 5f CLRX
601 0000e324 a681 L4 LDA #8B1 RTS
602 0000e326 b72c STA W3
603 0000e328 cde118 JSR DISP DISPLAY "LOAD OR UErIfy"
604
605 0000e32b ad5f INPUT BSR INCHD 7 BIT ASCII INTO A
606 0000e32d a153 CMP #'S' S ?
607 0000e32f 26fa BNE INPUT NO, TRY AGAIN
608 0000e331 ad59 BSR INCHD YES, GET NEXT CHARACTER
609 0000e333 a139 CMP #'9' 9 ?
610 0000e335 276f BEQ NINE YES, FINISH
611 0000e337 a131 CMP #'1' NO, 1 ?
612 0000e339 26f0 BNE INPUT NO, TRY AGAIN
613
614 0000e33b 3f32 LENGTH CLR CHKSUM YES, CLEAR CHECKSUM
615 0000e33d 3f35 CLR TMP2 AND TEMP. STORE
616 0000e33f cde3bd JSR BYTEI AND GET BYTE COUNT
617 0000e342 b736 STA BCNT AND SAVE IT
618
619 0000e344 cde3bd ADDR JSR BYTEI ADDRESS HIGH
620 0000e347 b039 SUB OFF OFFSET
621 0000e349 b730 STA ADDRH
622 0000e34b cde3bd JSR BYTEI ADDRESS LOW
623 0000e34e b72b STA ADDRL
624
625 0000e350 cde3bd DLOP JSR BYTEI 75 GET A BYTE
626 0000e353 271d BEQ CHCK 3 78 LAST BYTE ?
627 0000e355 0b310b BRCLR 5,STAT,L5 5 83 NO, VERIFYING ?
628 0000e358 b72f STA W6 4 87 YES
629 0000e35a cde20e JSR RAMACC 66 153 READ RAM
630 0000e35d b12f CMP W6 3 156 SAME ?
631 0000e35f 2658 BNE ERR7 3 159
632 0000e361 2007 BRA L6 3 162
633 0000e363 cde20e L5 JSR RAMACC 67 150 NO, WRITE TO RAM
634 0000e366 b12d CMP W4 3 153 READBACK
635 0000e368 263f BNE ERR2 3 156 OK ?
636 0000e36a 3c2b L6 INC ADDRL 5 161 5 167 INCREMENT L5 ADDRESS
637 0000e36c 2602 BNE NOOVR 3 164 3 170 OVERFLOW ?
638 0000e36e 3c30 INC ADDRH 5 169 5 175 YES, INC. HIGH BYTE
639 0000e370 20de NOOVR BRA DLOP 3 172 3 178

```

em64k.as5


```

641
642
643
644
645
646
647 0000e372 bb32
648 0000e374 b732
649 0000e376 a1ff
650 0000e378 27b1
651
652 0000e37a ae01
653 0000e37c b738
654 0000e37e bf37
655 0000e380 9f
656 0000e381 cde5f2
657 0000e384 3f24
658 0000e386 cde16
659 0000e389 cce000
660
661
662
663
664
665
666
667
668 0000e38c ad60
669 0000e38e 0501fd
670 0000e391 0401fd
671 0000e394 ae07
672 0000e396 bf33
673 0000e398 ad58
674
675 0000e39a ad52
676 0000e39c 050100
677 0000e39f 46
678 0000e3a0 3a33
679 0000e3a2 26f6
680
681 0000e3a4 44
682 0000e3a5 81
683
684 0000e3a6 cce04e
685
686 0000e3a9 ae02
687 0000e3ab 20cf
688 0000e3ad ae03
689 0000e3af 20cb
690 0000e3b1 ae04
691 0000e3b3 20c7
692 0000e3b5 ae05
693 0000e3b7 20c3
694 0000e3b9 ae07
695 0000e3bb 20bf

```

```

*****
*
*      Checksum byte & error routine.
*
*****
CHCK  ADD    CHKSUM
      STA    CHKSUM          DEBUG
      CMP    #$FF          IS CHECKSUM BYTE OK ?
      BEQ    INPUT          YES, AND AGAIN

ERR   LDX    #1
      STA    ERDAT          DEBUG
      STX    ERTYP          DEBUG
      TXA
      JSR    PRTDAT
      CLR    DTABL+4
      JSR    PRTADR
      JMP    SCAN

*****
*
*      Input routine, MC68HC05E0 : 0.5 uS.
*      Cycles per bit at 9600 baud : 208
*
*****
INCHD BSR    DEL191          191    GET OUT OF BIT 7
INCH  BRCLR  2,PORTB,*      5      IS LINE HIGH ?
      BRSET  2,PORTB,*      5      YES, WAIT FOR START
      LDX   #7              2  6    7 DATA BITS TO READ
      STX   COUNT          4  10
      BSR   DEL110         110 120  +/-3 OF 1st BIT

INBT  BSR    DEL191          191    +120-208=103
      BRCLR  2,PORTB,ZER    5  196  CYC 2 (105) READ
ZER   RORA   4              3  199  SAVE BIT
      DEC    COUNT          5  204
      BNE   INBT           3  207

      LSRA   3  16         MSB A ZERO
      RTS   6  22

NINE  JMP    GETCMD

ERR2  LDX   #2              READBACK FROM RAM
      BRA   ERR
ERR3  LDX   #3              LESS THAN ASCII 0
      BRA   ERR
ERR4  LDX   #4              BETWEEN ASCII 9 & A
      BRA   ERR
ERR5  LDX   #5              MORE THAN ASCII F
      BRA   ERR
ERR7  LDX   #7              VERIFY ERROR
      BRA   ERR

```

em64k.as5

```

697
698
699
700
701
702
703 0000e3bd adcd      BYTEI  BSR      INCHD      22      MS NIBBLE
704 0000e3bf ad17      BSR      ASCII      35  57  WHAT WAS IT
705 0000e3c1 48        LSLA      3          YES
706 0000e3c2 48        LSLA      3          SHIFT
707 0000e3c3 48        LSLA      3          IT
708 0000e3c4 48        LSLA      3  69    UP
709 0000e3c5 b734      STA      TMP1      4  71    AND SAVE IT
710 0000e3c7 b635      LDA      TMP2      3  74    RESTORE BYTE
711 0000e3c9 b332      ADD      CHKSUM    3  79    ACCUMULATE
712 0000e3cb b732      STA      CHKSUM    4  83    IN CHECKSUM BYTE
713 0000e3cd adbd      BSR      INCHD      22      LS NIBBLE
714 0000e3cf ad07      BSR      ASCII      35  57  WHAT WAS IT
715 0000e3d1 b334      ADD      TMP1      3  60    ADD TO MS NIBBLE
716 0000e3d3 b735      STA      TMP2      4  64    SAVE BYTE
717 0000e3d5 3a36      DEC      BCNT      5  69    DECREMENT BYTE COUNT
718 0000e3d7 81        RTS      6  75
719
720 0000e3d8 a130      ASCII    CMP      #$30      2          BEFORE ZERO ?
721 0000e3da 25d1      BLO      ERR3      3  5          YES, NOT LEGAL
722 0000e3dc a139      CMP      #$39      2  7          AFTER NINE
723 0000e3de 2203      BHI      MT9       3  10         YES TRY A-F
724 0000e3e0 a030      SUB      #$30      3  13         0-9, CONVERT TO HEX
725 0000e3e2 81        RTS      6  19
726
727 0000e3e3 a141      MT9      CMP      #$41      2  12         BEFORE A ?
728 0000e3e5 25ca      BLO      ERR4      3  15         YES, NOT LEGAL
729 0000e3e7 a146      CMP      #$46      2  17         AFTER F ?
730 0000e3e9 22ca      BHI      ERR5      3  20         YES, NOT LEGAL
731 0000e3eb a037      SUB      #$37      3  23         A-F, CONVERT TO HEX
732 0000e3ed 81        NFND     RTS      6  29
733
734 0000e3ee ae1d      DEL191  LDX      #29       2
735 0000e3f0 2002      BRA      DELAY     3  5
736 0000e3f2 ae10      DEL110  LDX      #16       2
737 0000e3f4 5a        DELAY    DECX     3
738 0000e3f5 26fd      BNE      DELAY     3          6xX
739 0000e3f7 81        RTS      6          12+6X (INC BSR)

```

em64k.as5

```

741
742
743
744
745
746
747 0000e3f8 1406      PUNCH   BSET     2,PORTBD   BIT 2 OUTPUT
748 0000e3fa cde0db    JSR     BLDRNG   BUILD RANGE
749 0000e3fd 0831a6    BRSET   4,STAT,NINE  NEW ADDRESS ENTERED ?
750 0000e400 be26      LDX     TEMP     NO. SWAP ADDRESSES
751 0000e402 b726      STA     TEMP
752 0000e404 bf30      STX     ADDRH
753 0000e406 b62b      LDA     ADDRLL
754 0000e408 be27      LDX     TEMP+1
755 0000e40a bf2b      STX     ADDRLL
756 0000e40c b727      STA     TEMP+1
757 0000e40e 1f31      BCLR   7,STAT     CLEAR END FLAG
758 0000e410 1531      BCLR   2,STAT     EMULATION ADDRESS
759
760 0000e412 b627      LOOP1   LDA     TEMP+1   END LSB
761 0000e414 b02b      SUB     ADDRLL     CURRENT LSB

```

762 0000e416	b72f	STA	W6	DIFFERENCE LSB
763 0000e418	b626	LDA	TEMP	END MSB
764 0000e41a	b230	SBC	ADDRH	CURRENT MSB
765 0000e41c	260d	BNE	LOTS	MSB ZERO ?
766 0000e41e	b62f	LDA	W6	YES, LOOK AT LSB
767 0000e420	4c	INCA		ADJUST
768 0000e421	2708	BEQ	LOTS	WAS \$FF ?
769 0000e423	a120	CMP	#\$20	MORE THAN 23 ?
770 0000e425	2204	BHI	LOTS	IF SO USE 23
771 0000e427	1e31	BSET	7,STAT	NO, LAST S1 RECORD
772 0000e429	2002	BRA	LTE20	LESS THAN OR EQUAL TO 20
773				
774 0000e42b	a620	LOTS	LDA	#\$20
775 0000e42d	ab03	LTE20	ADD	#\$03
776 0000e42f	b736		STA	BCNT
777 0000e431	a653		LDA	'S'
778 0000e433	cde484		JSR	OUCH
779 0000e436	a631		LDA	'1'
780 0000e438	ad4a		BSR	OUCH
781 0000e43a	3f32		CLR	CHKSUM
782 0000e43c	b636		LDA	BCNT
783 0000e43e	ad72		BSR	BYTE0
784 0000e440	b630		LDA	ADDRH
785 0000e442	ad6e		BSR	BYTE0
786 0000e444	b62b		LDA	ADDRL
787 0000e446	ad6a		BSR	BYTE0
788				ADDRESS LOW
789 0000e448	cde570	LOOP2	JSR	LOAD
790 0000e44b	3c2b		INC	ADDRL
791 0000e44d	2602		BNE	NOVR
792 0000e44f	3c30		INC	ADDRH
793 0000e451	ad5f	NOVR	BSR	BYTE0
794 0000e453	26f3		BNE	LOOP2

em64k.as5

796		*****		
797		*		*
798		*	Checksum byte.	*
799		*		*
800		*****		
801				
802 0000e455	b632	LDA	CHKSUM	CHECKSUM
803 0000e457	43	COMA		REQUIRED CHECKSUM BYTE
804 0000e458	ad58	BSR	BYTE0	SEND IT
805 0000e45a	ad22	BSR	CRLF	CRLF
806 0000e45c	0f31b3	BRCLR	7,STAT,LOOP1	FINISHED ?
807				
808		*****		
809		*		*
810		*	S9 record.	*
811		*		*
812		*****		
813				
814 0000e45f	a653	LDA	'S'	S
815 0000e461	ad21	BSR	OUCH	
816 0000e463	a639	LDA	'9'	9
817 0000e465	ad1d	BSR	OUCH	
818 0000e467	a603	LDA	#\$03	3 BYTES
819 0000e469	ad47	BSR	BYTE0	
820 0000e46b	a600	LDA	#\$00	
821 0000e46d	ad43	BSR	BYTE0	DUMMY (0)
822 0000e46f	a600	LDA	#\$00	
823 0000e471	ad3f	BSR	BYTE0	ADDRESS
824 0000e473	a6fc	LDA	#\$FC	
825 0000e475	ad3b	BSR	BYTE0	CHECKSUM
826 0000e477	ad05	BSR	CRLF	
827 0000e479	1506	BCLR	2,PORTBD	BIT 2 INPUT

```

828 0000e47b cce04e      JMP      GETCMD
829
830 0000e47e a60d      CRLF    LDA      #$0D      CR
831 0000e480 ad02      BSR     OUCH
832 0000e482 a60a      LDA     #$0A      LF

```

em64k.as5

```

834 *****
835 *
836 *      Output routine, 208 cycles per bit.
837 *
838 *****
839
840 0000e484 1401      OUCH    BSET    2,PORTB      5      MAKE SURE IT'S HIGH
841 0000e486 ae0a      LDX     #10      2 7      10 BITS TO SEND
842 0000e488 bf33      STX     COUNT    4 11     START, 8 DATA, STOP
843 0000e48a 9d      NOP
844 0000e48b 9d      NOP
845 0000e48c cde3f4    JSR     DELAY    72 83
846 0000e48f 98      CLC
847 0000e490 2008     BRA     STAR     2 85     START A ZERO
848
849 0000e492 ae1c      OUTBT   LDX     #28      2
850 0000e494 cde3f4    JSR     DELAY    180 182
851 0000e497 9d      NOP      2 184
852 0000e498 99      DEL3    SEC      2 186     FILL WITH ONES FOR STOP
853 0000e499 46      RORA    3 189     GET A BIT
854 0000e49a 2504     STAR    BCS     OUI     3 192     1 ?
855 0000e49c 1501     BCLR   2,PORTB  5 197     NO
856 0000e49e 2004     BRA     OBD     3 200
857 0000e4a0 1401     OUI    BSET    2,PORTB  5      YES
858 0000e4a2 2000     BRA     OBD     3
859 0000e4a4 3a33     OBD    DEC     COUNT    5 205
860 0000e4a6 26ea     BNE    OUTBT   3 208     DONE ?
861 0000e4a8 81      RTS
862
863 0000e4a9 ab30     ASCII0  ADD     #$30     3      CONVERT TO ASCII
864 0000e4ab a139     CMP     #$39     2 5      0-9 ?
865 0000e4ad 2302     BLS     NMT9    3 8
866 0000e4af ab07     ADD     #$07     3 8      NO, A-F
867 0000e4b1 81      NMT9    RTS     6 14
868
869 *****
870 *
871 *      Byte output sub-routine.
872 *
873 *****
874
875 0000e4b2 b734     BYTE0   STA     TMP1
876 0000e4b4 44      LSR     LSRA      SHIFT
877 0000e4b5 44      LSR     LSRA      DOWN TO
878 0000e4b6 44      LSR     LSRA      GET MSB
879 0000e4b7 44      LSR     LSRA
880 0000e4b8 adef     BSR     ASCII0   & CONVERT
881 0000e4ba adc8     BSR     OUCH
882 0000e4bc b634     LDA     TMP1     RESTORE BYTE
883 0000e4be bb32     ADD     CHKSUM   ACCUMULATE
884 0000e4c0 b732     STA     CHKSUM   IN CHECKSUM BYTE
885 0000e4c2 b634     LDA     TMP1
886 0000e4c4 a40f     AND     #$0F     LSB
887 0000e4c6 ade1     BSR     ASCII0   CONVERT IT
888 0000e4c8 adba     BSR     OUCH
889 0000e4ca 3a36     DEC     BCNT     DECREMENT BYTE COUNT
890 0000e4cc 81      RTS

```

em64k.as5

```

892
893
894
895
896
897
898 0000e4cd d7
899 0000e4ce 06
900 0000e4cf e3
901 0000e4d0 a7
902 0000e4d1 36
903 0000e4d2 b5
904 0000e4d3 f5
905 0000e4d4 07
906 0000e4d5 f7
907 0000e4d6 b7
908 0000e4d7 77
909 0000e4d8 f4
910 0000e4d9 d1
911 0000e4da e6
912 0000e4db f1
913 0000e4dc 71
914
915 0000e4dd cde235
916 0000e4e0 a6f1
917 0000e4e2 b721
918 0000e4e4 a660
919 0000e4e6 b722
920 0000e4e8 b723
921 0000e4ea cce059
922
923
924
925
926
927
928
929
930
931 0000e4ed cde05f
932 0000e4f0 24fb
933 0000e4f2 5f
934 0000e4f3 d1e0c3
935 0000e4f6 2703
936 0000e4f8 5c
937 0000e4f9 20f8
938 0000e4fb 9f
939 0000e4fc 81

```

```

*****
*
*      Segment codes for the MC145000.
*
*****

```

```

CTABL  FCB  $07  0
        FCB  $06  1
        FCB  $E3  2
        FCB  $A7  3
        FCB  $36  4
        FCB  $85  5
        FCB  $F5  6
        FCB  $07  7
        FCB  $F7  8
        FCB  $B7  9
        FCB  $77  A
        FCB  $F4  B
        FCB  $D1  C
        FCB  $E6  D
        FCB  $F1  E
        FCB  $71  F

```

```

ERROR  JSR    CLRTAB
        LDA    #$F1
        STA    DTABL+1
        LDA    #$60
        STA    DTABL+2
        STA    DTABL+3
        JMP    DSCN

```

```

*****
*
*      INPUT ONE CHARACTER
*      A REGISTER CONTAINS HEX VALUE
*      X REGISTER CONTAINS HEX VALUE
*
*****

```

```

CHRIN  JSR    KEYSCHN      GET KEY
        BCC   CHRIN        IF NOT VALID RETRY
        CLRX
CHRINI  CMP    STABL,X      CONVERT
        BEQ   CHRIN2       TO HEX
        INCX
        BRA   CHRIN1
CHRIN2  TXA    IF CANCEL
        RTS

```

em64k.as5

```

941
942
943
944
945
946
947 0000e4fd 1531
948 0000e4ff cde5b9
949 0000e502 a110
950 0000e504 2752
951
952 0000e506 ad68
953 0000e508 cde5f2
954 0000e50b cde5af
955 0000e50e a110
956 0000e510 2746
957 0000e512 a111

```

```

*****
*
*      Memory examine/change.
*
*****

```

```

MEMEX  BCLR   2,STAT      EMULATION ADDRESS
        JSR   GETADR      GET ADDRESS
        CMP   #$10        ESCAPE ?
        BEQ   MEMEX4
MEMEX3  BSR   LOAD        LOAD DATA
        JSR   PRDAT       PRINT IT
        JSR   GETNYB      GET NEW NIBBLE
        CMP   #$10        ESCAPE ?
        BEQ   MEMEX4
        CMP   #$11        ENTER ?

```

958	0000e514	271a	BEQ	ADRINC		
959	0000e516	a113	CMP	#\$13	MEMORY ?	
960	0000e518	272e	BEQ	ADRDEC		
961						
962	0000e51a	a10f	CMP	#\$0F		
963	0000e51c	2208	BHI	CMDMDL	VALID HEX ?	
964						
965	0000e51e	cde5f2	MEMEX1	JSR	PRTDAT	PRINT IT
966	0000e521	cde59d		JSR	GETBY2	SHIFT IN NEXT
967	0000e524	25f8		BCS	MEMEX1	IF VALID TRY AGAIN
968	0000e526	a111	CMDMDL	CMP	#\$11	ENTER ?
969	0000e528	2614		BNE	MEMEX2	NO
970	0000e52a	b629		LDA	W2	RESTORE ACCA
971	0000e52c	ad34		BSR	STORE	YES STORE IT
972	0000e52e	25d6		BCS	MEMEX3	STORE VALID ?
973	0000e530	0c3125	ADRINC	BRSET	6,STAT,MEMEX4	
974	0000e533	3c2b		INC	ADDRL	YES GOTO
975	0000e535	2602		BNE	MEMEX5	NEXT
976	0000e537	3c30		INC	ADDRH	
977	0000e539	cde616	MEMEX5	JSR	PRTADR	PRINT IT
978	0000e53c	20c8		BRA	MEMEX3	REPEAT
979	0000e53e	a113	MEMEX2	CMP	#\$13	M ?
980	0000e540	2616		BNE	MEMEX4	NO
981	0000e542	b629		LDA	W2	
982	0000e544	ad1c		BSR	STORE	
983	0000e546	25be		BCS	MEMEX3	
984	0000e548	0c310d	ADRDEC	BRSET	6,STAT,MEMEX4	
985	0000e54b	3d2b		TST	ADDRL	YES THEN
986	0000e54d	2602		BNE	CMDMB2	GET PREVIOUS
987	0000e54f	3a30		DEC	ADDRH	ADDRESS
988	0000e551	3a2b	CMDMB2	DEC	ADDRL	
989	0000e553	cde616		JSR	PRTADR	PRINT IT
990	0000e556	20ae		BRA	MEMEX3	REPEAT
991	0000e558	0d3102	MEMEX4	BRCLR	6,STAT,NORM2	
992	0000e55b	3f2b		CLR	ADDRL	
993	0000e55d	1d31	NORM2	BCLR	6,STAT	
994	0000e55f	cce04e		JMP	GETCMD	

em64k.as5

996						*****
997			*			*
998			*	LOAD/STORE AT ADDR(H), ADDRL		*
999			*			*
1000						*****
1001						
1002	0000e562	aec7	STORE	LDX	#\$C7	SET-UP
1003	0000e564	ad0c		BSR	LDSTCM	ROUTINE
1004	0000e566	b72d		STA	W4	TO DO
1005	0000e568	ad06		BSR	LOAD	TWO BYTE
1006	0000e56a	b12d		CMP	W4	STORE
1007	0000e56c	2701		BEQ	STRTS	
1008	0000e56e	99		SEC		
1009	0000e56f	81	STRTS	RTS		
1010						
1011	0000e570	aec6	LOAD	LDX	#\$C6	SET-UP ROUTINE
1012	0000e572	bf29	LDSTCM	STX	W2	TO DO
1013	0000e574	ae81		LDX	#\$B1	TWO BYTE
1014	0000e576	bf2c		STX	W3	LOAD
1015	0000e578	043120		BRSET	2,STAT,NORM	REAL ADDRESS ?
1016						
1017	0000e57b	b72d		STA	W4	
1018	0000e57d	b630		LDA	ADDRH	
1019	0000e57f	b039		SUB	OFF	
1020	0000e581	b72e		STA	W5	
1021	0000e583	b62d		LDA	W4	
1022						
1023	0000e585	1201	RMCC	BSET	1,PORTB	5 5 XFER A14 & A15 TO PORTB

1024	0000e587	0e2e02		BRSET	7,W5,A15HI	5	10	A15 HIGH ?
1025	0000e58a	1301		BCLR	1,PORTB	5	15	NO
1026	0000e58c	1001	A15HI	BSET	0,PORTB	5	20	YES
1027	0000e58e	0c2e02		BRSET	6,W5,A14HI	5	25	A14 HIGH ?
1028	0000e591	1101		BCLR	0,PORTB	5	30	NO
1029	0000e593	be2e	A14HI	LDX	W5	3	33	YES
1030	0000e595	bf2a		STX	ADDEH	4	37	
1031	0000e597	1c2a		BSET	6,ADDEH	5	42	A14 HIGH
1032	0000e599	1f2a		BCLR	7,ADDEH	4	47	A15 LOW
1033	0000e59b	bc29	NORM	JMP	W2	14	61	67 (66 FOR LDA) WITH JSR
1034								
1035								
1036								
1037								
1038								
1039								
1040								
1041	0000e59d	b729	GETBY2	STA	W2			
1042	0000e59f	ad0e		BSR	GETNYB			
1043	0000e5a1	240b		BCC	GETBRT			
1044	0000e5a3	3829		ASL	W2			
1045	0000e5a5	3829		ASL	W2			
1046	0000e5a7	3829		ASL	W2			
1047	0000e5a9	3829		ASL	W2			
1048	0000e5ab	ba29		ORA	W2			
1049	0000e5ad	99		SEC				
1050	0000e5ae	81	GETBRT	RTS				

```
*****
*
*   Build a byte.
*
*****
```

em64k.as5

1052								
1053								
1054								
1055								
1056								
1057								
1058								
1059	0000e5af	cde4ed	GETNYB	JSR	CHRN			GET CHARACTER
1060	0000e5b2	98		CLC				
1061	0000e5b3	a10f		CMP	#\$0F			VALID HEX?
1062	0000e5b5	2201		BHI	GETRET			NO
1063	0000e5b7	99		SEC				YES
1064	0000e5b8	81	GETRET	RTS				
1065								
1066								
1067								
1068								
1069								
1070								
1071								
1072								
1073	0000e5b9	cde235	GETADR	JSR	CLRTAB			BLANK DISPLAY
1074	0000e5bc	ad58		BSR	PRTADR			
1075	0000e5be	adef	BLOADR	BSR	GETNYB			GET CHARACTER
1076	0000e5c0	250a		BCS	GETAD1			VALID HEX?
1077	0000e5c2	a110		CMP	#\$10			
1078	0000e5c4	272b		BEQ	GETRTS			
1079	0000e5c6	a111		CMP	#\$11			NO ENTER?
1080	0000e5c8	2727		BEQ	GETRTS			NO TRY AGAIN
1081	0000e5ca	20ed		BRA	GETADR			
1082	0000e5cc	3f30	GETAD1	CLR	ADDRH			INIT HIGH ADDRESS
1083	0000e5ce	b72b		STA	ADDRL			PUT CHAR AWAY
1084	0000e5d0	ad44		BSR	PRTADR			PRINT NEW ADDRESS

```
*****
*
*   Get one character into ACCA
*   X destroyed, C set if hex.
*
*****
```

```
*****
*
*   Build address A,X dest., address
*   in ADDR/ADDRH, C set if new.
*
*****
```

```

1085 0000e5d2  addb
1086 0000e5d4  2412
1087 0000e5d6  48
1088 0000e5d7  48
1089 0000e5d8  48
1090 0000e5d9  48
1091 0000e5da  ae04
1092 0000e5dc  48
1093 0000e5dd  392b
1094 0000e5df  3930
1095 0000e5e1  5a
1096 0000e5e2  26f8
1097 0000e5e4  ad30
1098 0000e5e6  20ea
1099 0000e5e8  a110
1100 0000e5ea  2705
1101 0000e5ec  a111
1102 0000e5ee  26e2
1103 0000e5f0  99
1104 0000e5f1  81

```

```

GETALP  BSR  GETNYB  GET ANOTHER CHAR
        BCC  GETARG  VALID?
        ASLA  YES
        ASLA  SHIFT IT IN
        ASLA
        LDX  #4
GETASF  ASLA
        ROL  ADDR1
        ROL  ADDRH
        DECX
        BNE  GETASF
        BSR  PRTADR  PRINT NEW ADDR
        BRA  GETALP  GET ANOTHER CHAR
GETARG  CMP  #$10  ESCAPE ?
        BEQ  GETRTS
        CMP  #$11  ENTER ?
        BNE  GETALP  NO TRY AGAIN
        SEC
GETRTS  RTS  YES SET FLAG

```

em64k.as5

```

1106
1107
1108
1109
1110
1111
1112
1113
1114 0000e5f2  ae04
1115 0000e5f4  bf2B
1116 0000e5f6  b72d
1117 0000e5f8  44
1118 0000e5f9  44
1119 0000e5fa  44
1120 0000e5fb  44
1121 0000e5fc  97
1122 0000e5fd  d6e4cd
1123 0000e600  be28
1124 0000e602  e720
1125 0000e604  b62d
1126 0000e606  a40f
1127 0000e608  97
1128 0000e609  d6e4cd
1129 0000e60c  be28
1130 0000e60e  e721
1131 0000e610  cde1ec
1132 0000e613  b62d
1133 0000e615  81
1134
1135 0000e616  b72e
1136 0000e618  bf2c
1137 0000e61a  b630
1138 0000e61c  5f
1139 0000e61d  add5
1140 0000e61f  b62b
1141 0000e621  ae02
1142 0000e623  adcf
1143 0000e625  b62e
1144 0000e627  be2c
1145 0000e629  81
1146

```

```

*****
*
*   Print one byte (from A).
*
*   Print address ADDRH,ADDRL.
*
*****
PRTDAT  LDX  #4           PRINT IN LAST TWO LCD DIGITS
PRTBYT  STX  W1
        STA  W4
        LSRA
        LSRA
        LSRA
        LSRA
        TAX
        LDA  CTABL,X
        LDX  W1
        STA  DTABL,X
        LDA  W4
        AND  #$0F
        TAX
        LDA  CTABL,X
        LDX  W1
        STA  DTABL+1,X
        JSR  DISTAB
        LDA  W4
        RTS
PRTADR  STA  W5           PRINT ADDRESS (FIRST 4 DIGITS)
        STX  W3
        LDA  ADDRH
        CLRX
        BSR  PRTBYT
        LDA  ADDR1
        LDX  #2
        BSR  PRTBYT
        LDA  W5
        LDX  W3
        RTS

```


1147
1148
1149
1150
1151
1152
1153
1154
1155 0000fff4 e022
1156 0000fff6 e022
1157 0000fff8 e022
1158 0000fffa e04e
1159 0000fffc e022
1160 0000fffe e022
1161

```
*****  
*  
*      MC68HC05E0 Vectors.      *  
*  
*****
```

```
      ORG      $FFF4  
  
      FDB      START      SERIAL  
      FDB      START      TIMER B  
      FDB      START      TIMER A  
      FDB      GETCMD     EXTERNAL INTERRUPT  
      FDB      START      SWI  
      FDB      START      RESET  
  
      END
```

Driving LCDs with M6805 Microprocessors

By Peter Topping
MCU Applications Group
Motorola Ltd, East Kilbride

INTRODUCTION

M6805 microprocessors include a wide range of parts with a large diversity of on-chip features. These include A/D and D/A convertors, serial interfaces, timers and display drivers. The display drive capability of the microprocessors range from none beyond I/O pins, through high current ports, to specialised display drivers for LCDs and vacuum fluorescent displays.

The MC68HC05M series have vacuum fluorescent drive capabilities up to 40V. The MC68HC05L series include LCD drivers with capabilities ranging from the MC68HC05L6 (3 or 4 backplanes and 24 frontplanes) through the MC68HC05L7/9 with 8 or 16 backplanes and 60/40 frontplanes. The L9's 40 frontplanes can be expanded to 205 with three MC68HC68L9 expanders.

Microprocessors without special LCD circuitry can be used to drive single backplane LCDs directly but require regular software intervention if the requirement that the display receives only AC drive is to be met. Alternatively display driver chips can be used to interface microprocessors with single and multiple backplane displays.

This application note gives hardware and software examples for these different arrangements. The same methods also apply to other families of microprocessors, eg M6801 and M68HC11. The examples are arranged in the order of the number of backplanes.

SINGLE-BACKPLANE DISPLAYS

Single-backplane displays are commonly used where the number of segments required is limited, usually using the 7-segment format. They have the advantages over multiplexed displays of superior contrast and viewing angle and a wider range of operating voltage and temperature. They can be driven directly by microprocessors with the number of segments limited simply by the number of available pins which are (or can be configured as) outputs. An output pin is required for the backplane together with one for each segment. The ports are loaded with the segment data corresponding to the required display, as with any other peripheral being directly driven by I/O lines. In this case, however, the microprocessor must complement the signals (backplane and frontplanes) at regular intervals, thus satisfying the requirement that the display receives an AC waveform with only a small DC component. It is possible for interrupts to alter the timing of these voltage reversals and the programmer must ensure that the resultant DC component does not exceed that above which the life of the display is reduced.

An alternative method of driving single-backplane displays from microprocessors is to use an LCD driver. Figure 1 shows a 6-digit 7-segment circuit using 3 MC144115P LCD drivers. These chips are driven serially and constitute a simple shift register giving the programmer full control over the display.

They can also be simply cascaded to drive a display of any required size. Clearly, the number of chips and interconnections increases directly as the number of segments. This limits the practical size of a display using this arrangement. The output pin to segment connections can be chosen to suit the application. The arrangement used here has been chosen to be compatible with the 4-backplane MC145000 driver used in a later example.

The MC144115 has a three-line serial interface consisting of clock, data, and chip enable. The clock and data lines can be shared with other peripherals, provided that each peripheral has a separate enable line. The enable line can, however, be derived from the clock if no other chips share the clock and data. This method of saving an I/O line is used in application note ANE416. The MC144115 software example (listing 1) has been modified from the routine used in ANE416.

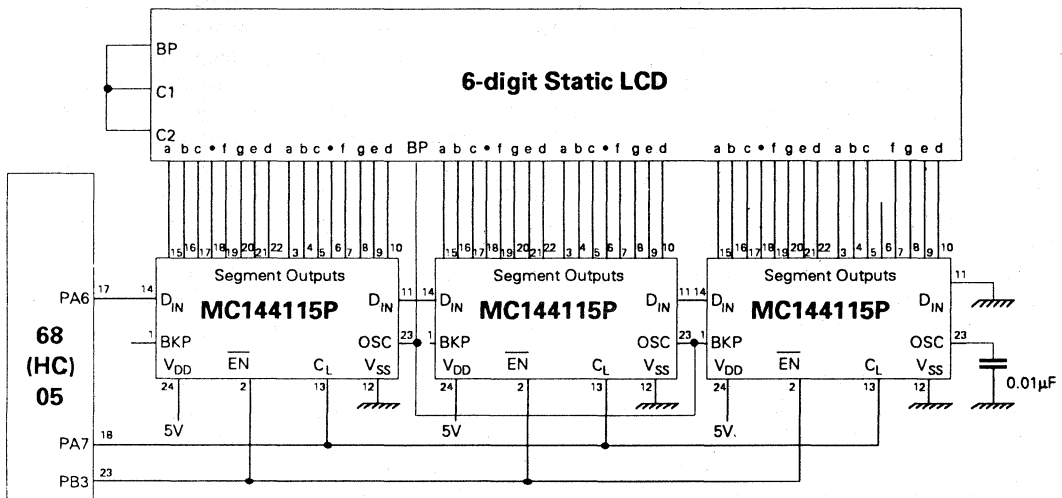


Figure 1. Single-backplane LCD display with MC144115P display drivers

LISTING 1

```

1
2
3
4
5
6
7
8
9 00000000          PORTA EQU $00          PORT A ADDRESS
10 00000001         PORTB EQU $01          " B "
11
12                   ORG $0050
13
14 00000050          R      RMB 6          WORKING NUMBER
15
16 00000056          TMP1  RMB 1          POSITION OF LSB
17 00000057          TMP2  RMB 1          POSITION OF MSB
18 00000058          TMP3  RMB 1
19 00000059          TMP4  RMB 1
20
21
22                   ORG $1000
23
24 *****
25 *
26 *      First part of the display subroutine
27 *      gets the segment codes corresponding to
28 *      the BCD data for display.
29 *
30 *****
31
32 00001000 a605      DISP LDA #05
33 00001002 b758          STA TMP3
34 00001004 be56          LDX TMP1          LSB
35 00001006 f6          D3  LDA 0,X
36 00001007 bf59          STX TMP4
37 00001009 97          TAX
38 0000100a d6103c      LDA STABL,X  FIND 7 SEGMENT CODE
39 0000100d be58          LDX TMP3
40 0000100f e750          STA R,X          PUT IN DISPLAY TABLE
41 00001011 3a58          DEC TMP3
42 00001013 be59          LDX TMP4
43 00001015 5a          DECX
44 00001016 b357          CPX TMP2          FINISHED ?
45 00001018 26ec          BNE D3

```

```

60
61
62
63
64
65
66
67
68 0000101a 1701      OUTT  BCLR  3,PORTB  ENABLE LOW
69
70 0000101c ae05      LDX   #5          SEND DISPLAY TABLE TO 144115
71 0000101e e650      DISCHR LDA  R,X
72 00001020 bf58      DISPLY STX  TMP3   SAVE INDEX
73 00001022 1d00      BCLR  6,PORTA  CLEAR DATA
74 00001024 ae08      LDX   #8
75 00001026 44      DIS1  LSRA          SET UP
76 00001027 2402      BCC   DIS2       BIT OF
77 00001029 1c00      BSET  6,PORTA  ACCUMULATOR
78 0000102b 1e00      DIS2  BSET  7,PORTA  CLOCK
79 0000102d 1f00      BCLR  7,PORTA  IT
80 0000102f 1d00      BCLR  6,PORTA  CLEAR DATA
81 00001031 5a      DECX          COMPLETE ?
82 00001032 26f2      BNE   DIS1       NO
83 00001034 be58      LDX   TMP3      RESTORE INDEX
84 00001036 5a      DECX
85 00001037 2ae5      BPL   DISCHR
86
87 00001039 1601      BSET  3,PORTB  ENABLE HIGH
88 0000103b 81      RTS
89
90
91
92
93
94
95
96 0000103c eb      STABL FCB  $EB    0  SEGMENT
97 0000103d 60      FCB  $60    1
98 0000103e c7      FCB  $C7    2  CODES
99 0000103f e5      FCB  $E5    3
100 00001040 6c      FCB  $6C    4  FOR THE
101 00001041 ad      FCB  $AD    5
102 00001042 af      FCB  $AF    6  MC145000/144115
103 00001043 e0      FCB  $E0    7
104 00001044 ef      FCB  $EF    8  LCD DRIVER
105 00001045 ed      FCB  $ED    9

```

```

*****
*
*   The second part of the display routine
*   sends the 48 bits required by the
*   display driver.
*
*****

```

```

*****
*
*   LCD segment table.
*
*****

```

THREE-BACKPLANE DISPLAYS

The MC68HC05L6 can drive 24 frontplanes and either 3 or 4 backplanes, the number of backplanes being selectable in software. The data to be displayed is arranged in the display RAM as shown in figure 2. Note that data sheet for the MC68HC05L6 (AD11254) shows this relationship wrongly.

It can be seen that each frontplane occupies a nibble in the 12-byte RAM. There is thus a simple relationship between RAM location and displayed digit on a 4-backplane 7-segment display (each 2 frontplane digit corresponds to one byte). With a 3-backplane display, however, each digit corresponds to 3 nibbles (1.5 bytes) so the software required to translate the required segments into display RAM data is more complex. Listing 2 shows a suggested method of doing this.

LCD data latch 00	7	6	5	4	3	2	1	0
(\$00)	Bp1	Bp2	Bp3	Bp4	Bp1	Bp2	Bp3	Bp4
	Fp 01				Fp 02			

Figure 2. MC68HC05L6 back/frontplane pin to LCD data latch bit relationship

The table which translates the required character into segments contains 2 bytes per character, the middle nibble of the 3 required being repeated. This simplifies the code required to write to the display RAM by using one nibble if the character is intended for an even position in the display and the other for an odd position. Figure 3 shows the L6 – LCD segment arrangement used in this example.

When using a microprocessor without an LCD drive capability a separate display driver can be used to drive a multiplexed display. The example shown in figure 4 and listing 3 uses the ICM7231B 3-backplane driver. The ICM7231B requires each character to be addressed through pins A0, A1 and A2 and the appropriate data written to pins D0, D1, D2 and D3. This parallel control uses more I/O lines than the serial arrangement employed in MC14500/1 and MC144115 drivers.

The fact that data is accepted in HEX and encoded into segments by the driver simplifies the software but reduces the versatility of the display as only the driver's 16 characters are available. The ICM7231B driver displays 0–9, -, E, H, L, P and blank while the ICM7231A displays 0–9, A, B, C, D, E and F.

As with any multiplexed LCD drive, the contrast is dependent on the supply voltage to the driver's multiplexer circuitry. In the case of the ICM7231, contrast can be adjusted using the potentiometer on pin 2 (figure 4).

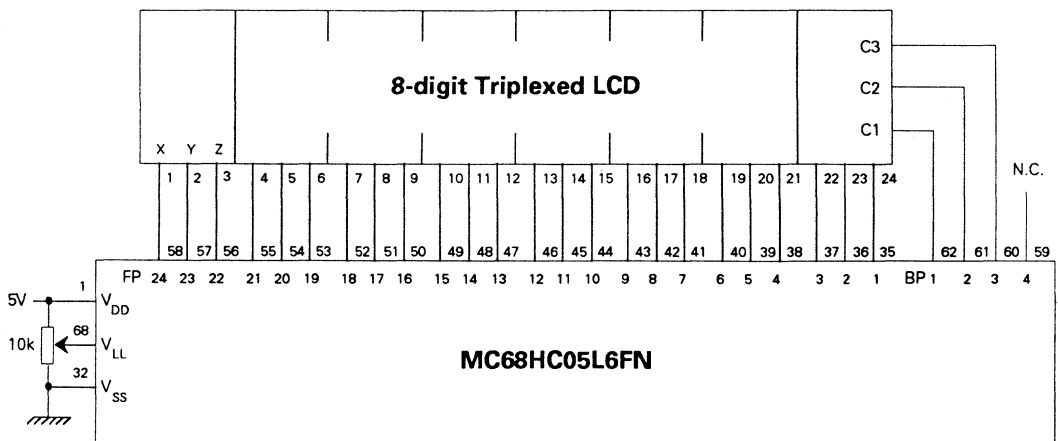


Figure 3. MC68HC05L6 with a 3-backplane LCD

LISTING 2

```

1
2
3
4
5
6
7
8
9
10 00000009      LADD EQU $0009      LCD ADDRESS REGISTER
11 00000008      LDAT EQU $0008      LCD DATA REGISTER
12
13
14                ORG $0050
15
16 00000050      Q      RMB 8      DISPLAY REGISTER
17 00000058      W1     RMB 1
18 00000059      W2     RMB 1
19
20
21                ORG $0100
22
23
24
25
26
27
28
29 00000100 acca  L6TAB FCB $AC,$CA      0
30 00000102 00c0      FCB $00,$C0      1
31 00000104 e48e      FCB $E4,$8E      2
32 00000106 e0ce      FCB $E0,$CE      3
33 00000108 48c4      FCB $48,$C4      4
34 0000010a e84e      FCB $E8,$4E      5
35 0000010c ec4e      FCB $EC,$4E      6
36 0000010e 80c8      FCB $80,$C8      7
37 00000110 ecce      FCB $EC,$CE      8
38 00000112 e8ce      FCB $E8,$CE      9
39 00000114 4004      FCB $40,$04      - (A: CC CC)
40 00000116 ec0e      FCB $EC,$0E      E (B: 6C 46)
41 00000118 4cc4      FCB $4C,$C4      H (C: AC 0A)
42 0000011a 2c02      FCB $2C,$02      L (D: 64 C6)
43 0000011c cc8c      FCB $CC,$8C      P (E: EC 0E)
44 0000011e 0000      FCB $00,$00      (F: CC 0C)

```

```

60
61
62
63
64
65
66
67
68
69
70
71
72
73 0000120 3f59 START CLR W2 Initialise digit pointer.
74 0000122 a680 LDA #580 First write to $09: Bus/LCD ratio = 256.
75 0000124 b709 STA LADD 4-backplane, fast charge enabled.
76 0000126 be59 L60P LDX W2
77 0000128 e650 LDA Q,X Get HEX data.
78 000012a a40f AND #50F Only lower nibble is relevant.
79 000012c 48 LSLA x 2 (two bytes per digit in table).
80 000012d 97 TAX
81 000012e d60100 LDA L6TAB,X Get first byte from segment table.
82 0000131 b708 STA LDAT Send it to LCD data latch.
83 0000133 3c09 INC LADD Keep LCD on, move to next latch.
84 0000135 d60101 LDA L6TAB+1,X Get second byte of segment data.
85 0000138 44 LSRA From this byte only the
86 0000139 44 LSRA upper nibble is relevant, lower
87 000013a 44 LSRA nibble is lost as upper nibble
88 000013b 44 LSRA is shifted down.
89 000013c b758 STA W1 Save nibble, to be combined with first
90 000013e 3c59 INC W2 nibble of next digit.
91 0000140 be59 LDX W2 Address of next digit in Q.
92 0000142 e650 LDA Q,X Get HEX data.
93 0000144 a40f AND #50F only lower nibble is relevant.
94 0000146 48 LSLA x 2 (two bytes per digit in table).
95 0000147 97 TAX
96 0000148 d60100 LDA L6TAB,X Get first byte from segment table.
97 000014b 48 LSLA From this byte only the
98 000014c 48 LSLA lower nibble is relevant, upper
99 000014d 48 LSLA nibble is lost as lower nibble
100 000014e 48 LSLA is shifted up.
101 000014f bb58 ADD W1 Combine nibble with last nibble
102 0000151 b708 STA LDAT of previous digit and send byte to LCD.
103 0000153 3c09 INC LADD Next LCD data latch.
104 0000155 d60101 LDA L6TAB+1,X Get second byte from segment table.
105 0000158 b708 STA LDAT and send it to LCD.
106 000015a 3c09 INC LADD Next latch (three per loop).
107 000015c 3c59 INC W2 Next digit (two per loop).
108 000015e b609 LDA LADD
109 0000160 a10c CMP #12 Finished ?
110 0000162 26c2 BNE L60P If not, do next two digits.
111 0000164 81 RTS

```

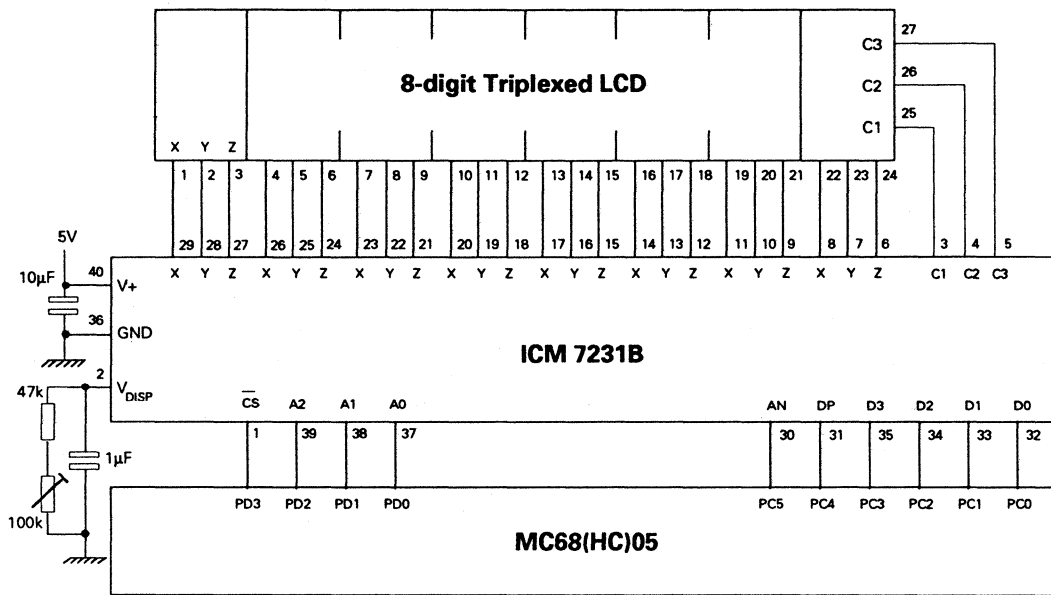



Figure 4. 3-backplane LCD driven by an ICM7321

LISTING 3

```

1
2
3
4
5
6
7
8 00000001
9 00000002
10
11
12
13
14 00000050
15
16
17
18
19
20
21
22
23
24
25 00000100 b601
26 00000102 a4f0
27 00000104 b701
28
29 00000106 ae08
30
31 00000108 e64f
32 0000010a b702
33 0000010c 1701
34 0000010e 1601
35 00000110 5a
36 00000111 2704
37 00000113 3c01
38 00000115 20f1
39
40 00000117 3f02
41 00000119 81

```

```

*****
*
* Example program using the ICM7231
* driver and a 3-backplane display.
*
*****

PORTD EQU $0001 PORT B DATA
PORTC EQU $0002 PORT C DATA

ORG $0050

Q RMB 8 DISPLAY REGISTER

ORG $0100

*****
*
* Display contents of Q.
*
*****

DISP LDA PORTD CLEAR
AND #$F0 LS NIBBLE OF PORTD
STA PORTD IE DIGIT ADDRESS = 0

LDX #8

AGAIN LDA Q-1,X
STA PORTC
BCLR 3,PORTD LATCH
BSET 3,PORTD DIGIT
DECX
BEQ OUT DONE ?
INC PORTD NO. GOTO NEXT DIGIT
BRA AGAIN

OUT CLR PORTC
RTS

```

FOUR-BACKPLANE DISPLAYS

As mentioned above, the MC68HC05L6 can drive a 4-backplane display with up to 24 frontplanes directly. The resultant 96 pixels could be used to drive 2 digits of an 5x8 dot matrix display, but with this number of segments most applications will use 7-segment or customised displays. A 7-segment display of up to 12 digits can be used. The software required is similar to, and simpler than, that shown for the L6 with a 3-backplane display. When it is required to drive a 4-backplane display using a microprocessor without an LCD drive capability, the MC145000 offers a versatile solution. Up to 6 digits (12 frontplanes) can be driven directly and more can be driven by the addition of one or more of the 18-pin MC145001 expanders, each adding 11 frontplanes. The example shown in figure 5 and listing 4 drives 6 digits, the software being very similar to that shown for the MC144115 single-backplane driver. This is the result of both chips having the same shift-register/latch architecture despite the actual output signals being quite different. The listing also shows a routine using an SCI rather than port lines.

A difference between the MC145000 and the MC144115 is that the MC145000 has no chip-enable input. It can share its data line with other peripherals but must have a dedicated clock so that the controller can supply data independently of other chips.

For applications requiring more than the 12 frontplanes made available by the MC145000, the MC145003/4 may be appropriate. They provide 32 frontplanes for use with a 4-backplane display, allowing up to 128

segments. The MC145003 and MC145004 are identical except for their serial protocol. The MC145004 has an IIC bus interface incorporating the usual acknowledge procedure associated with the IIC standard. The MC145003 is the same, except that there is no acknowledge and hence no associated clock cycle. The incoming data is automatically latched after 128 bits have been received. If, however, it is required that the data be latched at other times, an enable pin is available.

For applications where V_{dd} and V_{lcd} are connected together, the LCD contrast is adjusted by adjusting V_{dd}. If the data is coming from a chip with a higher supply voltage, the input pins may go higher than the supply voltage of the MC145003/4. This is allowed for the clock and data pins, but not recommended for the enable pin as its input protection circuitry may clamp the input voltage. It is therefore not advisable to use the enable pin if the MC145003/4 has a different V_{dd} from the chip supplying it with data. In applications not using this pin it can be left floating or tied high.

The example shown in figure 6 does not use the enable pin; the example software sends all 128 bits every time it is executed. The latching is thus performed automatically. The circuit shows 2 6-digit displays, each with 12 frontplanes. Any display or combination of displays with up to 32 frontplanes can be used with the software shown in listing 6, as all 128 bits are always sent. The 6 lines of code (45–50) are commented out for use with the MC145003; they are required for the MC145004.

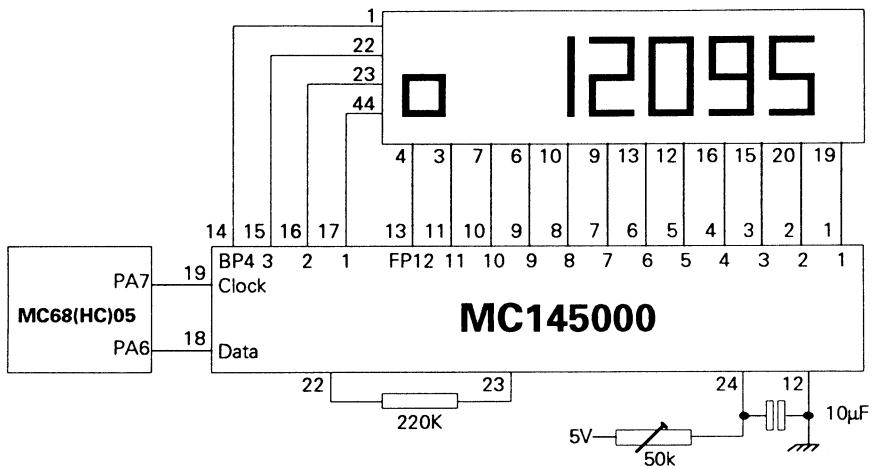


Figure 5. MC145000 driving a 4-backplane LCD

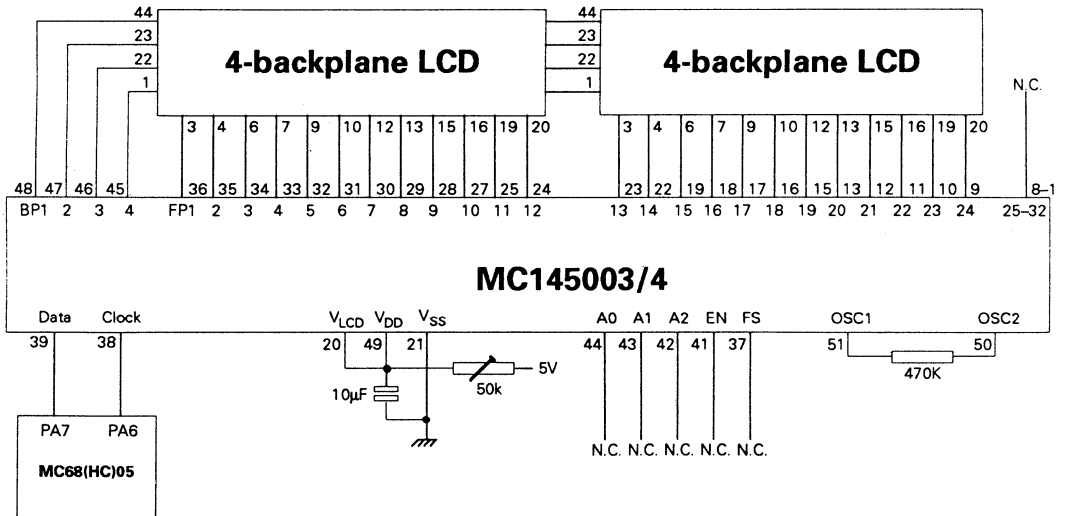


Figure 6. MC145003/4 driving 4-backplane LCDs

LISTING 4

```

1
2
3
4
5
6
7
8
9 00000000          PORTA EQU $00      PORT A DATA
10 0000000d         BAUD EQU $00      SCI BAUD RATE REGISTER
11 0000000e         SCR1 EQU $0E      " CONTROL REG. No. 1
12 0000000f         SCR2 EQU $0F      " " " " 2
13 00000010         SCSR EQU $10     " STATUS "
14 00000011         SDAT EQU $11     " DATA "
15
16                   ORG $0050
17
18 00000050          R      RMB 6      WORKING NUMBER
19
20 00000056          W2     RMB 1      POSITION OF LSB
21 00000057          W3     RMB 1
22 00000058          W4     RMB 1
23 00000059          W5     RMB 1
24 0000005a          W6     RMB 1      POSITION OF MSB
25
26                   ORG $1000
27
28
29
30
31
32
33
34
35
36 00001000 a605          DISP LDA #s05
37 00001002 b758          STA W4
38 00001004 be56          LDX W2      LSB
39 00001006 f6           D3     LDA 0,X
40 00001007 bf59          STX W5
41 00001009 97           TAX
42 0000100a d61047        LDA STABL,X  FIND 7 SEGMENT CODE
43 0000100d be58          LDX W4
44 0000100f e750          STA R,X    PUT IN DISPLAY TABLE
45 00001011 3a58          DEC W4
46 00001013 be59          LDX W5
47 00001015 5a          DECX
48 00001016 b35a          CPX W6    FINISHED ?
49 00001018 26ec          BNE D3

```

```

60
61
62
63
64
65
66
67
68
69
70 0000101a ae05      OUTT  LDX  #5      SEND DISPLAY TABLE TO 144115/145000
71 0000101c e650      DISCHR LDA  R,X
72 0000101e bf57      DISPLY STX  W3      SAVE INDEX
73 00001020 1d00      BCLR  6,PORTA  CLEAR DATA
74 00001022 ae08      LDX  #8
75 00001024 44      DIS1  LSRA      SET UP
76 00001025 2402      BCC  DIS2      BIT OF
77 00001027 1c00      BSET  6,PORTA  ACCUMULATOR
78 00001029 1e00      DIS2  BSET  7,PORTA  CLOCK
79 0000102b 1f00      BCLR  7,PORTA  IT
80 0000102d 1d00      BCLR  6,PORTA  CLEAR DATA
81 0000102f 5a      DECX      COMPLETE ?
82 00001030 26f2      BNE  DIS1      NO
83 00001032 be57      LDX  W3      RESTORE INDEX
84 00001034 5a      DECX
85 00001035 2ae5      BPL  DISCHR
86
87
88
89
90
91
92
93 00001037 ae05      MORE  LDX  #5      INITIALISE X
94 00001039 e650      LDA  R,X      FETCH DIGIT
95 0000103b 0f10fd      BRCLR 7,SCSR,*  WAIT UNTIL TDRE = 1
96 0000103e b711      STA  SDAT     WRITE IT TO SCI TX REG.
97 00001040 5a      DECX      NEXT DIGIT
98 00001041 2af6      BPL  MORE     DONE ?
99 00001043 0d10fd      BRCLR 6,SCSR,*  WAIT UNTIL TC=1
100 00001046 81      RTS
101
102 00001047 eb      STABL FCB  $EB  0  SEGMENT
103 00001048 60      FCB  $60  1
104 00001049 c7      FCB  $C7  2  CODES
105 0000104a e5      FCB  $E5  3
106 0000104b 6c      FCB  $6C  4  FOR THE
107 0000104c ad      FCB  $AD  5
108 0000104d af      FCB  $AF  6  MC145000/MC144115
109 0000104e e0      FCB  $E0  7
110 0000104f ef      FCB  $EF  8  LCD DRIVER
111 00001050 ed      FCB  $ED  9

```

```

*****
*
*   The second part of the display routine
*   sends the 48 bits required by the
*   display driver. For comparison two
*   routines are included, one using port A
*   lines and a second using the SCI.
*
*****

```

```

*****
*
*   SCI LCD driver interface.
*
*****

```

LISTING 5

```

1
2
3
4
5
6
7 00000002      IICP EQU $02          PORTC
8 00000006      IICDD EQU $06         PORTCD
9 00000006      SCL EQU $06          IIC - clock line
10 00000007     SDA EQU $07          IIC - data line
11 00000040     DIN EQU $40          INPUT DATA
12 000000c0     DOUT EQU $c0         OUTPUT DATA
13
14
15
16 00000050      W1 RMB 1
17 00000051     DPNT RMB 1          IIC WRITE POINTER
18 00000052     ADDR RMB 1          IIC ADDRESS
19 00000053     IIC RMB 16         IIC BUFFER (128 BITS)
20
21
22
23 00000b00 a67e      START LDA #$7E
24 00000b02 b752      STA ADDR
25 00000b04 a611      LDA #17
26 00000b06 b750      STA W1
27 00000b08 ae52      LDX #ADDR
28 00000b0a bf51      STX DPNT
29 00000b0c 1f02      SEND2 BCLR SDA,IICP          START CONDITION
30 00000b0e 1d02      BCLR SCL,IICP             DATA GOES LOW WHILE CLOCK HIGH
31
32 00000b10 be51      SLOOP LDX DPNT           DATA BUFFER POINTER
33 00000b12 f6        LDA 0,X                 GET A BYTE
34 00000b13 ae08      LDX #8                  8 BITS TO SHIFT
35 00000b15 49        SLOP ROLA
36 00000b16 2402      BCC DZERO              BIT = 0 ?
37 00000b18 1e02      BSET SDA,IICP          NO, BIT = 1
38 00000b1a 1c02      DZERO BSET SCL,IICP      CLOCK HIGH
39 00000b1c 1d02      BCLR SCL,IICP          CLOCK LOW
40 00000b1e 1f02      BCLR SDA,IICP          DATA LOW
41 00000b20 5a        DECX
42 00000b21 26f2      BNE SLOP
43
44
45
46
47
48
49
50 00000b23 3c51      * LDA #DIN             DATA LINE AN INPUT
51 00000b25 3a50      * STA IICDD
52 00000b27 26e7      * BSET SCL,IICP       CLOCK
53
54
55
56 00000b29 1c02      * BCLR SCL,IICP      ACKNOWLEDGE BIT
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

8/16-BACKPLANE DISPLAYS

For dot-matrix displays, 8 or 16 backplanes are common. This is the result of the large number of pixels required. A compromise between pin-count and contrast is made to decide the number of backplanes. The minimum pin requirement for a display with N segments would require the number of backplanes to be the square root of N, but with typical requirements of many hundred or thousands of pixels this is not practical as the resultant contrast would not be acceptable. Typical compromises are 8 or 16 backplanes, as this gives acceptable contrast and fits in conveniently with the 8x5 dot-matrix format commonly used for this type of display.

The MC68HC05L7 and L9 are designed to directly drive this type of display. The L7 has 16 backplanes and 60 frontplanes allowing it to drive up to 960 pixels or 24 8x5 dot matrix digits (12 with x 8 multiplexing). The L9 has only 40 frontplanes (16 8x5 digits) but is

capable of being used with MC68HC68L9 LCD drive expanders. Each MC68HC68L9, up to 3 of which may be added, contributes 55 frontplanes. An L9 and 3 expanders has thus 205 frontplanes allowing then to drive up to 3280 pixels or 82 8x5 dot matrix digits.

The display RAM contains a 5-bit word for each row of dots in the 8x5 format; thus, 8 locations are used for each digit, allowing easy addressing. The RAM corresponding to the digits driven by expanders is contained in the expanders, but appears in the L9's memory map as the data and address buses from the L9 to external memory are also used by the MC68HC68L9s.

Application note ANHK10/D shows an application using the L9 and also describes a method of extending the display size beyond that normally available using this device.

APPLICATION NOTES

The following application notes give complete applications using the type(s) of display indicated.

ANE404 An extended MC146805E2 CBUG05 system using the MC68HC25.
MC145000-driven 4-backplane, 6-digit and ICM7231B-driven 3-backplane, 8-digit display.

ANE416 MC68HC05B4 Radio Synthesizer.
MC145000-driven 4-backplane, 6-digit and MC144115-driven 1-backplane, 6-digit display.

ANE425 Use of the MC68HC68T1 RTC with M6805 Microprocessors.
ICM7231B-driven 3-backplane, 8-digit display.

ANHK10 The summary of the MC68HC05L9 Micro. App. Demo. Board.
MC68HC05L9/MC68HC68L9-driven dot matrix display.

DRIVER CHIPS

The following list shows some LCD driver devices. They are most suitable for 7-segment, 16-segment and custom displays. The 7SD column shows how many 7-segment digits each device can drive. With the possible exceptions of the MC145000/1 and the MC145003/4, they are not generally suitable for dot-matrix displays which have 35-40 segments per digit.

Device	Back	Front	7SD	Drive	Expan.	Pins	
MC14543	1	7	1	parallel	parallel	16	BCD
MC14544	1	7	1	parallel	parallel	18	ripple blank
MC144115	1	16	2	3-line	yes	24	
MC144117	2	16	4	3-line	no	24	
MC145000	4	12	6	2-line	MC145001	24	
MC145001	(4)	11	5.5	2-line	n/a	18	expander
MC145003	4	32	16	2- or 3-line	parallel	52	2- or 3-line
MC145004	4	32	16	2-line	parallel	52	IIC
MC145453	1	33	4	2-line	parallel	40	also 44-pin
ICM7231	3	24	8	parallel	no	40	BCD

MCM2814 Gang-programmer using an MC68HC805B6

By Peter Topping
MCU Applications Group
Motorola Ltd, East Kilbride

INTRODUCTION

Non-volatile memories of the type MCM2814 are widely used in consumer equipment to store semi-permanent, user-definable information. One of the most common applications is TVs. In a TV the NVM is used to store the channel number or frequency associated with each program number and may also store other information about the program (for example, fine-tuning and transmission standard). The NVM will also contain the optimum settings for the sound and picture analogue values. In some sets other data may be stored, for example, user-defined names for some or all of the available channels. In a production environment the initial loading of this type of information can be done quickly by copying an existing NVM. This application note describes a programmer, shown in Figure 1, which can perform this function. In four seconds it can fully program 8 MCM2814s in parallel and verify them individually. The programmer is controlled by an MC68HC805B6 microprocessor.

The B6 is ideally suited to this application as it has a 256 byte NVEEPROM, the same size as the MCM2814. The contents of a master MCM2814 can thus be loaded into the B6 and will remain there until a change of data is required. Both the B6 and the MCM2814 have a byte in which some bits are dedicated to data protection so, in fact, only 255 bytes are used.

PRINCIPLE OF OPERATION

The programmer has been designed with an emphasis on ease of use and consequently has as few controls as possible. The only control which is used regularly is the "RUN" button S1. When it is pressed, the NVMs are powered-up and programmer operation starts. When it is released, power is disconnected from the NVMs but remains on within the programmer so that the LED indicators remain.

Three different operations are available.

The first requirement is to load data into the programmer. This procedure is selected by pressing and holding button S3, and started by pressing, and holding, the button S1. The contents of a master NVM in socket #0 will be loaded into the programmer; this takes about 12 seconds. The MC68HC805B6 in the programmer retains this data in its non-volatile EEPROM. During loading both LEDs at socket #0 are on and the rest are off. Once the B6's EEPROM is written it is verified against the NVM in socket #0, a green LED indicating a pass and a red LED a fail. Sockets 1-7 should be empty during this procedure but if they contain an NVM they will also be checked. An attempt to perform this routine without an NVM in socket #0 will not destroy the data in the programmer as the software checks for an IIC acknowledge before overwriting the contents of the EEPROM within the MCM68HC05B6. If S3 is pressed, the position of the slide switch S2 (program and verify or verify only) is not relevant. As S1 supplies power to the NVM sockets, it is important that it is held until the procedure is complete. Reliable results will not be obtained if S1 is released before the verification has finished.

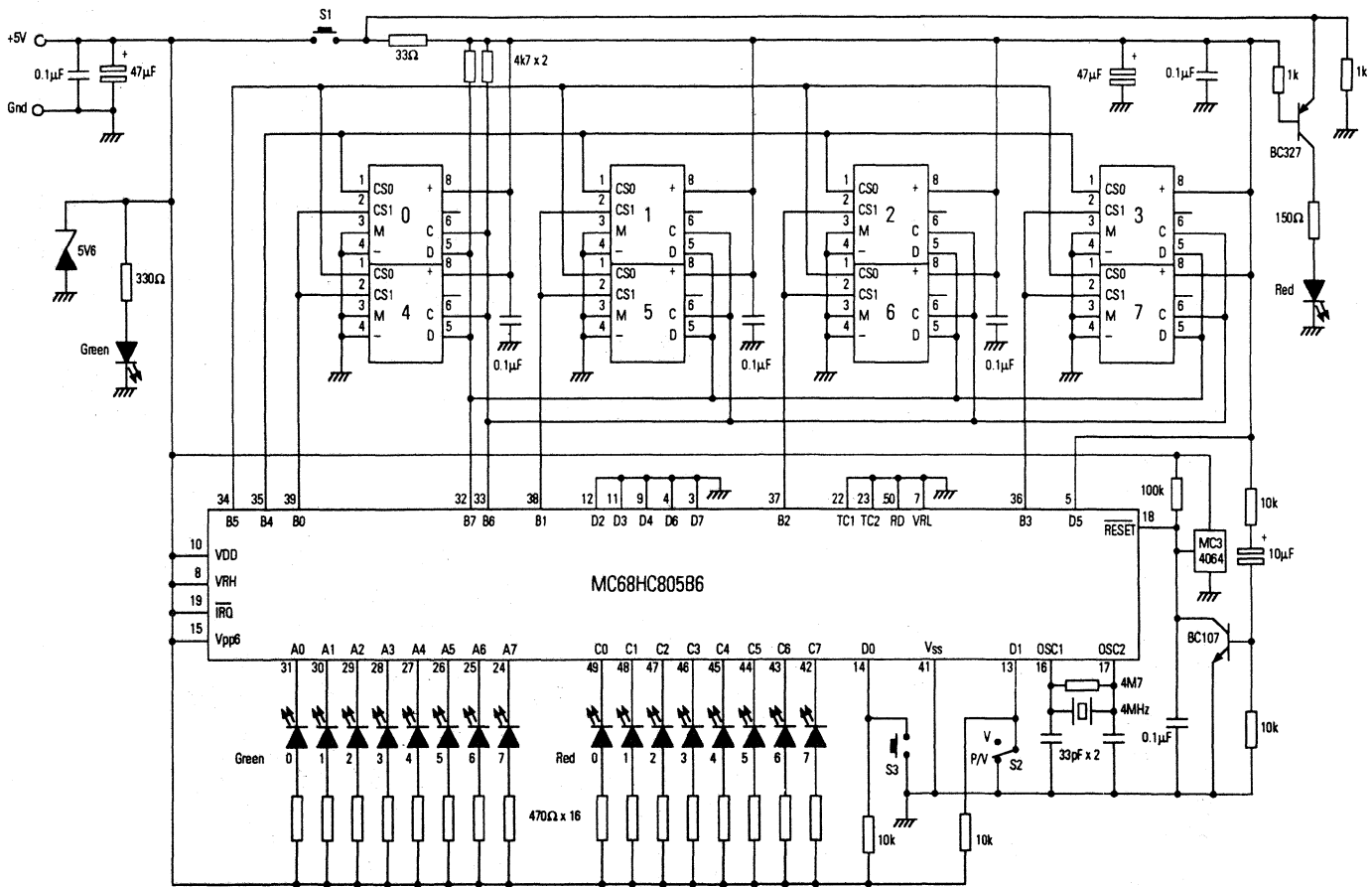


Figure 1. MCM2814 Gang-programmer

The main programming sequence is selected by putting S2 in position P/V (program and verify). The NVMs should be placed in the sockets and button S1 pressed, and held, until programming and verification are complete (4 seconds). During programming, both red and green LEDs are on. The procedure clears the write-protect bytes and programs the NVMs in parallel using the data stored within the B6 in the programmer. It then verifies the NVMs individually, the results being shown on the LEDs (green for pass, red for fail and neither for no acknowledge).

Running only the verify routine can be selected by placing S2 in position "V". The procedure is otherwise similar to programming. S1 should be held in until verification is finished (up to 1 second depending on results).

CIRCUIT

The NVMs are arranged in a matrix using their IIC chip-selects (pins 1 and 2). These pins configure the two least significant bits (1 and 2) of the recognised IIC address. Bit 0 determines if the device is receiving or transmitting. The software uses only the address 1010000x and so an NVM will only be addressed if both its chip-select pins are low. The matrix thus allows each NVM to be addressed individually by I/O lines using common software. Six port B I/O lines (0-5) are used for this purpose.

The NVMs (and the IIC pullups) are only powered up when S1 is closed, allowing them to be inserted and removed when they are powered down. When S1 is pressed, 5 volts is also applied to the RC circuit connected to the base of the BC107 NPN transistor. This supplies a base current to the transistor while the 10 μ F capacitor is charging. During this transient the transistor is on and the MC68HC05B6 is in reset. The program starts when the capacitor is charged and the transistor switches off. The MC34064 also connected to the reset pin ensures that the microprocessor is held in reset if the supply is below 4.75V. This will be the case during power-up and power-down. This is advisable in any system with EEPROM in order to prevent the possibility of it being corrupted during the rise or fall of Vdd.

The 33 ohm resistor in series with the supply to the NVMs limits the current if a device is plugged in the wrong way round. If this happens, the BC327 PNP transistor is turned on and the RED fault LED lit. The actual voltage on the VDD pins of the NVMs is monitored by bit 5 on port D. Thus the microprocessor can tell if there is a short or a device inserted the wrong way round. If this is the case the software aborts any attempt to program or verify NVMs.

Ports A and C are used for the pass and fail LEDs respectively, while the input-only port D has two pins used to read the positions of S2 and S3.

SOFTWARE

On reset, the program initialises the ports, IIC addresses and the RAM location (STAT) which is used for status flags. The state of bit 0 on port D is then checked. This will be low unless the button (S3), which selects the loading of data into the programmer, is pressed. If it is low, either program and verify or verify only is executed according to the level of bit 1 on port D which is connected to switch S2.

The programming routine (PROG) switches on all the LEDs, enables all the NVMs and clears their write-protect by writing \$00 to address \$FF. A loop which reads a byte from the B6 within the programmer and writes it to the NVMs is then performed 255 times. Writing to the NVMs is carried out without checking for an acknowledge, so the full procedure will take place even if all the NVM sockets are empty. This is the only routine which does not check for an acknowledge.

Only 255 bytes are transferred, as the last byte in the MCM2814 is reserved for the write-protect status. The B6 also has a reserved byte, but it is the first byte, so the correspondence of the addresses between the B6 and the NVMs is offset by one.

The verify routine (VERF) switches off all the LEDs and compares the data in the B6 with the data in each of the NVMs. It immediately fails a socket which does not return an IIC acknowledge. The RAM location COUNT contains the number of the socket being checked and is used to switch on the appropriate LED once each verify is complete.

The NVM sockets are arranged in a 2 x 4 matrix using their two IIC chip selects. The IIC address used is always \$A1 (\$A0 for writing) and thus an MCM2814 will only respond if both its chip-selects are low. During the programming routine all lines in the matrix are low, but during verification each chip is individually selected by a low on one row and one column. The required value to be sent to port B to achieve this is derived from the value in COUNT using the table TAB1.

If the LOAD routine is selected it performs a loop which does the opposite of the program loop. It transfers the contents of an MCM2814 into the EEPROM of the MC68HC805B6 within the programmer. It only looks at socket #0 and checks for an acknowledge before it proceeds to over-write the data in the B6.

The three routines WRT, RDALL and WRTAL were used during de-bug, are self-explanatory, and have been left in the program for interest and for future use if required. They are not used by the MC68HC805B6 but can be used if the program is run on a development system (eg an M68HC05EVM).

The IIC READ, SEND and WRITE routines are self contained and, by using ordinary I/O lines, could be used on any 68(HC)05 microprocessor. The READ sub-routine reads one byte using the IIC address in ADDR and the sub-address in SUBARD. The commented out lines (382-385) provide a simple method of also reading a second byte. Data is returned in IOBUF (and IOBUF+1 in the case of a 2-byte read). Clearly, the number of bytes can be increased by including more in-line code or by executing a loop.

The SEND sub-routine writes data on the IIC bus at IIC address ADDR starting at the address in RAM contained in the index register. The number of bytes sent is determined by the value stored the accumulator. If, as in this example, a sub-address has to be sent, then it should be in the RAM location pointed to by the index register with subsequent locations containing the data. SEND is suitable for most IIC peripherals. The WRITE subroutine constitutes a SEND followed by a delay and a READ. This is what is required to write a byte to an MCM2814, EEPROM programming is started by an IIC write and continues until the chip is addressed again by READ.

The IIC routines use simple BCLR and BSET instructions on the data registers so that an active high level is generated. This is not strictly what should be done on an IIC bus but is OK in the case where there are no other possible masters in the system. A passive high can be obtained by keeping the data bit of the I/O lines used for the IIC bus at a zero and using BCLR and BSET instructions on the corresponding data direction bits. If this is done care must be taken to ensure that these zeros are not lost by any other read-modify-write operations carried out on the data register. A BCLR or BSET on another bit on the same port will read the whole port, modify the required bit and then write the whole port. The data bit on an IIC line which was pulled high by the external pull-up would thus become a one. This would cause a malfunction the next time the corresponding DDR bit was set to a one, as the pin would output a high instead of a low. This can be avoided by using other types of instructions (eg STA) or by making sure the relevant data bit is a zero before enabling an output.

```

0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014 0000
0015 0001
0016 0002
0017 0003
0018 0004
0019 0005
0020 0006
0021 0007
0022 0100
0023 0f00
0024
0025
0026
0027
0028
0029
0030
0031 0050
0032
0033 0050
0034 0051
0035 0052
0036 0053
0037
0038 0055
0039 0056
0040 0057
0041 0058
0042 0059
0043 005a
0044 005b
0045 005c
0046
0047 005d
0048
0049
0050
0051
0052 005e
0053 00f8
0054 00ff
0055
0056 0800

NVM.A55
0058
0059
0060
0061
0062
0063
0064
0065 0800 a6 c0
0066 0802 b7 01
0067 0804 a6 ff
0068 0806 b7 00
0069 0808 b7 02
0070 080a b7 04

*****
*
*           MC68HC05B6 controlled MCM44182 programmer.
*
* This software was developed by Motorola Ltd. for demonstration purposes.
* No liability can be accepted for its use in any specific application.
* Original software copyright Motorola - all rights reserved.
*
*           P. Topping                               14th May '91
*
*****

PORTA EQU $00          PORT A ADDRESS
PORTB EQU $01          " B "
PORTC EQU $02          " C "
PORTD EQU $03          " D "
PORTAD EQU $04         PORT A DATA DIRECTION REG.
PORTBD EQU $05         " B " " "
PORTCD EQU $06         " C " " "
EECTL EQU $07          EEPROM CONTROL REGISTER
E2PR EQU $0100         EEPROM
BUFF EQU $0F00         DEBUG USE ONLY

*****
*
*           RAM allocation.
*
*****

          ORG          $0050

ADDR RMB 1           IIC ADDRESS
DPNT RMB 1           IIC DATA POINTER
SUBADR RMB 1         IIC SUB-ADDRESS
IOBUF RMB 2          IIC BUFFER

W1 RMB 1             W
W2 RMB 1             O
W3 RMB 1             R
W4 RMB 1             K
W5 RMB 1             I
W6 RMB 1             N
W7 RMB 1             G
COUNT RMB 1         LOOP COUNTER

STAT RMB 1           STATUS BYTE :-
*                   0: not used
*                   1: IIC R/W 1:READ, 0:WRITE
*                   2: ACKNOWLEDGE OK

          RMB 154     not used
STACK RMB 7          8 BYTES USED (0 INTERRUPTS
SP RMB 1             AND 4 NESTED SUBROUTINES)

          ORG          $0800

page 2

*****
*
*           Main program.
*
*****

START LDA #$C0
      STA PORTB      CONTROL LINES LOW
      LDA #$FF
      STA PORTA      LEDS OFF
      STA PORTC      " "
      STA PORTAD     ALL OUT - GREEN LEDS

```

```

0071 080c b7 06          STA  PORTCD          ALL OUT - RED LEDS
0072 080e a6 3f          LDA  #$3F            6,7 IN - IIC BUS
0073 0810 b7 05          STA  PORTBD          0-5 OUT - NVM SELECTION
0074 0812 ae 64          LDX  #100            WAIT 230ms FOR MCM2814 SUPPLY
0075 0814 cd 0a 35       JSR  PN              TO SETTLE
0076 0817 0a 03 02       BRSET 5,PORTD,CONT  NVMs POWERED UP ?
0077 081a 20 fe          BRA  *               NO, DO NOT PROCEED
0078
0079 081c a6 a0          CONT LDA  #$A0        YES, INITIALISE IIC ADDRESS
0080 081e b7 50          STA  ADDR
0081 0820 3f 07          CLR  EECTL          READ FROM EEPROM (B6)
0082 0822 3f 52          CLR  SUBADR
0083 0824 3f 5d          CLR  STAT
0084 0826 5f            CLRX
0085 0827 00 03 03       BRSET 0,PORTD,SKLD  LOAD B6 EEPROM ?
0086 082a cc 09 08       JMP  LOAD           YES
0087 082d 02 03 2e       SKLD BRSET 1,PORTD,VERF NO, VERIFY ONLY ?
0088
0089
0090
0091
0092
0093
0094
0095 0830 3f 00          PROG CLR  PORTA        GREEN LEDS ON
0096 0832 3f 02          CLR  PORTC          RED LEDS ON
0097 0834 a6 c0          LDA  #$C0
0098 0836 b7 01          STA  PORTB          ENABLE ALL NVMS
0099 0838 a6 ff          LDA  #$FF
0100 083a b7 52          STA  SUBADR         SUBADR = $FF
0101 083c cd 09 92       JSR  READ           CLEAR POR WRITE PROTECT
0102 083f 4f            CLRA                DATA = $00
0103 0840 ad 0e          BSR  WR1            CLEAR WRITE PROTECT BYTE
0104 0842 3f 52          CLR  SUBADR
0105 0844 5f            CLRX
0106
0107 0845 d6 01 01       OLOOP LDA  E2PR+1,X  GET BYTE AND
0108 0848 ad 06          BSR  WR1            SEND IT TO NVM
0109 084a a3 ff          CPX  #255           DONE ?
0110 084c 26 f7          BNE  OLOOP
0111
0112 084e 20 0e          BRA  VERF           YES, VERIFY
0113
0114 0850 b7 53          WR1  STA  IOBUF
0115 0852 a6 02          LDA  #2             No. BYTES TO SEND (INC. SUB-ADDRESS)
0116 0854 ae 52          LDX  #SUBADR
0117 0856 cd 09 89       JSR  WRITE           IIC WRITE
0118 0859 3c 52          INC  SUBADR         NEXT LOCATION
0119 085b be 52          LDX  SUBADR
0120 085d 81            RTS

```

NVM.AS5

page 3

```

0122
0123
0124
0125
0126
0127
0128
0129
0130
0131 085e a6 ff          VERF LDA  #$FF
0132 0860 b7 02          STA  PORTC          ALL LEDS OFF
0133 0862 b7 00          STA  PORTA
0134 0864 3f 5c          CLR  COUNT          START AT SOCKET #0
0135
0136 0866 be 5c          VLP  LDX  COUNT
0137 0868 d6 09 00       LDA  TAB1,X         SELECT NVM
0138 086b b7 01          STA  PORTB         ACC. TO COUNT
0139 086d ad 73          BSR  VRF1
0140 086f 24 35          BCC  P0

```

```

*****
*
*          B6 EEPROM ($101-$1FF) -> 8 NVms ($00 - $FE).
*
*****

```

```

*****
*
*          B6 EEPROM ($0101-$01FF) v. NVms ($00 - $FE).
*
*****

```

```

0141
0142 0871 04 5d 30      F0      BRSET   2,STAT,FINP      IF NO ACK. THEN NO LED
0143
0144 0874 be 5c          LDX      COUNT
0145 0876 26 02          BNE      F1
0146 0878 11 02          BCLR     0,PORTC
0147 087a a3 01          F1      CPX      #1
0148 087c 26 02          BNE      F2
0149 087e 13 02          BCLR     1,PORTC
0150 0880 a3 02          F2      CPX      #2
0151 0882 26 02          BNE      F3
0152 0884 15 02          BCLR     2,PORTC
0153 0886 a3 03          F3      CPX      #3
0154 0888 26 02          BNE      F4
0155 088a 17 02          BCLR     3,PORTC
0156 088c a3 04          F4      CPX      #4
0157 088e 26 02          BNE      F5
0158 0890 19 02          BCLR     4,PORTC
0159 0892 a3 05          F5      CPX      #5
0160 0894 26 02          BNE      F6
0161 0896 1b 02          BCLR     5,PORTC
0162 0898 a3 06          F6      CPX      #6
0163 089a 26 02          BNE      F7
0164 089c 1d 02          BCLR     6,PORTC
0165 089e a3 07          F7      CPX      #7
0166 08a0 26 02          BNE      FINP
0167 08a2 1f 02          BCLR     7,PORTC
0168
0169 08a4 20 30          FINP     BRA      OUT
0170

```

page 4

NVM.AS5

```

0172
0173
0174
0175
0176
0177
0178
0179
0180
0181 08a6 be 5c          P0      LDX      COUNT
0182 08a8 26 02          BNE      P1
0183 08aa 11 00          BCLR     0,PORTA
0184 08ac a3 01          P1      CPX      #1
0185 08ae 26 02          BNE      P2
0186 08b0 13 00          BCLR     1,PORTA
0187 08b2 a3 02          P2      CPX      #2
0188 08b4 26 02          BNE      P3
0189 08b6 15 00          BCLR     2,PORTA
0190 08b8 a3 03          P3      CPX      #3
0191 08ba 26 02          BNE      P4
0192 08bc 17 00          BCLR     3,PORTA
0193 08be a3 04          P4      CPX      #4
0194 08c0 26 02          BNE      P5
0195 08c2 19 00          BCLR     4,PORTA
0196 08c4 a3 05          P5      CPX      #5
0197 08c6 26 02          BNE      P6
0198 08c8 1b 00          BCLR     5,PORTA
0199 08ca a3 06          P6      CPX      #6
0200 08cc 26 02          BNE      P7
0201 08ce 1d 00          BCLR     6,PORTA
0202 08d0 a3 07          P7      CPX      #7
0203 08d2 26 02          BNE      OUT
0204 08d4 1f 00          BCLR     7,PORTA
0205
0206 08d6 3c 5c          OUT     INC      COUNT      NEXT SOCKET
0207 08d8 b6 5c          LDA      COUNT
0208 08da a1 07          CMP      #7
0209 08dc 23 88          BLS      VLP      FINISHED ?
0210 08de 3f 01          CLR      PORTB     YES, ALL CONTROL LINE LOW

```



```

0211 08e0 20 fe          BRA      *          AND WAIT HERE
0212
0213 08e2 3f 52          VRF1    CLR      SUBADR
0214 08e4 5f              CLRX
0215 08e5 cd 09 92          VLOOP   JSR      READ          GET A BYTE FROM NVM
0216 08e8 04 5d 13          BRSET   2,STAT,FAIL      ACKNOWLEDGE OK ?
0217 08eb b6 53              LDA     IOBUF          YES, CHECK DATA
0218 08ed be 52              LDX     SUBADR
0219 08ef d1 01 01          CMP     E2PR+1,X        COMPARE WITH B6 EEPROM BYTE
0220 08f2 26 0a              BNE     FAIL           SAME ?
0221 08f4 3c 52              INC     SUBADR          YES, CONTINUE
0222 08f6 be 52              LDX     SUBADR
0223 08f8 a3 ff              CPX     #255           FINISHED (255 BYTES) ?
0224 08fa 26 e9              BNE     VLOOP
0225 08fc 98                CLC
0226 08fd 81                RTS          PASS, EXIT WITH C CLEAR
0227 08fe 99                FAIL      SEC          FAIL, EXIT WITH C SET
0228 08ff 81                RTS
0229
0230 0900 ee ed eb e7 de dd TAB1  FCB     $EE,$ED,$EB,$E7,$DE,$DD,$DB,$D7
      db d7

```

NVM.AS5

page 5

```

0232
0233 *****
0234 *
0235 *          NVM (#0, $00-$FE) -> B6 EEPROM ($101-$1FF). *
0236 *
0237 *****
0238
0239 0908 a6 ee          LOAD   LDA     #$EE          SELECT POSITION 0
0240 090a b7 01          STA     PORTB
0241 090c a6 fe          LDA     #$FE          POSITION 0 LEDS ON
0242 090e b7 00          STA     PORTA
0243 0910 b7 02          STA     PORTC
0244
0245 0912 cd 09 92          ILOOP  JSR      READ          GET BYTE FROM NVM
0246 0915 04 5d 17          BRSET   2,STAT,SKIP      ACKNOWLEDGE OK ?
0247 0918 b6 53              LDA     IOBUF          YES, GET DATA
0248 091a 12 07          BSET   1,EECTL          SET E1LAT
0249 091c 14 07          BSET   2,EECTL          SET E1ERA
0250 091e ad 12          BSR     DOIT           ERASE BYTE
0251 0920 cd 0a 2d          JSR     P10            WAIT 9.2 mS
0252 0923 12 07          BSET   1,EECTL          SET E1LAT TO WRITE BYTE
0253 0925 ad 0b          BSR     DOIT
0254 0927 3c 52          INC     SUBADR
0255 0929 be 52          LDX     SUBADR
0256 092b a3 ff          CPX     #255
0257 092d 26 e3          BNE     ILOOP
0258
0259 092f cc 08 5e          SKIP   JMP     VERF
0260
0261 0932 be 52          DOIT   LDX     SUBADR          GET ADDRESS
0262 0934 d7 01 01          STA     E2PR+1,X        LATCH DATA
0263 0937 10 07          BSET   0,EECTL          START PROGRAMMING
0264 0939 cd 0a 33          JSR     P20            WAIT 18.4 mS
0265 093c 3f 07          CLR     EECTL          STOP
0266 093e 81          RTS

```

NVM.AS5

page 6

```

0268 *****
0269 *
0270 *          Debug routines for use with EVM, not used *
0271 *          in 805B6. *
0272 *
0273 *          RAM ($F00 - $FFE) -> B6 EEPROM ($101 - $1FF). *
0274 *
0275 *****
0276
0277
0278 093f 3f 52          WRT    CLR     SUBADR

```

```

0279 0941 5f          CLRX
0280
0281 0942 d6 0f 00    LOOP2  LDA    BUFF,X
0282 0945 12 07        BSET   1,EECTL      SET EILAT
0283 0947 14 07        BSET   2,EECTL      SET EIERA
0284 0949 ad e7        BSR    DOIT          ERASE BYTE
0285 094b cd 0a 33     JSR    P20
0286 094e 12 07        BSET   1,EECTL      SET EILAT TO WRITE BYTE
0287 0950 ad e0        BSR    DOIT
0288 0952 3c 52        INC    SUBADR
0289 0954 be 52        LDX   SUBADR
0290 0956 a3 ff        CFX   #255
0291 0958 26 e8        BNE   LOOP2
0292
0293 095a 20 fe        BRA   *
0294
0295
0296
0297
0298
0299
0300
0301 095c 3f 52       RDALL  CLR    SUBADR
0302 095e a6 ee       LDA    #SEE         SELECT POSITION 0
0303 0960 b7 01       STA   PORTB
0304
0305 0962 cd 09 92     RLOOP  JSR    READ
0306 0965 be 52        LDX   SUBADR
0307 0967 b6 53        LDA   IOBUF
0308 0969 d7 0f 00     STA   BUFF,X
0309 096c 3c 52        INC   SUBADR
0310 096e 26 f2        BNE   RLOOP
0311
0312 0970 20 fe        BRA   *
0313
0314
0315
0316
0317
0318
0319
0320 0972 3f 52       WRTAL  CLR    SUBADR
0321 0974 a6 ee       LDA    #SEE         SELECT POSITION 0
0322 0976 b7 01       STA   PORTB
0323 0978 5f          CLRX
0324
0325 0979 d6 0f 00     WLOOP  LDA    BUFF,X
0326 097c cd 08 50     JSR   WR1
0327 097f 26 f8        BNE   WLOOP
0328
0329 0981 20 fe        BRA   *

NVM.AS5                                page 7
0331
0332
0333
0334
0335
0336
0337
0338 0001             IICP   EQU    $01      PORTB
0339 0005             IICDD  EQU    $05      DDRB
0340
0341 0006             SCL   EQU    $06      IIC - clock line
0342 0007             SDA   EQU    $07      IIC - data line
0343 007f             DIN   EQU    $7F      INPUT DATA
0344 00ff             DOUT  EQU    $FF      OUTPUT DATA
0345 003f             OPEN  EQU    $3F      TRI-STATE BOTH
0346
0347 0983 13 5d       SEND  BCLR   1,STAT    WRITE IIC DATA
0348 0985 b7 55       STA   W1             SAVE No. BYTES TO SEND

```

0349	0987	20	0b		BRA	IICBUS		
0350								
0351	0989	13	5d	WRITE	BCLR	1,STAT	WRITE TO NVM	
0352	098b	b7	55		STA	W1	SAVE No. BYTES TO SEND	
0353	098d	ad	05		BSR	IICBUS		
0354	098f	cd	0a	2d	JSR	P10	9.2 mS NVM WRITE TIME	
0355								
0356	0992	12	5d	READ	BSET	1,STAT	READ IIC DATA	
0357								
0358	0994	1d	01	IICBUS	BCLR	SCL, IICP	CLOCK LOW	
0359	0996	1e	01		BSET	SDA, IICP	DATA HIGH	
0360	0998	1c	01		BSET	SCL, IICP	CLOCK HIGH	
0361								
0362	099a	a6	ff		LDA	#DOUT	DRIVE	
0363	099c	b7	05		STA	IICDD	BOTH	
0364								
0365	099e	11	50		BCLR	0, ADDR	RW = 0 ALWAYS WRITE (SUB-ADDRESS)	
0366	09a0	b6	50		LDA	ADDR	SEND CHIP ADDRESS	
0367								
0368	09a2	1f	01	IICD3	BCLR	SDA, IICP	START CONDITION	
0369	09a4	1d	01		BCLR	SCL, IICP	DATA GOES LOW WHILE CLOCK HIGH	
0370	09a6	bf	51		STX	DPNT		
0371	09a8	ad	57		BSR	SHIFT	CHIP ADDRESS OUT	
0372								
0373	09aa	03	5d	40	BRCLR	1,STAT,WRBUS	READ OR WRITE ?	
NVM.A55								page 8
0375								
0376								
0377								
0378								
0379								
0380								
0381								
0382	09ad	b6	52	RDBUS1	LDA	SUBADR		
0383	09af	ad	50		BSR	SHIFT	WRITE SUB-ADDRESS	
0384	09b1	10	50	RDBUSQ	BSET	0, ADDR	SET BIT 0 FOR READ	
0385	09b3	b6	50		LDA	ADDR	CHIP ADDRESS	
0386	09b5	1e	01		BSET	SDA, IICP	BOTH HIGH, NO	
0387	09b7	1c	01		BSET	SCL, IICP	STOP CONDITION	
0388	09b9	1f	01		BCLR	SDA, IICP	START CONDITION	
0389	09bb	1d	01		BCLR	SCL, IICP		
0390	09bd	ad	42		BSR	SHIFT	RE-SEND CHIP ADDRESS	
0391								
0392	09bf	a6	7f	RDBUS3	LDA	#DIN	DATA IN FROM BUS	
0393	09c1	b7	05		STA	IICDD		
0394	09c3	ad	04		BSR	RDB	READ 8 BITS	
0395								
0396				*	BSR	RACKF	DUMMY ACKNOWLEDGE	
0397				*	LDA	IOBUF	MOVE	
0398				*	STA	IOBUF+1	UP	
0399				*	BSR	RDB	AND READ 8 MORE	
0400								
0401	09c5	ad	11		BSR	RACK		
0402	09c7	20	2f		BRA	IICEND		
0403								
0404	09c9	ae	08	RDB	LDX	#8		
0405	09cb	1c	01	RDBUS2	BSET	SCL, IICP	CLOCK HIGH	
0406	09cd	0e	01	00	BRSET	SDA, IICP, *+3	DATA LINE (RESULT IN CARRY)	
0407	09d0	1d	01		BCLR	SCL, IICP	CLOCK LOW	
0408	09d2	39	53		ROL	IOBUF		
0409	09d4	5a			DECX			
0410	09d5	26	f4		BNE	RDBUS2		
0411	09d7	81			RTS			
0412								
0413	09d8	1e	01	RACK	BSET	SDA, IICP	LAST BYTE READ : SDA HIGH	
0414	09da	a6	ff	RA2	LDA	#DOUT	SDA OUT	
0415	09dc	b7	05		STA	IICDD	DUMMY ACKNOWLEDGE	
0416	09de	1c	01		BSET	SCL, IICP	CLOCK	
0417	09e0	1d	01		BCLR	SCL, IICP		
0418	09e2	1f	01		BCLR	SDA, IICP		

```

0419 09e4 a6 7f          LDA    #DIN          SDA IN
0420 09e6 b7 05          STA    IICDD
0421 09e8 81             RTS
0422
0423 09e9 1f 01          RACKF  BCLR    SDA,IICP    ACKN. WITH FOLLOWING BYTE
0424 09eb 20 ed          BRA    RA2

```

NVM.AS5 page 9

```

0426
0427 *****
0428 *
0429 *          Send sub-address and write data onto bus.
0430 *
0431 *****
0432
0433 09ed be 51          WRBUS  LDX    DPNT          DATA BUFFER POINTER
0434 09ef f6           LDA    0,X          DATA
0435 09f0 ad 0f          BSR    SHIFT
0436 09f2 3c 51          INC    DPNT
0437 09f4 3a 55          DEC    W1          No. BYTES
0438 09f6 26 f5          BNE    WRBUS
0439
0440 09f8 1c 01          IICEND BSET    SCL,IICP    STOP CONDITION
0441 09fa 1e 01          BSET  SDA,IICP    DATA GOES HIGH WHILE CLOCK HIGH
0442 09fc a6 3f          LDA    #OPEN      TRI-STATE
0443 09fe b7 05          STA    IICDD
0444 0a00 81             RTS
0445
0446 *****
0447 *
0448 *          Shift out 8 bits and check acknowledge bit.
0449 *
0450 *****
0451
0452 0a01 ae 08          SHIFT  LDX    #8
0453 0a03 49          SHIF1  ROLA
0454 0a04 24 04          BCC    SHIF2      SHIFT OUT 8 BITS 0 ?
0455 0a06 1e 01          BSET  SDA,IICP    NO, DATA = 1
0456 0a08 20 04          BRA    SHIF3
0457 0a0a 1f 01          SHIF2  BCLR   SDA,IICP    DATA = 0
0458 0a0c 20 00          BRA    SHIF3      DELAY
0459 0a0e 1c 01          SHIF3  BSET   SCL,IICP    CLOCK HIGH
0460 0a10 1d 01          BCLR  SCL,IICP    CLOCK LOW
0461 0a12 1f 01          BCLR  SDA,IICP    DATA LOW
0462 0a14 5a          DECX
0463 0a15 26 ec          BNE    SHIF1
0464
0465 0a17 a6 7f          WACK   LDA    #DIN          WRITE ACKNOWLEDGE
0466 0a19 b7 05          STA    IICDD
0467 0a1b 15 5d          BCLR  2,STAT      CLEAR FLAG
0468 0a1d 1c 01          BSET  SCL,IICP    CLOCK
0469
0470 0a1f 0f 01 02        BRCLR  SDA,IICP,ACOK  ACKNOWLEDGE OK ?
0471 0a22 14 5d          BSET  2,STAT      NO, SET FLAG
0472
0473 0a24 1d 01          ACOK   BCLR   SCL,IICP
0474 0a26 1f 01          BCLR  SDA,IICP
0475 0a28 a6 ff          LDA    #DOUT
0476 0a2a b7 05          STA    IICDD
0477 0a2c 81             RTS

```

NVM.AS5 page 10

```

0479
0480 *****
0481 *
0482 *          Delay (W2 x 2.3mS with a 2MHz bus).
0483 *
0484 *****
0485
0486 0a2d ae 04          P10   LDX    #4          9.2 mS

```

0487 0a2f bf 56	STX	W2		
0488 0a31 20 04	BRA	TPAU		
0489 0a33 ae 08	P20 LDX	#8	18.4	mS
0490 0a35 bf 56	PN	STX	W2	
0491 0a37 5f	TPAU	CLR X		
0492 0a38 ad 07	DLOOP	BSR	DALLY	12
0493 0a3a 5a		DEC X		3 15
0494 0a3b 26 fb		BNE	DLOOP	3 18 18x256/2=2304uS
0495 0a3d 3a 56		DEC	W2	
0496 0a3f 26 f6		BNE	TPAU	
0497 0a41 81	DALLY	RTS		

0498
0499
0500
0501
0502
0503
0504
0505
0506
0507

```
*****
*
*      B6 reset and interrupt vectors.
*
*****
```

0508 1ff2	ORG	\$1FF2		
0509				
0510 1ff2 08 00	FDB	START	SCI	
0511 1ff4 08 00	FDB	START	TIMER (OVER)	
0512 1ff6 08 00	FDB	START	TIMER (OUT CMP)	
0513 1ff8 08 00	FDB	START	TIMER (IN CAP)	
0514 1ffa 08 00	FDB	START	IRQ	
0515 1ffc 08 00	FDB	START	SWI	
0516 1ffe 08 00	FDB	START	RESET	
0517				
0518	END			

“FLOF” Teletext using M6805 Microcontrollers

By Peter Topping
MCU Applications
Motorola Ltd, East Kilbride

1. INTRODUCTION

The “T” members of the MC68HC05 family of MCUs provide a convenient and cost effective method of adding on-screen-display (OSD) to TVs and VCRs. As well as the 64-character OSD capability, they include 8 Kbytes of ROM (adequate for Teletext, frequency-synthesis, stereo and OSD), 320 bytes of RAM, a 16-bit timer and 8 pulse-width-modulated D/A converters. The MC68HC05T7 also includes IIC hardware and, by using a 56-pin package, 4 ports of I/O independent of the OSD, serial and D/A outputs. It is thus suitable for large full-feature chassis. The MC68HC05T1 is in the middle of the price/performance range and includes most of the features of the MC68HC05T7 but in a 40-pin package. This is achieved by sharing I/O with the other pin functions (SPI, OSD, D/A). Even if all these features are used, there is sufficient I/O for most applications.

The MC68HC05T2 is a 16K upgrade of the MC68HC05T1 and the MC68HC05T3 a 24K version with increased RAM (512 bytes) and enhanced OSD (112 characters and 2 rows of OSD buffer). The low cost MC68HC05T4 has 5 Kbytes of ROM and 96 bytes of RAM making it suitable in simpler (eg mono, non-Teletext) applications. The T4 and T7 also include a 14-bit D/A converter to facilitate voltage synthesis tuning. There are EPROM (and OTP) versions of the T3 (including T1 and T2 emulation), T4 and T7.

This application note describes an example of Teletext control software written for the MC68HC05T7 which directly controls Teletext chips of the type 5243. Spanish FLOF Teletext (level 1.5) is handled using packet X/26. If no CCT teletext chip is present on the IIC bus (as indicated by the lack of an acknowledge), all Teletext functions are disabled in software. About 3Kbytes of ROM are used allowing the code to fit into the 7.9K bytes available in an MC68HC05T7 along with tuning, OSD and stereo functions.

The software in the included listing has been written for the MC68HC05T7 but could, with a little modification, be implemented on other M6805 microcontrollers. A microcontroller without IIC hardware can be used as long as additional software is included to facilitate the IIC bus using I/O pins. An example of IIC master I/O driven software can be found in application note AN446.

2. “FLOF” TELETEXT FEATURES

Full Level One Feature (FLOF) Teletext utilises “ghost” packets to provide features in addition to those available with the original CCT Teletext. The primary enhancement is the provision of a menu with a choice of four linked pages selectable by the user with a single press of one of four coloured buttons on the remote control. The menu itself is sent in the ghost page using packet 24 while the linked page numbers are contained in packet 27. In addition to linked pages, packets 26 and 30 are used. Packet 26 allows for the substitution of selected characters in the display by special characters specific to a particular country. This example application includes the Spanish implementation of packet 26. The broadcast service data packet (8/30) is used to get the initial (index) page for each channel and to display station identification information.

"Ghost" packets handled

X/24 :

The FLOF menu information contained in this page extension packet is transferred by the microcomputer to row 24 of the display chapter. When links are disabled because there is no packet 27 (destination code 0) or when bit 4 of byte 43 is 0, row 24 is blank.

X/26 :

Optional handling of modes 1xxxx, 01111 and 00010 in accordance with the Spanish Teletext specification. All the additional characters which are available in the 5243 CCT chip are handled. The feature can be disabled with a hardware link on an I/O pin (see figure 1) so that the software can be used at level 1.0 in non-Spanish countries also using packet 26.

X/27 :

This packet contains the linked page numbers for the red, green yellow, blue and index (black) keys. Bit 4 on the link control byte (byte 43) is used to determine if these links are enabled (1) or disabled (0). When enabled, the Spanish specification requires that bits 1, 2 and 3 be used to enable the green, yellow and blue links respectively. This use of these bits is not defined in the World Teletext Specification. For this reason their use is selectable by a hardware link (see figure 1). If these bits are not used, all links (if enabled by bit 4) will be taken from packet 27 but will be automatically disabled if the broadcast links are default (FF3F7F) or invalid.

8/30 :

The broadcast service packet is used to supply the index page number on exit from standby and (if teletext is not stopped) after a channel change. Bytes 10-30 of this packet are displayed for 5 seconds on exit from standby and (if teletext is not stopped) after a channel change.

3. IMPLEMENTATION

The software listing is in two parts. The first part contains the "idle" loop and IIC routines from the main TV control part of the MC68HC05T7 application. The idle loop controls the timing of everything performed by the microprocessor, scans the local keyboard, checks whether or not an IR command has been received, etc. It also monitors the relevant flags in the Teletext chip and performs the tasks (eg fetching linked pages) which have to be performed independently of requests for the user.

The second and main listing is the Teletext module itself. It contains all the subroutines required to carry out automatic and user requested Teletext activity. Both modules use the same RAM allocation file (RAMT8.S05) which is included in the listing of the Teletext module. This listing also includes a symbol cross-reference table.

Figure 1 shows a simplified circuit diagram of the application. Most of the MC68HC05T7's I/O is used for purposes other than Teletext and is not shown in detail. Communication with the 5243 Teletext chip is via an IIC bus in which the T7 is always the master. The function of the three I/O pins used for Teletext is described under "Ghost packets handled" and "Inputs and Outputs".

A version of this Teletext software has been implemented on an MC68HC05C4 for use in a TV where the other control functions were handled by a separate microcontroller. The signal from the IR pre-amp was fed into the C4 which used Teletext commands to control a 5243 via a software IIC bus. Non-Teletext commands were re-generated by the C4 and sent to the other microcontroller. This arrangement allows Teletext to be added to a chassis which was originally designed without considering Teletext.

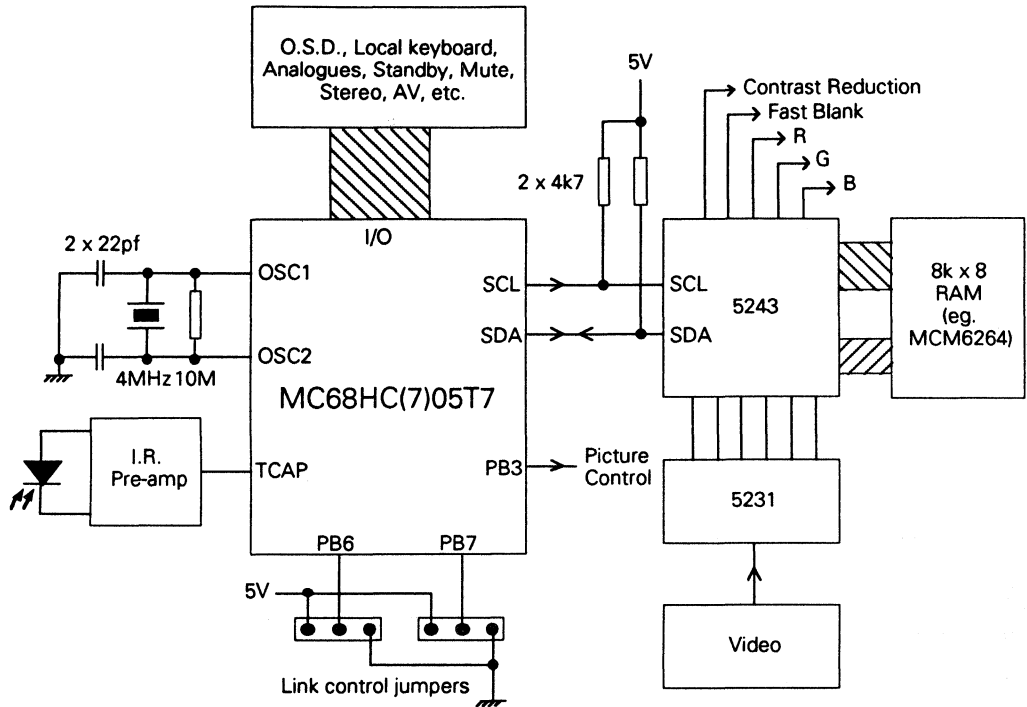


Figure 1. MC68HC(7)05T7 – Teletext application circuit

4. IDLE LOOP

In the example application the idle loop code is in the main TV control software module rather than in the teletext module. Listing 1 shows the relevant parts of this module. The loop time is 12.8mS and it is at this rate that the timing counters used by Teletext (CNT1 and CNT4) are incremented. The standby condition is checked first; if the TV set is in standby then there is no IIC activity and hence no reading from, or writing to, the 5243. If the TV has just exited from standby, as indicated by the flag 3,STAT2, then Teletext is initialised using the sub-routine RESTRT. This sub-routine writes to the 5243's control and mode registers (R5, R6 and R7) and checks that the IIC acknowledge is present. If there was no acknowledge, as indicated by flag 6,STAT7, then no further Teletext activity is attempted.

If an acknowledge is present, Teletext polling goes ahead, although it is suspended if there is a mute or time display. A mute indicates that the channel has just been changed, or no channel is tuned. During time display, all other Teletext activity is suspended. Re-initialisation using sub-routine START2 is performed if flag 7,STAT5 is set by a change of the tuned frequency.

Counter CNT4 is used to delay the transfer of packets 24 (page extension – FLOF menu), 27 (links), 26 (enhanced display characters) and the control bits from row 25 (display page) after the initial arrival of a page. When row 24 is read the 5243 FOUND flag is set to indicate that the arrival has been acted upon. If UPDATE is on then an update indicator appears if the update control bit (C9) is set or if the sub-page has changed or if it is the first arrival of the page. The update display is performed by the sub-routine ARRVD which clears the transient flags and enables the required display, i.e. page no. in normal mode and the whole of row 0 in sub-page mode. Any boxed information (eg sub-titles or newflash) in the current page is also displayed. The last Teletext function performed by the idle loop is the checking of the FOUND flag in the 5243. This is accessed via the IIC bus; it is on the last (not displayed) row of the display page along with the current page and sub-page numbers and the control bits.

If there is a current Teletext transient (time, row 0 box or packet 8/30), the transient control branch from the idle loop is executed. This routine checks to see if it is time to end the transient. If it is, the subroutine OSDLE is executed. It resets transients for both the OSD generated by the MC68HC05T7 and Teletext. The sub-routine RSTMD2 performs this function for Teletext. It is called from within the sub-routine OSDLE (not listed).

5. REMOTE CONTROL FUNCTIONS

TV/TXT

Toggle between TV & Teletext mode.

0-9

Number keys for entry of page and sub-page numbers

Red, Green, Yellow, Blue

Linked page access keys. The decoder stores four pages of text. These are the display page and the three pages corresponding to the red, green and yellow links. The blue linked page is not acquired in advance. In the absence of FLOF data or if the links are disabled by the control bit in packet 27, the red key is page+1 and the green key page-1. Under these circumstances the requested page and the next three pages are acquired.

PC+/-

These keys always select page+1/page-1 regardless of the availability of FLOF information. As with the red, green and yellow keys, the page is displayed immediately if it is already in RAM.

INDEX

This key operates as an additional link with the difference that if the link is invalid the initial page from packet 8/30 is selected.

SUB-PAGE/TIME

Text mode: Enter sub-page mode, (max. 3979). TV mode: Display time in top-right-hand corner for 5 seconds. Pressing this key during a station identification display (packet 8/30 bytes 10-30) can be used to extend this display beyond the five seconds it appears for, after a channel change.

STOP

Halt acquisition, "STOP" is displayed instead of page number. Press again to restart. If acquisition has been stopped by partially entering a new page number then this key can be used to return to the original page.

MIX/NO-MIX

Toggle between Teletext and mixed display. Use of this key causes the display of the top status row for 5 seconds if it is not being displayed because the current page is a newsflash or a sub-title. 5243 contrast reduction is enabled in mixed mode.

FULL/TOP/BOT

Selects one of the three display formats, normal, top half enlarged, bottom half enlarged.

REVEAL

Reveal hidden text, toggle action.

UPDATE

Return to picture until a new version of the requested page arrives. When it arrives, its page no. is displayed in the top-right-hand corner, the key operates in both TV and Teletext mode, set is put into TV mode. Any boxed information (alarm clock, newsflash or sub-title) will be displayed. In sub-page mode the complete header is displayed so that both page & sub-page numbers can be seen. Cancel update by entering Teletext mode and then going back to TV mode by pressing the TV/Text key twice.

6. TELETEXT SUBROUTINES

6a. Subroutines: TVTX, UPDATE, DIGIT0 and GETIT

The Teletext module (listing 2) comprises various sub-routines which are used both by the idle loop and to perform any Teletext actions initiated by commands from the IR remote control. They are described in the order in which they appear in the listing.

TVTX is executed when the TV/TEXT button is pressed. Its function is to toggle between TV mode and Teletext mode. The flag 0,STAT indicates the current mode. This flag routes the microprocessor to execute either TXTOFF or TXTON according to the current mode. TXTON checks that Teletext hardware is present and does nothing if there has been no IIC acknowledge. If, however, a 5243 is present in the TV, it clears all transients (OSDLE) and sets up the Teletext mode. It initialises the control registers (R5 and R6) to display text and background both in and out of boxes. For newsflashes the set-up is text and background within boxes and picture outside. TXTOFF also resets transients but forces TV mode and sync. Polling and updating continue as a background activity.

When the UPDATE key is pressed the update flag 4,STAT2 is set and TXTOFF executed so the TV is forced to TV mode. If there is a current transient hold (eg time), the hold is cleared before TXTOFF is executed.

The number entry sub-routine DIGIT0 branches to DIGITS in sub-page mode but otherwise accepts any number key as a page number input. Three digits are required, the pointer PDP holding the current position (0, 1 or 2 for hundreds, tens or units). During entry the flag 2,STAT is set to stop Teletext activity. The numbers have to be written to the top-left-hand corner of the display page as well as saved in RAM. Once all three digits have been entered the page is requested and page acquisition restarted.

The code at label GETIT makes this request after first checking whether or not the selected page has already been requested (it could be the current display page or an already requested linked page). If it has, then a switch is made to the chapter associated with the appropriate acquisition circuit and no new request is generated. If not, the new request is made and the FOUND flag set.

6b. Subroutines: Colours, INDEX, NPAGE and PPAGE

The four colour keys (Red, Green, Yellow and Blue) are primarily intended for selecting Teletext linked pages. When pressed the chapter which corresponds to the appropriate acquisition circuit is selected for display. If links are disabled (by the link control bit or because there is no packet 27), then the RED and GREEN keys select current page +1 and -1 respectively. This choice is taken according to the state of flag 3, STAT3 which reflects the condition of the link control bit in packet 27. The code executed by RED, if links are not in use, is the same as that executed by the "+" function (NPAGE) which always selects the next page. Similarly the alternative GREEN function (PPAGE) is the same as for the "-" key. The YELLOW and BLUE keys do nothing under these circumstances. In Spanish Teletext the GREEN, YELLOW and BLUE links can be individually inhibited, but the RED link is only inhibited if all links are off.

The chapter associated with the selected page is displayed immediately if it has already been requested. This will normally be the case if a linked page (red, green or yellow) has been selected. The code at label LPT is executed if the page has already been requested. If not, a jump to CLRPD is performed. CLRPD is a label within DIGIT0; the code at CLRPD requests a new page just as if the page number had been entered manually. If the required acquisition circuit is the one already current, then the "unstop" code is executed. This causes the green page-being-looked-for header to roll as though the page number had just been entered. This means that something can be seen to happen in the case where the linked page differs only from the current page in its sub-page number. Linked sub-pages are not fully supported in this implementation as they are rarely used by broadcasters and would significantly increase the size of the software. When the chapter is changed the Teletext PBLF (page being looked for) flag is checked. If it is low the FOUND flag is cleared. This forces the fetching of the links associated with the new display page. If the page is not already in, this will automatically happen when it arrives so the FOUND flag does not need to be cleared.

The BLUE (or cyan) key is different in that its page will not normally be immediately available (the four pages: display, red, green and yellow occupy the four acquisition circuits and RAM chapters).

The INDEX (or black link) function is similar to BLUE except that if its link is not valid it defaults to the initial (index) page number supplied by packet 8/30 (see sub-routine GIP).

6c. Subroutines: LINK, GLP1, GLP2, SRCH, CHCK1 and NOTOKx

The sub-routine LINK allocates the three linked pages (RED, YELLOW and GREEN) to the three free acquisition circuits (not in use by the display page). To do this it checks the page numbers in turn to see if they have already been requested. If so they are left in their current acquisition circuit. If they have not already been requested the page number is put into a LIFO. Only 0-9 are regarded as acceptable digits for page numbers; this is consistent with the Spanish specification although the additional HEX numbers (A-F) may be used experimentally or by Teletext page generators. Within this first loop the sub-routine GLP1 is used to get the linked page numbers from packet 27, perform a decode of the Hamming encoded data and calculate the new magazine number (page hundreds) if different from that of the display page. GLP1 uses sub-routine SRCH to check if the page has already been requested. If there are no links, or if links are disabled, then displayed page +1, +2 and +3 are requested.

The second loop in LINK allocates new page numbers to the remaining unused acquisition circuits. It uses GLP2 to clear the relevant chapters in the Teletext memory and make the new requests. Subroutine CHCK1 is used to check whether or not an acquisition circuit is in use before it is loaded with a new page number from the LIFO.

This method of organising new page requests prevents unnecessary requests being made for pages already requested. This is particularly important when links are disabled and pages are being requested using the "+" or "-" functions. Under these circumstances when the page number is incremented (or decremented) only one new page has to be requested (new display page+3), while page, page+1 and page+2 do not need to change and can be left in their current acquisition circuits.

NOTOK3 and NOTOK2 handle the RED and GREEN functions when links are disabled. They are disabled if the link control bit (packet 27 bit 3, byte 43) is zero or if there is no packet 27. These subroutines respectively increment and decrement the current page number (units and tens). The current magazine number (page hundreds) is not affected.

6d. Subroutines: ROW24, W2B, R2B, GCYI, CLINK and DECODE

ROW24 is used to transfer ghost row 20 (packet 24) into the display chapter. This has to be done via the IIC bus. The loop reads two bytes via the IIC (sub-routine R2B) bus from the ghost page and writes it to the display page (sub-routine W2B). The FOUND flag is then set to indicate that the arrival of the page has been recognised and acted upon. This sub-routine is only called by the idle loop and is used along with the other sub-routines which get information from the ghost page (CLINK, LINK and GET25).

R2B and W2B use IIC routines READ and SEND which are outwith the Teletext module. These subroutines will differ according to the microprocessor in use. An MC68HC05C8 implementation would need to use I/O lines (see reference for suitable software) while the MC68HC05T7 can use its IIC hardware. The routines used in this example are included in the listing extract from the TV control software module (listing 1).

The sub-routine GCYI is used by LINK to store the data associated with the BLUE and INDEX links. As explained above, these pages will not be acquired in advance, the page number only being sent to an acquisition circuit if requested by an IR command.

CLINK fetches the link control byte from packet 27 if the destination code is OK and, after decoding the Hamming encoded data, transfers the bits to STAT3.

The Hamming decode sub-routine DECODE corrects for single bit errors. This is done with in-line code using the table HAM (at the end of listing 2) as this uses less ROM than an algorithmic method.

6e. Subroutines: MIX, TRANx, TXTx, HOLD, and NOHOLD

The mixed display capability of the Teletext chip (5243) is toggled using an IR key which calls the sub-routine MIX. When mixed mode is entered, interlaced broadcast sync. (312/313) is selected because the non-interlaced sync. used for teletext is not suitable if a TV picture is present on the screen. This is set up via the 5243 mode register R1. The control registers R5 and R6 are updated to provide the mixed display.

When returning to a non-mixed display, the code at NOMIX is used to re-configure the control registers and to set up a Teletext only 312/312 non-interlaced sync. This sync. reduces adjacent line flicker in a pure Teletext display.

The sub-routine TRAN2 sets up a transient which retains a black background on the top row so that the page number, time etc. can be seen clearly. This type of transient is also started if the page number or sub-page number is being entered in mixed mode. Sub-routines TRAN1, TRAN2 and TRAN3 are used to initialise the various transient displays. These displays are cancelled as discussed above by actions taken within the idle loop controlled by the free-running timer within the MC68HC05T7.

The TXTx sub-routines are used in conjunction with the IIC SEND routine to write to various sub-sets of the registers within the 5243.

If the Teletext STOP function is requested by an IR command the routine HOLD is executed. This is a toggled function when requested in this way. HOLD displays the word "STOP" in place of the page number and stops the display acquisition circuit by clearing the 5243 HOLD flag accessed via its page request register R3.

NOHOLD is executed to restart the display acquisition circuit. It returns the page number to the top-left-hand corner. If a new page number has been partially entered, a press of STOP (executing an UNHOLD) will allow a return to the most recent page request. This takes only a single press as the start of the entry of a new page number cause a HOLD. The completion of a page number entry (3 digits) causes a NOHOLD.

6f. Subroutines: REVEAL, EXPTB and TIME

The REVEAL function causes any hidden display information to appear. It is controlled by a bit in the display mode register (R7). The software example leaves any revealed information permanently displayed. If, however, it is required that such information disappear when the page is updated (this may be better for a quiz page), then the two commented out lines (80 and 81) in the idle loop should be enabled.

The display expand facility is controlled by another two bits in R7. The EXPTB sub-routine cycles through normal, top-half double height and bottom-half double height.

The example application uses a single IR key (subroutine TIME) for both the display of the Teletext clock and the entry into sub-page mode. If the set is in TV mode then the time is displayed for 5 seconds. If the TV is in Text mode then sub-page mode is selected. Sub-page number entry is described in the following section. When the Teletext clock is requested it appears (boxed) at the top-right-hand corner. It is removed by the idle loop 5 seconds after the last press of the time button. When the time is being displayed all other Teletext activity is stopped using UCHOLD.

6g. Subroutines: DIGITS, SUBPG, GET25 and GET26

DIGITS is the sub-page version of DIGIT0 and uses similar code. More checks on the input data are required as the four digits of the sub-page number have different maximum values. These maximums are 3 for thousands, 7 for the tens and 9 for the hundreds and units. These values reflect the sub-page number's original use as a time (24hr format). For tens and thousands a keyed 8 becomes a 0 and a 9 becomes a 1; for thousands only 4, 5, 6 and 7 become 0, 1, 2 and 3 respectively.

The code at the label SETIT is the sub-page equivalent of GETIT, described above. It requests the new sub-page and sets the FOUND flag.

The sub-routine SUBPG is called when the TIME (or clock) key is pressed (TV in Teletext mode). It toggles between normal mode and sub-page mode. When sub-page mode is entered the page number display (P—) is replaced with **** to indicate the mode change and to prompt for the entry of a sub-page number. Once all four digits have been entered the new sub-page is requested by SETIT. The code at the label RSTR is used to exit from this mode back to the normal (page number) mode, restoring the page number display to the top-left-hand corner.

GET25 is used by the idle loop to get the information stored in row 25 of the display chapter. This row is not displayed but contains various information used by the control microprocessor. The current page number, magazine number, sub-page number, Teletext control bits and the FOUND and PBLF flags are available. GET25 gets the required information and stores it in the RAM of the MC68HC05T7.

At the end of this sub-routine the I/O line 7, portB is checked. If it is low, packet 26 is handled. If it is high, this packet is disabled. This would be required if this application were to be used in a country other than Spain which used packet 26. It would require to be switched off as the enhanced display feature uses different characters depending on the country. In countries which do not use packet 26 (eg the UK) it does not matter whether or not packet 26 is enabled.

If packet 26 is enabled, GET26 processes all packet 26 data present in the ghost page. The tables G2TAB, G3TAB and CTAB contain the characters used to replace the character at the display location defined by each packet.

6h. Subroutines: GIP, R24T and SR24T

The sub-routine GIP gets the initial (index) page from packet 8/30. It will be doing this as the set is brought out of standby or just after a channel change. It may thus initially get a poor signal (or there may be no Teletext) so it tries repeatedly until it finds a valid packet 8/30 format 1. If this is not found after 96 tries it gives up and sets the flag 6,STAT2 to indicate that there is no packet 8/30 (or no Teletext). In this circumstance it defaults to an index page number of 100.

R24T transfers bytes 10-30 of the broadcasting service data packet (8/30) into the display chapter. It is called once a second for five seconds after power-on or a channel change. The data is transferred to row 0 of the display page which can be displayed either at the bottom or, as in this example, the top of the screen. This transient display is setup using the sub-routine SR24T if Teletext is present. If the flag 6,STAT2 has been set by GIP as described above then SR24T does nothing. The transient display is terminated by code executed at the appropriate time from within the idle loop.

7. INPUT AND OUTPUTS

Apart from the IIC bus, only three pins on the controlling microprocessor are relevant to Teletext. Two inputs select the usage of packets 26 and 27 and one output can be used to control any hardware which requires to be changed according to whether or not there is a TV picture currently being displayed. In many applications some or all of these functions will not be required and could be eliminated from the software thus freeing up the pins for other uses.

PB3)

This pin is active (high) during a pure (no-mixed, no-boxed) teletext display, otherwise it is low.

PB6)

When this pin is low, Spanish use of link control bits 1, 2 and 3 is enabled. When it is high, these bits are ignored.

PB7)

Packet 26 control. When low, packet 26 is enabled and handles all the Spanish alternate characters which are available in the 5243. When PB7 is high, packet 26 is ignored.

8. REFERENCES

Application note AN446, MCM2814 Gang-programmer using an MC68HC05B6.

LISTING 1

```

30
31
32
33
34
35
36 00000000 0d13fd      ILP  BRCLR  6,TSR,*      OUTPUT COMPARE FLAG
37 00000003 >3c00      INC  CNT1      TELETEXT TRANSIENT
38 00000005 >3c00      INC  CNT4      ROW 24 DELAY
39 00000007 >3c00      INC  CNT5      MUTE TRANSIENT
40 00000009 >cc0000    JSR  KBD      KEYBOARD & TIMERS
41 0000000c 030104    BRCLR 1,PORTB,FON  STANDBY ?
42 0000000f >1600    BSET  3,STAT2   MAKE SURE FLAG AGREES
43 00000011 2055f    BRA  F1        AND IDLE WITH NO IIC ACTIVITY
44 00000013 >070009    BRCLR 3,STAT2,ALRON NO, JUST ON ?
45 00000016 >1700    BCLR  3,STAT2   YES, RESTART
46 00000018 >1500    BCLR  2,STAT2   CLEAR THIS FLAG ALSO ?
47 0000001a >1f00    BCLR  7,STAT5   RE-INITIALISATION NOT NECESSARY
48 0000001c >cd0000    JSR  RSTRT
49 0000001f >cd0000    ALRON JSR  VCRPOLL   POLL SCART LINES
50 00000022 >02004d    BRSET 1,STAT2,F1  REMOTE REPEATING ?
51 00000025 >02004a    BRSET 1,STAT4,F1  LOCAL REPEATING ?
52 00000028 >0c0047    BRSET 6,STAT7,F1  TELETEXT CHIP ON BUS ?
53 0000002b >040044    BRSET 2,STAT2,F1  SEARCH/STANDBY ?
54 0000002e >0A0041    BRSET 5,STAT,F1   TIME DISPLAY HOLD
55 00000031 >06003e    BRSET 3,STAT4,F1  TRANSIENT MUTE ?
56 00000034 >0c003b    BRSET 6,STAT4,F1  COINCIDENCE MUTE ?
57 00000037 >0f0005    BRCLR 7,STAT5,DNTRS TO BE RE-INITIALISED ?
58 0000003a >1f00    BCLR  7,STAT5   YES, CLEAR FLAG &
59 0000003c >cd0000    JSR  START2    RE-INITIALISE TELETEXT
60 0000003f >01001c    DNTRS BRCLR 0,STAT2,NO24
61 00000042 >bd0000    LDA  CNT4      PAUSE WHILE PACKET 24
62 00000044 a130    CMP  #48      (PAGE EXT.) ARRIVES
63 00000046 252a    BLO  F1
64 00000048 >cd0000    JSR  CLINK    CHECK LINK CONTROL BYTE
65 0000004b >cd0000    JSR  LINK    FETCH LINKS
66 0000004e >cd0000    JSR  ROW24   FETCH ROW 24 AND SET FOUNDB
67 00000051 >cd0000    JSR  GET25   GET ROW 25 & PACKET 26
68 00000054 >090005    BRCLR 4,STAT2,NOUP UPDATE ENABLED ?
69 00000057 >0b0002    BRCLR 5,STAT2,NOUP DIFFERENCES ?
70 0000005a a66e    BSR  ARRD0
71 0000005c >1100    NOUP BCLR  0,STAT2
72 0000005e >b600    NO24 LDA  ACC
73 00000060 >b700    STA  R8
74 00000062 a608    LDA  #8      COLUMN 8 (FOUNDB & PBLF)
75 00000064 >b700    STA  R10
76 00000066 a619    LDA  #25    ROW
77 00000068 >cd0000    JSR  R28
78 0000006b >0b0104    BRSET 4,IOBUF+1,F1 FOUNDB FLAG SET ?
79 0000006e >1000    BSET  0,STAT2  NO, SO FETCH GHOST ROWS
80
81
82
83
84
85
86
87
88
89
90
91
92
93 0000007b >b600      LDA  CNT1      YES
94 0000007d a150      CMP  #80
95 0000007f 2403      BHS  NILP
96 00000081 >cc0000    JMP  ILP      1S TIMER
97 00000084 >b600      LDA  R4
98 00000086 a104      CMP  #4
99 00000088 2603      BNE  NOTE
100 0000008a >cd0000    JSR  R24T    IF PAGE 4 THEN IT'S
101 0000008d >3f00    CLR  CNT1    THE 0/30 TRANSIENT
102 0000008f >3a00    DEC  TMR    CLEAR 1S TIMER
103 00000091 2703      BEQ  DNILP   DECREMENT SECONDS COUNTER
104 00000093 >cc0000    JMP  ILP    TRANSIENT FINISHED ?
105
106
107 00000096 >cd0000    DNILP JSR  OSDLE  OSD TIMEOUT (INC RSTMD)
108 00000099 >cc0000    JMP  ILP
109
110
111
112
113
114
115
116
117
118 0000009c >010003    RSTMD BRCLR 0,STAT5,SOS2 2-DIGIT Pr. No. ENTRY ?
119 0000009f >cd0000    JSR  RES      YES, RESTORE DISP
120 000000a2 >1500    SOS2 BCLR  2,STAT4  MAKE SURE ITS PROGRAM MODE
121
122 000000a4 >1900    RSTMD2 BCLR  4,STAT4  RESET OSD TRANSIENT FLAG
123 000000a6 >1900    RSTMD3 BCLR  4,STAT  RESET MAIN TRANSIENT FLAG
124 000000a8 >0b0011    BRCLR 5,STAT,TXTR1  TIME HOLD ?
125 000000ab >1b00    BCLR  5,STAT  YES, CLEAR IT
126 000000ad a603      LDA  #503
127 000000af >b700    STA  R5
128 000000b1 >b700    STA  R6
129 000000b3 >cd0000    JSR  TXT2    STOP TIME EXIT FLASH
130 000000b6 >0A0003    BRSET 2,STAT,TXTR1 OTHER HOLD ?
131 000000b9 >cd0000    JSR  NOTTH   NO, SO CLEAR HOLD
132 000000bc >1100    BCLR  0,R7    BOX OFF ROW 0
133 000000be >000006    BRSET 0,STAT,TXTR2 TELETEXT ?
134 000000c1 >1b00    LDA  ACC
135 000000c3 >b700    STA  R4
136 000000c5 >3f00    CLR  R7
137 000000c7 >cc0000    TXTR2 JMP  TXT2    NO, ALL BOXES OFF
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

139
140
141
142
143
144
145 000000ca >b600
146 000000cc >b700
147 000000ce >1900
148 000000d0 >1b00
149 000000d2 4f
150 000000d3 >cd0000
151 000000d6 >0c0005
152 000000d9 a606
153 000000db >cd0000
154 000000de a646
155 000000e0 >b700
156 000000e2 >b700
157 000000e4 a603
158 000000e6 >050002
159 000000e9 a602
160 000000eb >b700
161 000000ed >cc0000
162
163
164 000000f0 a610
165 000000f2 >b700
166 000000f4 a606
167 000000f6 >b700
168 000000f8 >b700
169 000000fa >3f00
170 000000fc >cd0000
171
172 000000ff 013c03
173 00000102 >1c00
174 00000104 81
175
176 00000105 >cc0000
177
178
179 00000108 >b600
180 0000010a >b700
181 0000010c >1100
182 0000010e 81
183
184
185
186
187
188
189
190 0000010f ad23
191
192 00000111 >bf00
193 00000113 >1100
194 00000115 >b600
195 00000117 ad25
196
197 00000119 >b600
198 0000011b a180
199 0000011d 2606
200 0000011f >b600
201 00000121 ad1b
202 00000123 >3c00
203
204 00000125 >b600
205 00000127 f6
206 00000128 ad14
207 0000012a >3c00
208 0000012c >3a00
209 0000012e 26e9
210
211 00000130 1b3b
212 00000132 9a
213 00000133 81
214
215 00000134 9b
216 00000135 3f3c
217 00000137 3f3a
218 00000139 a6b0
219 0000013b b73b
220 0000013d 81
221
222 0000013e b73d
223 00000140 0f3cfd
224 00000143 81
225
226 00000144 adc9
227 00000146 a602
228 00000148 >cd0000
229

```

```

*****
*
* Updated page has arrived.
*
*****
ARRVD LDA ACC
STA R4
BRCLR 4,STAT KILL TRANSIENTS
BCLR 5,STAT
CLRA
JSR BOX00N
BRSET 6,STAT,SPMD SUB-PAGE MODE ?
LDA #6 NO, SMALL BOX
JSR BOX00F
SPMD LDA #446
STA R5
STA R6
LDA #803
BRCLR 2,C3,NNF NEWSFLASH ?
LDA #802 YES, NO ROW 0
NMF STA R7
JMP TXT2
RESTRT LDA #510 BROADCAST SYNC.
STA R1
LDA #6
STA R5
STA R6
CLR R7
JSR TXT2 SWITCH PICTURE ON
BRCLR 0,MSR,ACKOK ACKNOWLEDGE ?
BSET 6,STAT7 NO, SET FLAG
RTS
ACKOK JMP INITXT
RES LDA PROG YES, RESTORE PROG. NO.
STA DISP
BRCLR 0,STAT5
ABS RTS
*****
*
* IIC write.
*
*****
SEND BSR IICSU
STX DPNT SAVE X
BCLR 0,ADDR SET-UP TO WRITE
LDA ADDR
BSR SHIFT SEND CHIP ADDRESS
WRBU LDA ADDR STEREOTONE ?
CMP #580
BNE WRB
LDA SUBADR YES, SO ENABLE AUTO
BSR SHIFT SUB-ADDRESS INCREMENTING
INC SUBADR
WRB LDX DPNT DATA BUFFER POINTER
LDA 0,X
BSR SHIFT SEND DATA
INC DPNT
DEC W1
BNE WRBU DONE ?
BCLR 5,MCR STOP
CLI
RTS
IICSU SEI
CLR MSR IIC SET-UP
CLR FDR 90 KHz
LDA #580 ENABLE IIC AS MASTER
STA MCR TRANSMITTER & START
RTS
SHIFT STA MDR
BRCLR 7,MSR,*
RTS
WRITE BSR SEND
LDA #2
JSR TPAU WAIT 10ms (EEPROM WRITE)

```



```

230
231
232
233
234
235
236 0000014b addc
237 0000014d >b600
238 0000014f >b701
239 00000151 >b600
240 00000153 a1a1
241 00000155 2602
242 00000157 >3c00
243
244 00000159 add9
245 0000015b >1100
246 0000015d >b600
247 0000015f addd
248 00000161 >b600
249 00000163 add9
250 00000165 1b3b
251
252 00000167 1a3b
253 00000169 >1000
254 0000016b >b600
255 0000016d adcf
256 0000016f 193b
257 00000171 163b
258 00000173 b63d
259
260
261
262
263
264
265 00000175 0f3cfd
266 00000178 1b3b
267 0000017a b63d
268 0000017c >b700
269 0000017e 9a
270 0000017f 81
271
272
273
274
275
276
277
278 00000180 3f3c
279 00000182 80

```

```

*****
*
*      IIC read.
*
*****

```

```

READ  BSR  READ1  GET FIRST BYTE
      LDA  IOBUF
      STA  IOBUF+1  MOVE IT UP
      LDA  ADDR
      CMP  #8A1     NVH ?
      BNE  READ1
      INC  SUBADR   YES, NEXT SUB-ADDRESS

READ1 BSR  IICSU
      BCLR 0,ADDR  RM - 0 ALWAYS WRITE (SUB-ADDRESS)
      LDA  ADDR
      BSR  SHIFT  SEND CHIP-ADDRESS
      LDA  SUBADR
      BSR  SHIFT  SEND SUB-ADDRESS
      BCLR 5,MCR  NO STOP BUT
      BSET 5,MCR  A RESTART
      BCLR 0,ADDR SET BIT 0 FOR READ
      LDA  ADDR
      BSR  SHIFT  RE-SEND CHIP ADDRESS
      BCLR 4,MCR  CHANGE TO RECEIVER
      BSET 3,MCR  SWITCH OFF ACK.
      LDA  MDR     INITIATE RECEPTION

*      BRCLR 7,MSR,*  WAIT FOR IT
*      BSET 3,MCR     SECOND LAST SO SWITCH OFF ACK.
*      LDA  MDR       GET FIRST BYTE
*      STA  IOBUF+1   AND SAVE IT

      BRCLR 7,MSR,*  WAIT FOR IT
      BCLR 5,MCR     LAST BYTE SO STOP
      LDA  MDR       GET BYTE
      STA  IOBUF     AND SAVE IT
      CLI
      RTS

```

```

*****
*
*      IIC interrupt.
*
*****

```

```

NBINT CLR  MSR
RETURN RTI

```



```

27 .....
27 *
27 *       Teletext RAM allocation.
27 *
27 .....
27
27 00000000   SUB1   RMB   1
27 00000001   R1     RMB   1       mode register
27 00000002   R2     RMB   1       page request address register
27 00000003   R3     RMB   1       page req. data reg. col. 0 : mag.
27 00000004   C1     RMB   1       " " " " " 1 : pgt.
27 00000005   C2     RMB   1       " " " " " 2 : pgu.
27 00000006   C3     RMB   1       " " " " " 3 : ht.
27 00000007   C4     RMB   1       " " " " " 4 : hu.
27 00000008   C5     RMB   1       " " " " " 5 : mt.
27 00000009   C6     RMB   1       " " " " " 6 : nu.
27 0000000a   SUB2   RMB   1
27 0000000b   R4     RMB   1       display chapter register
27 0000000c   R5     RMB   1       display control register (normal)
27 0000000d   R6     RMB   1       display control register (news/sub)
27 0000000e   R7     RMB   1       display mode register
27 0000000f   SUB3   RMB   1
27 00000010   R8     RMB   1       active chapter register
27 00000011   R9     RMB   1       active row register
27 00000012   R10    RMB   1       active column register
27 00000013   R11    RMB   1       active data register
27 00000014   PH     RMB   1       2nd " "
27 00000015   PT     RMB   1       3rd " "
27 00000016   PU     RMB   1       4th " "
27 00000017   LIFO   RMB   9       LINKED PAGE No. LIFO BUFFER
27 00000020   PAGE   RMB   7       PAGE No. INPUT BUFFER
27 00000027   PAG0   RMB   3       ACU PAGE No.
27 0000002a   PAG1   RMB   3       AC1 PAGE No.
27 0000002d   PAG2   RMB   3       AC2 PAGE No.
27 00000030   PAG3   RMB   3       AC3 PAGE No.
27 00000033   PAGC   RMB   3       CYAN PAGE No.
27 00000036   PAGI   RMB   3       INDEX PAGE No.
27 00000039   PDP    RMB   1       PAGE DIGIT POINTER
27 0000003a   ACC    RMB   4       DISP, RED, GREEN, YELLOW AC. CIR.
27 0000003e   WACC   RMB   1       WORKING ACC No.
27 0000003f   ADDR   RMB   1       IIC ADDRESS
27 00000040   DPNT   RMB   1       IIC DATA POINTER FOR WRITE
27 00000041   SUBADR  RMB   1       IIC SUB-ADDRESS
27 00000042   IOBUF  RMB   4       IIC BUFFER, +2 & +3 RSRVD FOR PLL
27 00000044   STAT2  RMB   1       0: ROW24 FETCH FLAG
27 *         *         *         1: REMOTE REPEATING
27 *         *         *         2: SEARCH/STANDBY IIC LOCK
27 *         *         *         3: STANDBY STATUS
27 *         *         *         4: UPDATE PENDING
27 *         *         *         5: DIFFERENCE FOUND
27 *         *         *         6: NO TELETEXT TRANSMISSION
27 *         *         *         7: MIXED
27
27 00000047   STAT3  RMB   1       0: CYAN LINK ON
27 *         *         *         1: YELLOW LINK ON
27 *         *         *         2: GREEN LINK ON
27 *         *         *         3: LINKS/ROW24 ON
27
27 .....
27 *
27 *       General RAM allocation.
27 *
27 .....
27
27 00000048   PLLHI  RMB   1       PLL DIVIDE RATIO MSB
27 00000049   PLLOW  RMB   1       PLL DIVIDE RATIO LSB
27 0000004a   W1     RMB   1       WORKING
27 0000004b   W2     RMB   1       "
27 0000004c   W3     RMB   1       "
27 0000004d   COUNT  RMB   1       LOOP COUNTER
27 0000004e   COUNT  RMB   1       LOCAL KEYBOARD COUNTER
27 0000004f   CNT    RMB   1       12.0ms (inc. free running)
27 00000050   CNT1   RMB   1       12.8ms (inc. reset every 1S during transient)
27 00000051   CNT3   RMB   1       3.25 S (dec, automatic standby timeout)
27 00000052   CNT4   RMB   1       12.8ms (cleared for row24 delay when page arrives)
27 00000053   CNT5   RMB   1       12.8ms (inc. transient mute)
27 00000054   TMR    RMB   1       TRANSIENT DISPLAY SECONDS COUNTER
27 00000055   STAT   RMB   1       0: TV/TELETEXT
27 *         *         *         1: IIC R/W
27 *         *         *         2: HOLD
27 *         *         *         3: IR REPEAT INHIBIT
27 *         *         *         4: TRANSIENT DISPLAY ON
27 *         *         *         5: TIME HOLD
27 *         *         *         6: SUB-PAGE MODE
27 *         *         *         7: IR TASK PENDING
27 00000056   STAT4  RMB   1       0: KEY FUNCTION PERFORMED
27 *         *         *         1: LOCAL REPEATING
27 *         *         *         2: P/C PROG : 0, CHAN : 1
27 *         *         *         3: MUTE (TRANSIENT)
27 *         *         *         4: OSD STATUS TRANSIENT
27 *         *         *         5: MUTE (BUTTON)
27 *         *         *         6: COINCIDENCE MUTE
27 *         *         *         7: SEARCH
27 00000057   PWR    RMB   1       $$$ AT RESET, SAA NORMALLY
27 00000058   PROG   RMB   1       CURRENT PROGRAM NUMBER
27 00000059   CHAN   RMB   1       CURRENT CHANNEL NUMBER
27 0000005a   DISP  RMB   1       CURRENT DISPLAY NUMBER (PROGRAM)
27 0000005b   DISC  RMB   1       CURRENT DISPLAY NUMBER (CHANNEL)
27 0000005c   FTUNE  RMB   1       FINE TUNING REGISTER
27 0000005d   AVOL   RMB   1       VOLUME LEVEL
27 0000005e   BRLL   RMB   1       BRILLIANCE LEVEL
27 0000005f   KEY    RMB   1       CODE OF PRESSED KEY (LOCAL)
27 *         *         *         *NUM0 RMB 4       LED DISPLAY RAM
27 00000060   IRRA1  RMB   1       IR INTERRUPT TEMP.
27 00000061   IRRA2  RMB   1       " "
27 00000062   IRRA3  RMB   1       " "
27 00000063   IRRA4  RMB   1       " "
27 00000064   DIFFH  RMB   1       IR TIME DIFFERENCE
27 00000065   DIFFL  RMB   1       " "
27 00000066   IRL    RMB   1       IR CODE BIT
27 00000067   IRL    RMB   1       COLLECTION
27 00000068   IRCODE  RMB   1
27 00000069   IRCNT  RMB   1
27 0000006a   TRMCT  RMB   1
27 0000006b   OLDIR  RMB   1

```

```

27 .....
27 *
27 *   RAM allocation for Stereoton.
27 *
27 .....
27
27 0000006c SHADMAT RMB 1      TEMPORARY MATRIX
27 0000006d LBAL RMB 1      Loudspeaker balance variable
27 0000006e SNDMD RMB 1      SOUND MODE 0:ST, 1:DA, 2:DB, 3:W, 4:M, 5:FM
27 0000006f ABAV RMB 1      SCART SOUND MODE 0:STEREO, 1:DUAL A, 2:DUAL B
27
27 00000070 K1 RMB 1      K1 level (reg 0)
27 00000071 LVL RMB 1      Loudspeaker left volume (reg 1)
27 00000072 LVR RMB 1      Loudspeaker right volume (reg 2)
27 00000073 HVL RMB 1      Headphone volume left (reg 3)
27 00000074 HVR RMB 1      Headphone volume right (reg 4)
27 00000075 TONE RMB 1      Tone variable (Bass/Treble) (reg 5)
27 00000076 MATRIX RMB 1      Current matrix (reg 6)
27 00000077 K2 RMB 1      K2 level (reg 7)
27
27 00000078 STAT5 RMB 1      0: 2-DIGIT PROGRAM ENTRY
27 *          1: ANY MUTE REQUIRED ?
27 *          2: OSD NAME TABLE
27 *          3: OSD DEFAULT P/C NUMBER
27 *          4: ANALOGUE OSD ON
27 *          5: NAME-TABLE STANDARD
27 *          6: STANDARD CHANGED
27 *          7: RE-INITIALISE TELETEXT
27
27 00000079 STAT6 RMB 1      0: AV MODE BIT 0 (0:TV, 1:S-VHS)
27 *          1: AV MODE BIT 1 (2:SCRT1, 3:SCART2)
27 *          2:
27 *          3:
27 *          4:
27 *          5:
27 *          6: SCART INPUT #1
27 *          7: SCART INPUT #2
27
27 0000007a STAT7 RMB 1      0: AV MODE CHANGE
27 *          1: FORCE FM SOUND
27 *          2: C5 : TELETEXT NEWSFLASH
27 *          3: C6 : TELETEXT SUBTITLES
27 *          4: LANGUAGE A/B (TV)
27 *          5: WIDE-PSEUDO
27 *          6: NO TELETEXT ACKNOWLEDGE
27 *          7: POWER UP IN STANDBY
27
27 .....
27 *
27 *   OSD RAM allocation.
27 *
27 .....
27
27 0000007b CAS1 RMB 1      ROW 1, colour 1/2 & outline enable
27 0000007c RAD1 RMB 1      Row address & character size
27 0000007d CCR1 RMB 1      Window colour & end column
27 0000007e CAS2 RMB 1      ROW 2, colour 1/2 & outline enable
27 0000007f RAD2 RMB 1      Row address & character size
27 00000080 CCR2 RMB 1      Window colour & end column
27 00000081 CAS3 RMB 1      ROW 3, colour 1/2 & outline enable
27 00000082 RAD3 RMB 1      Row address & character size
27 00000083 CCR3 RMB 1      Window colour & end column
27 00000084 CAS4 RMB 1      ROW 4, colour 1/2 & outline enable
27 00000085 RAD4 RMB 1      Row address & character size
27 00000086 CCR4 RMB 1      Window colour & end column
27 00000087 CAS5 RMB 1      ROW 5, colour 1/2 & outline enable
27 00000088 RAD5 RMB 1      Row address & character size
27 00000089 CCR5 RMB 1      Window colour & end column
27 0000008a CAS6 RMB 1      ROW 6, colour 1/2 & outline enable
27 0000008b RAD6 RMB 1      Row address & character size
27 0000008c CCR6 RMB 1      Window colour & end column
27 0000008d CAS7 RMB 1      ROW 7, colour 1/2 & outline enable
27 0000008e RAD7 RMB 1      Row address & character size
27 0000008f CCR7 RMB 1      Window colour & end column
27 00000090 CAS8 RMB 1      ROW 8, colour 1/2 & outline enable
27 00000091 RAD8 RMB 1      Row address & character size
27 00000092 CCR8 RMB 1      Window colour & end column
27
27 00000093 OSDL RMB 1      CURRENT OSD ROW POINTER
27 00000094 LIND RMB 1      ROW TABLE INDEX
27 00000095 BROW RMB 1      CHARACTER FLASH ROW
27 00000096 BCOL RMB 1      CHARACTER FLASH COLUMNS
27 00000097 WROW RMB 1      WINDOW FLASH ROW
27 00000098 ROW1 RMB 1      FIRST ROW No. (NAME TABLE)
27
27 00000099 ANAL RMB 1
27 0000009a ANAF RMB 1
27
27 0000009b TMP1 RMB 1
27 0000009c TMP2 RMB 1
27
27 0000009d RMB 12      UNUSED
27
27 000000a9 STACK RMB 22     23 BYTES USED FOR STACK
27 000000bf SP RMB 1      (1 INTERRUPT AND 9 NEEDED SUBS)
27
27 SECTION .RAM2, COMM
27
27 00000000 DRAM RMB 128
27
27 SECTION .ROM2

```

```

31
32
33
34
35
36
37 00000000 >000037
38 00000003 >0c0074
39 00000006 >1000
40 00000008 >cd0000
41 0000000b a616
42 0000000d >b700
43 0000000f >1900
44 00000011 >1900
45 00000013 >1f00
46 00000015 >0b0008
47 00000018 >1b00
48 0000001a >040003
49 0000001d >cd0000
50 00000020 a6cc
51 00000022 >b700
52 00000024 a646
53 00000026 >b700
54 00000028 >b600
55 0000002a >b700
56 0000002c >cc0000
57
58 0000002f >0c0048
59 00000032 >1800
60 00000034 >090003
61 00000037 >cd0000
62 0000003a >1100
63 0000003c >cd0000
64 0000003f >1100
65 00000041 a610
66 00000043 >b700
67 00000045 >1900
68 00000047 >1b00
69 00000049 a603
70 0000004b >b700
71 0000004d >b700
72 0000004f >3f00
73 00000051 >cd0000
74 00000054 a602
75 00000056 >cc0000
76
77 00000059 >a602
78 0000005b a139
79 0000005d 221b
80 0000005f a130
81 00000061 2317
82 00000063 >a601
83 00000065 a139
84 00000067 2211
85 00000069 a130
86 0000006b 2504
87 0000006d >a600
88 0000006f a137
89 00000071 2207
90 00000073 a130
91 00000075 2503
92 00000077 >b700
93 00000079 81
94 0000007a 99
95 0000007d 81

```

.....
*
* Teletext/TV switching.
*
.....

```

TVTX BRSET 0,STAT,TXTOFF
TXTON BRSET 6,STAT7,PANIC TELETEXT CHIP ON BUS ?
      BSET 0,STAT TELETEXT MODE
      JSR OSDLE
      LDA #S16 CCT, 312/312 SYNC
      STA R1 ENABLING GHOST ROWS
      BCLR 4,STAT ABORT TRANSIENTS
      BCLR 4,STAT2 KILL UPDATES
      BCLR 7,STAT2 NOT MIXED
      BRCLR 5,STAT,NOTT
      BCLR 5,STAT
      BRSET 2,STAT,NOTT
      JSR NOTTH
      LDA #SCC
NOTT STA R5
      LDA #446
      STA R6
      LDA ACC
      STA R4
      JMP TRAN2
UPDATE BRSET 6,STAT7,PANIC TELETEXT CHIP ?
      BSET 4,STAT2 UPDATE ON
      BRCLR 4,STAT,TXTOFF TRANSIENT HOLD ?
      JSR NOTTH YES, RESTART
      BCLR 0,STAT TV MODE
      JSR OSDLE
      BCLR 0,STAT TV MODE
      LDA #S10 BROADCAST, 312/313 SYNC
      STA R1 ENABLING GHOST ROWS
      BCLR 4,STAT ABORT TRANSIENTS
      BCLR 5,STAT ABORT TIME TIMEOUT
RST LDA #S03 506 FOR TRANSIENTS
      STA R5
      STA R6
      CLR R7
      JSR TXT2
      LDA #2
      JMP SPM
TEST LDA PAGO+2,X
      CMP #S39
      BHI PANIC
      CMP #S30
      BLO PANIC
      LDA PAGO+1,X
      CMP #S39
      BHI PANIC
      CMP #S30
      BLO PANIC
      LDA PAGO,X
      CMP #S37
      BHI PANIC
      CMP #S30
      BLO PANIC
      STA PAGE
ABO RTS OK, CARRY CLEAR
PANIC SEC NOT OK, CARRY SET
      RTS

```

```

97
98
99
100
101
102
103 0000007c >0d0003 DIGITO BRCLR 6,STAT,DIGIT
104 0000007f >cc0000 JMP DIGITS
105 00000082 >1700 LDA 3,R3 HOLD DURING
106 00000084 >b600 LDA ACC ENTRY
107 00000086 >cd0000 JSR UP
108 00000089 a604 LDA #4
109 0000008b >cd0000 JSR SPH
110 0000008e >1400 BSET 2,STAT SET HOLD FLAG
111 00000090 >b600 LDA W2
112 00000092 a010 SUB #16
113 00000094 >b600 LDX PDP
114 00000096 z606 BNE NOCH NOT HUNDREDS SO DON'T CHANGE
115 00000098 a107 CMP #7 YES, MORE THAN 7 ?
116 0000009a z302 BLS NOCH NO, SO DON'T CHANGE
117 0000009c a008 SUB #8 YES, 8->0 & 9->1
118 0000009e ab30 NOCH ADD #530 CONVERT TO ASCII
119 000000a0 >b700 STA PAGE,X
120 000000a2 a302 CPX #2 UNITS ?
121 000000a4 z70e BEQ CLRPPD YES, SO CLEAR PDP
122 000000a6 a62d LDA #52D DASH
123 000000a8 ab01 CPX #1 TENS ?
124 000000aa z702 BEQ TEN YES, SO LEAVE TENS
125 000000ac >b701 STA PAGE+1 CLEAR TENS
126 000000ae >b702 STA PAGE+2 CLEAR UNITS
127 000000b0 >3c00 INC PDP
128 000000b2 z002 BRA DPCN
129 000000b4 >3f00 CLRPPD CLR PDP
130 000000b6 >b600 DPGN LDA R4
131 000000b8 >b700 STA R8
132 000000ba >3f00 CLR R9 ROW 0
133 000000bc a602 LDA #2 COLUMN 2
134 000000be >b700 STA R10 P
135 000000c0 a650 LDA #550
136 000000c2 >b700 STA R11
137 000000c4 >b600 LDA PAGE
138 000000c6 >b700 STA PH
139 000000c8 >b601 LDA PAGE+1
140 000000ca >b700 STA PT
141 000000cc >b602 LDA PAGE+2
142 000000ce >b700 STA PU
143 000000d0 >cd0000 JSR TXT138
144 000000d3 >cd0000 LDA TRAN1
145 000000d6 >b600 LDA PDP
146 000000d8 z63f BNE ABO
147 000000da a606 LDA #6
148 000000dc >cd0000 JSR NOBX
149 000000df >b600 LDA PAGE
150 000000e1 >b700 STA PH
151
152
153
154
155
156
157
158 000000e3 >cd0000 GETIT JSR SRCH IS PAGE ALREADY IN ?
159 000000e6 z545 BLO LPT2 YES
160 000000e8 ad23 BSR INDX DISPLAY CHAPTER
161 000000ea >b600 LDA PAGE PAGE HUNDREDS
162 000000ec >b700 STA PAGE,X SAVE IN RAM
163 000000ee >b601 LDA PAGE+1 PAGE TENS
164 000000f0 >e701 STA PAGE+1,X SAVE IN RAM
165 000000f2 >b700 STA C1 PAGE REQUEST TENS
166 000000f4 >b602 LDA PAGE+2 PAGE UNITS
167 000000f6 >e702 STA PAGE+2,X SAVE IN RAM
168 000000f8 >b700 STA C2 PAGE REQUEST UNITS
169 000000fa >b600 LDA PAGE PAGE HUNDREDS
170 000000fc a018 SUB #518
171 000000fe >b700 STA R3 PAGE REQUEST HUNDREDS
172 0000100 >b600 LDA R4
173 0000102 >cd0000 JSR UP
174 0000105 >cd0000 JSR TXT1 REQUEST IT
175 0000108 >1500 BCLR 2,STAT RESET HOLD FLAG
176 000010a >cc0000 JMP SFND WRITE ONE TO FOUND
177
178
179 0000010d >b600 INDX LDA ACC
180 0000010f 48 LSLA x2
181 00000110 >bb00 ADD ACC x3
182 00000112 97 TAX
183 00000113 81 YIP RTS
184
185 00000114 48 UP LSLA
186 00000115 48 LSLA
187 00000116 48 LSLA
188 00000117 48 LSLA
189 00000118 >b700 STA R2
190 0000011a 81 RTS

```

```

192
193
194
195
196
197
198 0000011b >3f00
199 0000011d >06000b
200
201 00000120 >c40000
202 00000123 >c40000
203 00000126 252c
204 00000128 >c00000
205 0000012b >b601
206 0000012d 2025
207
208 0000012f >3f00
209 00000131 >06000b
210
211 00000134 >c40000
212 00000137 >c40000
213 0000013a 2518
214 0000013c >c00000
215 0000013f 0c0103
216 00000142 >010061
217 00000145 >b602
218 00000147 200b
219
220 00000149 >07005a
221 0000014c 0c0103
222 0000014f >030054
223 00000152 >b603
224 00000154 >b700
225 00000156 48
226 00000157 >bb00
227 00000159 9f
228 0000015a >c40000
229 0000015d 2547
230 0000015f >b600
231 00000161 >b100
232 00000163 2604
233 00000164 1400
234 00000167 2009
235 00000169 >040003
236 0000016c >c40000
237 0000016f >c40000
238 00000172 >3f00
239 00000174 >050003
240 00000177 >c00000
241 0000017a a60f
242 0000017c >b700
243 0000017e >b600
244 00000180 >b700
245 00000182 >b700
246 00000184 >c40000
247 00000187 >1500
248 00000189 >c00000
249
250
251
252
253
254
255
256 0000018c ae0f
257 0000018e >c40000
258 00000191 2414
259 00000193 >c00000
260
261 00000196 >07000d
262 00000199 0c0103
263 0000019c >050007
264 0000019f ae0c
265 000001a1 >c40000
266 000001a4 2401
267 000001a6 81
268
269 000001a7 >1400
270 000001a9 >3f00
271 000001ab >e602
272 000001ad >b700
273 000001af >b700
274 000001b1 >e601
275 000001b3 >b700
276 000001b5 >b700
277 000001b7 >e600
278 000001b9 >b700
279 000001bb a018
280 000001bd >b700
281 000001bf >c40000
282 000001c2 >b600
283 000001c4 >e700
284 000001c6 >b600
285 000001c8 >e701
286 000001ca >b600
287 000001cc >e702
288 000001ce >b600
289 000001d0 >b700
290 000001d2 >c40000
291 000001d5 a650
292 000001d7 >3f00
293 000001d9 >3f00
294 000001db a602
295 000001dd >b700
296
297 000001df >1500
298 000001e1 >c40000
299 000001e4 >c40000
300 000001e7 >c40000
301 000001ea >c00000

```

```

*****
*
* Red, Green & Yellow keys.
*
*****
RED CLR PDP
BRSET 3,STAT3,RED2 LINKS ON ?

NPAGE JSR INDXP
JSR NOTOK3 NO, SO FORCE AN INCREMENT
BLO LPT ALREADY REQUESTED ?
JMP CLRPD NO, GETIT

RED2 LDA ACC+1
LPT2 BRA LPT

GREEN CLR PDP
BRSET 3,STAT3,GLOK LINKS ON ?

PPAGE JSR INDXP
JSR NOTOK2 NO, SO FORCE A DECREMENT
BLO LPT ALREADY REQUESTED ?
JMP CLRPD NO, GETIT
GLOK BRSET 6,PORTB,IG0 GYC BITS ENABLED ?
BRCLR 0,STAT3,ABC GREEN LINK ON ?
IGO LDA ACC+2
BRA LPT

YELLOW BRCLR 3,STAT3,ABC LINKS ON ?
BRSET 6,PORTB,IG1 GYC BITS ENABLED ?
BRCLR 1,STAT3,ABC YELLOW LINKS ON ?

IG1 LDA ACC+3
LPT STA W3

LSLA W3 X2
ADD W3 X3 FOR PAGE POINTER
TAX
JSR TEST IS PAGE No. OK ?
BCS ABC IF NOT ABORT
LDA W3 ACC No
CMP ACC IF SAME ACC CCT
BNE MTSAC THEN FORCE UNSTOP
BSET 2,STAT3
BRA CARO

NTSAC BRCLR 6,STAT,SKOSP SUB-PAGE MODE ?
JSR OUTSP YES, ABANDON IT
SKOSP JSR RSTR PUT PAGE No. BACK
CARO CLR PDP
COK BRCLR 2,STAT,NOTHLD IF OLD PAGE ON HOLD
NOHLD JSR NOHOLD CANCEL HOLD
LDA #50F CORRUPT C6 FOR UPDATE
STA C6
LDA W3
STA R4
STA ACC
JSR CFND CHECK PBLE, IF HIGH DO NOTHING
BCLR 2,STAT IF LOW (PAGE FOUND) CLEAR FOUND
JMP TXT2 TO FORCE FETCHING OF LINKS.

*****
*
* Index & Cyan keys.
*
*****
INDEX LDX #15
JSR TEST
BCC IAC
JMP GIP

CYAN BRCLR 3,STAT3,ABC LINKS ON ?
BRSET 6,PORTB,IG2 GYC BITS ENABLED ?
BRCLR 2,STAT3,ABC CYAN LINK ON ?

IG2 LDX #12
JSR TEST
BCC IAC
ABC RTS

IAC BCLR 6,STAT RESET PAGE MODE
CLR PDP
LDA PAGO+2,X
STA PU
STA C2
LDA PAGO+1,X
STA PT
STA C1
LDA PAGO,X
STA PH
SUB #518
STA R3
JSR INDX
LDA PH
STA PAGO,X
LDA PT
STA PAGO+1,X
LDA PU
STA PAGO+2,X
LDA ACC
STA R8
JSR UP
LDA #550
STA R11
CLR R9
LDA #2
STA R10

CYOK BCLR 2,STAT RESET HOLD FLAG
JSR TXT38
JSR TRAM1 DISPLAY TOP ROW
JSR SFND SET FOUND
JMP TXT1

```

303
304
305
306
307
308
309 000001ed >b600
310 000001ef ab04
311 000001fi >b700
312 000001f3 >3f00
313 000001f5 a601
314 000001f7 >b700
315 000001f9 a6ff
316 000001fb >b701
317 000001fd >b702
318 000001ff >b703
319 00000201 >cd0000
320 00000204 >3c00
321 00000206 >b600
322 00000208 >b700
323 0000020a a443
324 0000020c 2406
325 0000020e >b800
326 00000210 >e700
327 00000212 2003
328 00000214 >cd0000
329 00000217 >b600
330 00000219 ab06
331 0000021b >b700
332 0000021d >b600
333 0000021f a103
334 00000221 25e1
335
336 00000223 >cd0000
337 00000226 >3f00
338 00000228 a604
339 0000022a >b700
340
341 0000022c >3a00
342 0000022e >3f00
343 00000230 >e600
344 00000232 a1ff
345 00000234 2612
346 00000236 >cd0000
347 00000239 >b600
348 0000023b >cd0000
349 0000023e >be00
350 00000240 >e700
351 00000242 >cd0000
352 00000245 >cd0000
353 00000248 >b600
354 0000024a a101
355 0000024c 22de
356
357 0000024e 01
358
359
360
361
362
363
364
365 0000024f >b600
366 00000251 a113
367 00000253 2503
368 00000255 >f006c
369 00000258 a610
370 0000025a >cd0000
371 0000025d >b601
372 0000025f >cd0000
373 00000262 >b700
374 00000264 >b600
375 00000266 >cd0000
376 00000269 >b700
377 0000026b >b600
378 0000026d >b700
379 0000026f >cd0000
380 00000272 >e600
381 00000274 >b700
382
383 00000276 >cd0000
384 00000279 >070059
385 0000027c >000004
386 0000027f >1000
387 00000281 2002
388 00000283 >1100
389 00000285 >cd0000
390 00000288 >050009
391 0000028b >020004
392 0000028e >1200
393 00000290 >2002
394 00000292 >1300
395 00000294 >070009
396 00000297 >040004
397 0000029a >1400
398 0000029c 2002
399 0000029e >1500
400 000002a0 >cc0000
401
402 000002a3 >cd0000
403 000002a6 >b601
404 000002a8 >cd0000
405 000002ab >b700
406 000002ad >b600
407 000002af >cd0000
408 000002b2 >b700
409 000002b4 20c0
410
411 000002b6 >cd0000
412 000002b9 a018
413 000002bb >b700
414 000002bd a604
415 000002bf >cc0000

```

.....
*
*   Get linked page nos & allocate to accs.
*
.....
LINK   LDA   ACC   CHAPTER
      ADD   #4     ADD 4 FOR GHOST ROWS
      STA   #8
      CLR   COUNT
      LDA   #1
      STA   W3
      LDA   #FFF
      STA   ACC+1
      STA   ACC+2
      STA   ACC+3
      JSR   INDXP   LOOP ROUND RED, GREEN & YELLOW
      INC   COUNT
      LDA   R10
      BSR   GLP1   GET LINKED PAGE NO.
      BHS   NOTFND ALREADY IN RAM ?
      LDX   COUNT  YES, SAVE ACC NO.
      STA   ACC,X  AGAINST COLOUR
      BRA   NEXTC
      JSR   PUSH   NOT IN RAM, SO SAVE
      LDA   W3     PAGE NUMBER IN LIFO
      ADD   #6
      STA   W3     NEXT LINK
      LDA   COUNT
      CHP   #3     ALL DONE ?
      BLO   LLOOP
      JSR   GCYI   GET CYAN AND INDEX LINKS
      CLR   WACC
      LDA   #4
      STA   COUNT
      LLOOP DEC   COUNT
            LDX   COUNT
            LDA   ACC,X
            CHP   #FFF  IF STILL AN ACC AT SFF THEN
            BME   ALOC  RECOVER PAGE NO. FROM LIFO
            JSR   PULL
            LDA   WACC
            JSR   CHCK1 ALREADY USED ? IF SO INCREMENT
            LDX   COUNT
            STA   ACC,X
            JSR   UP
            JSR   GLP2
      ALOC  LDA   COUNT
            CHP   #501
            BHI   LLOOP
            RTS
.....
*
*   Fetch linked page & magazine numbers.
*
.....
GLP1   LDA   R10
      CHP   #19   IF INDEX IGNORE LINK CONTROL
      BHI   COB
      BRCLR #3,STAT3,NOTOK LINKS OK ?
      LDA   #16   YES, ROW 16 FOR LINKED PAGES
      R2B   #2B   FETCH 2 LINK BYTES
      LDA   IOBUF+1
      JSR   DECODE DECODE UNITS
      STA   W2
      LDA   IOBUF
      JSR   DECODE DECODE TENS
      STA   PT
      LDA   W2
      STA   PU
      JSR   INDX  CHECK FOR ZERO ?
      LDA   PAG0,X  FETCH CURRENT MAG. NO.
      STA   PH     PAGE HUNDREDS
R2BJ1  JSR   RADIO
      BRCLR #3,IOBUF,OK0  MAG BIT ZERO OK ?
      BRSET #0,PH,H1     NO, SO TOGGLE
      BSET  #0,PH
      BRA   OK0
      BCLR #0,PH
      JSR   RADIO
      BRCLR #2,IOBUF,OK1  MAG BIT ONE OK ?
      BRSET #1,PH,PT1    NO, SO TOGGLE
      BSET  #1,PH
      BRA   OK1
      BCLR #1,PH
      BRCLR #3,IOBUF,OK2  MAG BIT TWO OK ?
      BRSET #2,PH,PU1    NO, SO TOGGLE
      BSET  #2,PH
      BRA   OK2
      BCLR #2,PH
      OK2  JMP   SRCH
R2BJ2  JSR   R2B
      LDA   IOBUF+1
      JSR   DECODE DECODE UNITS
      STA   FU
      LDA   IOBUF
      JSR   DECODE DECODE TENS
      STA   PT
      BRA   R2BJ1
NOTTH  JSR   REL1
      SUB   #518
      STA   R3
      LDA   #4
      JMP   SPM

```



```

417
418
419
420
421
422
423 000002c2 >d100
424 000002c4 >cd0000
425 000002c7 >e600
426 000002c9 >b700
427 000002cb >b602
428 000002cd 4c
429 000002ce >b700
430 000002d0 >b702
431 000002d2 a139
432 000002d4 2312
433 000002d6 a630
434 000002d8 >b700
435 000002da >b702
436 000002dc >3c01
437 000002de >b601
438 000002e0 a139
439 000002e2 2304
440 000002e4 a630
441 000002e6 >b701
442 000002e8 >b601
443 000002ea >b700
444 000002ec 20b2
445
446 000002ee >d100
447 000002f0 >e600
448 000002f2 >b700
449 000002f4 >b602
450 000002f6 4a
451 000002f7 >b700
452 000002f9 >b702
453 000002fb a130
454 000002fd 24e9
455 000002ff a639
456 00000301 >b700
457 00000303 >b702
458 00000305 >3a01
459 00000307 >b601
460 00000309 >b700
461 0000030b 24db
462 0000030d a639
463 0000030f 20d5
464
465
466
467
468
469
470
471 00000311 >b700
472 00000313 44
473 00000314 79
474 00000315 44
475 00000316 >b700
476 00000318 44
477 00000319 >bb000
478 0000031b 97
479 0000031c >b600
480 0000031e >b700
481 00000320 >7e2
482 00000322 >b600
483 00000324 >b700
484 00000326 >e701
485 00000328 >b600
486 0000032a >b700
487 0000032c a018
488 0000032e >b700
489
490 00000330 a309
491 00000332 221c
492 00000334 a650
493 00000336 >b700
494 00000338 >b600
495 0000033a ab08
496 0000033c >b700
497 0000033e >3f00
498 00000340 a602
499 00000342 >b700
500 00000344 >b600
501 00000346 a139
502 00000348 2206
503 0000034a >b600
504 0000034c a139
505 0000034e 2301
506 00000350 81
507
508 00000351 >cd0000
509 00000353 a606
510 00000356 >cd0000
511 00000359 a1700
512 0000035b >cd0000
513 0000035e >cd0000
514 00000361 >cc0000
515
516 00000364 ae08
517 00000366 >e600
518 00000368 >e700
519 0000036a 5a
520 0000036b 2af9
521 0000036d 81

```

```

*****
*
*      New bits for default (+1 & -1) links.
*
*****

```

```

NOTOK3 BCLR 6,STAT      CANCEL SUB-PAGE
NOTOK  JSR  INDX
        LDA  PAG0,X
        STA  PH
        LDA  PAGE+2
        INCA
        STA  PU
        STA  PAGE+2
        CMP  #839
        BLS  NOV9
        LDA  #830
        STA  PU
        STA  PAGE+2
        INC  PAGE+1
        LDA  PAGE+1
        CMP  #839
        BLS  NOV9
        LDA  #830
        STA  PAGE+1
NOV9A  LDA  PAGE+1
NOV9   LDA  PAGE+1
        STA  PT
        BRA  OK2

```

```

NOTOK2 BCLR 6,STAT      CANCEL SUB-PAGE
        LDA  PAG0,X
        STA  PH
        LDA  PAGE+2
        DECA
        STA  PU
        STA  PAGE+2
        CMP  #830
        BHS  NOV9
        LDA  #839
        STA  PU
        STA  PAGE+2
        DEC  PAGE+1
        LDA  PAGE+1
        CMP  #830
        BHS  NOV9
        LDA  #839
        BRA  NOV9A

```

```

*****
*
*      Request new linked page.
*
*****

```

```

GLP2  STA  R2
        LSRA
        LSRA
        LSRA
        STA  C2          x2
        LSRA
        ADD  C2          x3
        TAX          X <- 3 x ACC NO.
        LDA  PU
        STA  C2
        STA  PAG0+2,X
        LDA  PT
        STA  C1
        STA  PAG0+1,X
        LDA  PH
        STA  PAG0,X
        SUB  #818
        STA  R3
        CPY  #9
        BHI  ABORT
        LDA  #850
        STA  R11
        LDA  WACC
        ADD  #808
        STA  R8          ACC
                          CLEAR CHAPTER
        CLR  R9          INTO IIC
        LDA  #2          ROW 0
        STA  R10        COLUMN 2
        LDA  C2
        CMP  #839
        BHI  ABORT
        LDA  C1
        CMP  #839
        BLS  LOK
ABORT  RTS
LOK   JSR  TXT3      CLEAR CHAPTER
        LDA  #6      WAIT
        JSR  TPAUZ   FOR IT
        BCLR BCLR 3,R8  DON'T CLEAR THIS TIME
        JSR  TXT3#   PUT PAGE NUMBER IN CHAPTER
        JSR  SFND    SET FOUND FLAG
        JMP  TXT1L   AND REQUEST IT
PUSH  LDX  #8
PSHL  LDA  PH,X
        STA  LIFO,X
        DECB
        BPL  PSHL
        RTS

```

523
524
525
526
527
528

529 0000036e >3f00
530 00000370 >b600
531 00000372 48
532 00000373 >bb00
533 00000375 97
534 00000376 >e600
535 00000378 >b100
536 0000037a 260c
537 0000037c >e601
538 0000037e >b100
539 00000380 2606
540 00000382 >e602
541 00000384 >b100
542 00000386 2708
543 00000388 >3c00
544 0000038a >b600
545 0000038c a104
546 0000038e 25e0
547
548 00000390 >b600
549 00000392 a104
550 00000394 81

551
552
553
554
555
556
557

558 00000395 >3c00
559 00000397 5f
560 00000398 >b600
561 0000039a >e100
562 0000039c 27f7
563 0000039e 5c
564 0000039f a304
565 000003a1 25f5
566 000003a3 81
567
568 000003a4 >3c00
569 000003a6 >3c00
570 000003a8 >cd0000
571 000003ab >b600
572 000003ad >cd0000
573 000003b0 >b700
574 000003b2 81
575
576
577
578
579
580
581

582
583
584
585
586
587
588
589

590 000003bf a620
591 000003c1 >b700
592 000003c3 >b700
593 000003c5 >0700c8
594
595 000003c8 >b601
596 000003ca >b700
597 000003cc >b600
598 000003ce >b700
599 000003d0 >b600
600 000003d2 >b700
601 000003d4 a618
602 000003d6 ad31
603 000003d8 23db
604

605
606
607
608
609
610
611
612

613 000003e9 >cd0000
614 000003ec 250f
615 000003ee >1900
616 000003f0 20ea
617
618 000003f2 >cd0000
619 000003f5 >e601
620 000003f7 >b701
621 000003f9 >e602
622 000003fb >b702
623 000003fd 81
624

625
626
627
628
629
630
631

625 000003fe 5f
626 000003ff >e600
627 00000401 >e700
628 00000403 5c
629 00000404 a308
630 00000406 25f7
631 00000408 81

.....
*
* Is page already in RAM ?
*
.....

SRCH CLR WACC
LOOPS LDA WACC
LDA LSLA
ADD WACC
TAX
LDA PAGO,X
CMP PH
* BNE FINI
* LDA PAGO+1,X
CMP PT
BNE FINI
LDA PAGO+2,X
CMP PU
BEQ FND2
FINI INC WACC
LDA WACC
CMP #4
BLO LOOPS
FND2 LDA WACC
CMP #4
RTS

IF MATCH THEN CHECK FOR
SUB-PAGE MATCH (SHOULD
DISPLAY PAGE BE DIFFERENT)

.....
*
* Is Acquisition circuit in use ?
*
.....

SAM INC WACC
CHCK1 CLRX
CHCK2 LDA WACC
CMP ACC,X
BEQ SAM
INCX
CPX #4
BLO CHCK2
RTS

RADIO INC R10
INC R10
JSR R2B89
LDA IOBUF
JSR DECODE
STX IOBUF
DDI RTS

.....
*
* Transfer ghost row 20 to display row 24.
* * set found flag.
*
.....

ROW24 CLR R10
MRE LDA ACC CHAPTER
ADD #4 ADD 4 FOR GHOST ROWS
STA R8
LDA #20 ROW 20
BSR R2B

LDA #520 SPACE
STA R11
STA PH
BRCLR 3,STAT3,BLANK ROW24 ENABLED ?

LDA IOBUF+1 YES, SO USE DATA
STA R11
LDA IOBUF
STA PH
BLANK LDA ACC BACK TO
STA R8 DISPLAY CHAPTER
LDA #24
BSR W2B
BLS MRE

SFND BSET 4,R11 SET FOUND FLAG
SFND2 LDA #25 WRITE IT
STA R9 ROW
LDA #8
STA R10 COLUMN
LDA #5
JMP TXT32

CFND JSR CPBLF
BCS ABCF
BCLR 4,R11 CLEAR FOUND FLAG
BRA SFND2

INDXP JSR INDX
LDA PAGO+1,X
STA PAGE+1
LDA PAGO+2,X
STA PAGE+2
ABCF RTS

FULL CLRX
FLLL LDA LIFO,X
STA PH,X
INCX
CPX #9
BLO FLLL
RTS

```

633
634
635
636
637
638
639
640
641 00000409 >b700          W2B  STA  R9          ROM 24
642 0000040b a606          LDA  #6
643 0000040d >cd0000      JSR  TXT32
644 00000410 >3c00          INC  R10
645 00000412 >3c00          INC  R10
646 00000414 >b600          LDA  R10
647 00000416 a126          CMP  #38
648 00000418 81
649
650 00000419 >b700          R2B  STA  R9          ROM
651 0000041b a608          LDA  #8
652 0000041d >b700          R2BN9 STA SUB3
653 0000041f a604          LDA  #4
654 00000421 >b700          STA  W1
655 00000423 >a#00          LDX  #SUB3
656 00000425 >cd0000      JSR  SEND22
657 00000428 42          MUL
658 00000429 42          MUL
659 0000042a a60b          LDA  #11
660 0000042c >b700          STA  SUBADDR
661 0000042e a622          LDA  #22
662 00000430 >b700          READ22 STA ADDR
663 00000432 >cc0000      JMP  READ
664
665 00000435 a613          GCYI LDA  #19          CYAN
666 00000437 >b700          STA  R10
667 00000439 >cd0000      JSR  GLP1
668 0000043c a640          LDA  #40
669 0000043e >cd0000      JSR  GLP2
670 00000441 a61f          LDA  #31          INDEX
671 00000443 >b700          STA  R10
672 00000445 >cd0000      JSR  GLP1
673 00000448 a650          LDA  #50
674 0000044a >cc0000      JMP  GLP2
675
676 0000044d >b600          CLINK LDA ACC
677 0000044f ab04          ADD  #4
678 00000451 >b700          STA  R8
679 00000453 >3f00          CLR  STAT3
680 00000455 >3f00          CLR  R10
681 00000457 a610          LDA  #16
682 00000459 >cd0000      JSR  R2B
683 0000045c >b601          LDA  IOBUF+1      DESTINATION BYTE
684 0000045e 260e          BNE  NPK27        IF NOT ZERO, NO PK27
685 00000460 a625          LDA  #25          CHAIN CONTROL BYTE
686 00000462 >b700          STA  R10
687 00000464 a610          LDA  #16
688 00000466 adb1          BSR  R2B
689 00000468 >b601          LDA  IOBUF+1
690 0000046a ad03          BSR  DECODE
691 0000046c >b700          STX  STAT3
692 0000046e 81          NPK27 RTS
693
694
695
696
697
698
699
700 0000046f >b700          DECODE STA W1
701 00000471 5f          CLRX
702 00000472 >d60000      TRA  LDA  HAM,X
703 00000475 >b100          CMP  W1
704 00000477 2732          BEQ  FNDJ
705
706 00000479 >b700          TRZE STA SUB2
707 0000047b >0000004      BRSET 0,SUB2,ZE1
708 0000047d >1000          BSET 0,SUB2
709 00000480 2002          BRA  ZE1+2
710 00000482 >1100          ZE1  BCLR 0,SUB2
711 00000484 >cd0000      JSR  SSUB
712 00000487 2722          BEQ  FNDJ
713
714 00000489 >d60000      TRON LDA HAM,X
715 0000048c >b700          STA  SUB2
716 0000048e a200004      BRSET 1,SUB2,ON1
717 00000491 >1200          BSET 1,SUB2
718 00000493 2002          BRA  ON1+2
719 00000495 >1300          ON1  BCLR 1,SUB2
720 00000497 ad7a          BSR  SSUB
721 00000499 2774          BEQ  FND
722
723 0000049b >d60000      TRTW LDA HAM,X
724 0000049e >b700          STA  SUB2
725 000004a0 >0400004      BRSET 2,SUB2,TW1
726 000004a3 >1400          BSET 2,SUB2
727 000004a5 2002          BRA  TW1+2
728 000004a7 >1500          TW1  BCLR 2,SUB2
729 000004a9 ad68          BSR  SSUB
730 000004ab 2762          FNDJ BEQ  FND
731
732 000004ad >d60000      TRTH LDA HAM,X
733 000004b0 >b700          STA  SUB2
734 000004b2 >0600004      BRSET 3,SUB2,TH1
735 000004b5 >1600          BSET 3,SUB2
736 000004b7 2002          BRA  TH1+2
737 000004b9 >1700          TH1  BCLR 3,SUB2
738 000004bb ad56          BSR  SSUB
739 000004bd 2750          BEQ  FND

```

```

741
742
743
744
745
746
747 000004bf >d60000
748 000004c2 >b700
749 000004c4 >080004
750 000004c7 >1800
751 000004c9 2002
752 000004cb >1900
753 000004cd ad44
754 000004cf 273e
755
756 000004d1 >d60000
757 000004d4 >b700
758 000004d6 >0a0004
759 000004d9 >1a00
760 000004db 2002
761 000004dd >1b00
762 000004df ad32
763 000004e1 272c
764
765 000004e3 >d60000
766 000004e6 >b700
767 000004e8 >0c0004
768 000004eb >1c00
769 000004ed 2002
770 000004ef >1d00
771 000004f1 ad20
772 000004f3 271a
773
774 000004f5 >d60000
775 000004f8 >b700
776 000004fa >0e0004
777 000004fd >1e00
778 000004ff 2002
779 00000501 >1f00
780 00000503 ad0e
781 00000505 2708
782
783 00000507 5c
784 00000508 a30f
785 0000050a 2203
786 0000050c >cc0000
787 0000050f >d60000
788 00000512 81
789
790 00000513 >b600
791 00000515 >b100
792 00000517 81
793
794
795
796
797
798
799
800 00000518 >0e0015
801 0000051b >1e00
802 0000051d a610
803 0000051f >b700
804 00000521 a606
805 00000523 >cd0000
806 00000526 a66e
807 00000528 >b700
808 0000052a a617
809 0000052c >b700
810 0000052e 2015
811
812 00000530 >1f00
813 00000532 a616
814 00000534 >b700
815 00000536 a6cc
816 00000538 >b700
817 0000053a a646
818 0000053c >b700
819 0000053e 2005
820
821 00000540 a606
822 00000542 >cd0000
823 00000545 a602
824 00000547 >cd0000
825 0000054a 4f
826 0000054b >cd0000
827 0000054d >1800
828 00000550 ad15
829 00000552 a606
830 00000554 >b700
831 00000556 a607
832 00000558 >b700
833
834 0000055a a605
835 0000055c >b700
836 0000055e a604
837 00000560 >b700
838 00000562 >ae00
839 00000564 >cc0000
840
841
842 00000567 a619
843 00000569 >b700
844 0000056b a606
845 0000056d >b700
846 0000056f >b600
847 00000571 >b700
848 00000573 >3f00
849 00000575 a605
850 00000577 >cc0000

```

```

*****
*
*   More Hamming decode.
*
*****
TRFO  LDA   HAM,X
      STA   SUB2
      BRSET 4,SUB2,FO1
      BSET  4,SUB2
      BRA   FO1+2
FO1   BCLR  4,SUB2
      BSR   SSUB
      BEQ   FND
TRFI  LDA   HAM,X
      STA   SUB2
      BRSET 5,SUB2,FI1
      BSET  5,SUB2
      BRA   FI1+2
FI1   BCLR  5,SUB2
      BSR   SSUB
      BEQ   FND
TRSI  LDA   HAM,X
      STA   SUB2
      BRSET 6,SUB2,S11
      BSET  6,SUB2
      BRA   S11+2
S11   BCLR  6,SUB2
      BSR   SSUB
      BEQ   FND
TRSE  LDA   HAM,X
      STA   SUB2
      BRSET 7,SUB2,SE1
      BSET  7,SUB2
      BRA   SE1+2
SE1   BCLR  7,SUB2
      BSR   SSUB
      BEQ   FND
FND   INXC
      CPX   #50F
      BHI   FND
      JMP   TRA
      LDA   NUM,X
      RTS
SSUB  LDA   SUB2
      CMP   W1
      RTS
*****
*
*   Mix/nomix.
*
*****
MIX   BRSET 7,STAT2,NOMIX  ALREADY MIXED ?
      BSET  7,STAT2        NO, SO MIX IT
      LDA   #810           BROADCAST, 312/312 SYNC
      STA   R1             ENABLING GHOST ROWS
      LDA   #806
      JSR   NOBX
      LDA   #846
      STA   R5
      LDA   #817           $46 FOR NOMIX FLASH/SUBT.
      STA   R6
      BRA   TRAN2
NOMIX BCLR  7,STAT2        MIXED, SO NOMIX
      LDA   #816           CCT, 312/312 SYNC
      STA   R1             ENABLING GHOST ROWS
      LDA   #8CC
      STA   R5
      LDA   #846
      STA   R6
      BRA   TRAN2
TRAN1 LDA   #6
      JSR   BOXOOF
TRAN2 LDA   #2
      JSR   SPM           SET-UP SYNC
      CLRA
TRAN3 JSR   BOXOON
      BSET  4,STAT2
      BSR   FR0           FORCE HEADER DISPLAY
      LDA   #6
      STA   TMR          5s TIMER
      LDA   #807
      STA   #807         ENABLE ALL BOXES
      STA   R7
TXT2  LDA   #5
      STA   W1           DISPLAY CONTROL
      LDA   #4
      STA   SUB2
      LDX   #SUB2
      JMP   SEND22
FR0   LDA   #25
      STA   R9           FORCE DISPLAY OF HEADER
      LDA   #6
      STA   R10
      LDA   ACC
      STA   R8
      CLR  R11
      LDA   #5
      JMP   TXT32

```

```

852
853
854
855
856
857
858 0000057a >3f00
859 0000057c >040062
860 0000057f >1400
861 00000581 >b600
862 00000583 >b700
863 00000585 >cd0000
864 00000588 >3f00
865 0000058a >1d00
866 0000058c 5f
867 0000058d ad2b
868 0000058f >b600
869 00000591 >b700
870 00000593 >cd0000
871 00000596 >3f00
872 00000598 >1700
873 0000059a a604
874 0000059c ad11
875 0000059e 20aa
876
877 000005a0 >0c000a
878 000005a3 >3f00
879 000005a5 >3f00
880 000005a7 >3f00
881 000005a9 a60f
882 000005ab >b700
883 000005ad a60a
884 000005af >b700
885 000005b1 a601
886 000005b3 >b700
887 000005b5 >a600
888 000005b7 >cc0000
889
890 000005ba >b700
891 000005bc >3f00
892 000005be 4e
893 000005bf ad07
894 000005c1 >b600
895 000005c3 ab04
896 000005c5 97
897 000005c6 a604
898 000005c8 >b700
899 000005ca >d60000
900 000005cb >b700
901 000005cd >d60001
902 000005d2 >b700
903 000005d4 >d60002
904 000005d7 00
905 000005d9 >d60003
906 000005dc >b700
907 000005de >cc0000
908
909
910
911
912
913
914
915 000005e1 >1500
916 000005e3 >b600
917 000005e5 >b700
918 000005e7 >3f00
919 000005e9 a602
920 000005eb >b700
921 000005ed a650
922 000005ef >b700
923 000005f1 ad0b
924 000005f3 ad14
925 000005f5 >cd0000
926 000005f8 >cd0000
927 000005fb >cc0000
928
929 000005fe >b600
930 00000600 >cd0000
931 00000603 >cd0000
932 00000606 >a600
933 00000608 81
934 00000609 >b700
935 0000060b ad18
936 0000060d >b700
937 0000060f >a601
938 00000611 >b700
939 00000613 >b700
940 00000615 >a602
941 00000617 >b700
942 00000619 >b700
943 0000061b >cc0000
944
945 0000061e >b600
946 00000620 >b700
947 00000622 a609
948 00000624 >b700
949 00000626 a619
950 00000628 >cd0000
951 0000062b 99
952 0000062c >0a0101
953 0000062f 98
954 00000630 81

```

```

*****
*
* Hold.
*
*****
HOLD CLR PDP
BRSET 2,STAT,NOHOLD
BSET 2,STAT
LDA ACC
STA R8
JSR UP
CLR R9 ROW 0
BCLR 6,STAT RESET SUB-PAGE MODE
CLR R9
UCHOLD LDA ACC
STA R8 DISPLAY CHAPTER
JSR UP
CLR R9 ROW 0
BCLR 3,R3 HOLD
LDA #4 WAS TXT1
BSR SPM
BRA TRAN3
TXT1 BRSET 6,STAT,SPM2
TXTIL CLR C3
CLR C4
CLR C5
LDA #50F CORRPUT C6 SO THAT NEXT
STA C6 ARRIVAL IS SEEN BY UPDATE
SPM2 LDA #10
SPM STA W1
LDA #1
STA SUB1
LDX #SUB1
JMP SEND22
DISP# STX W3
CLR R9
CLR# CLR#
BSR DISP4
LDA W3
ADD #4
TAX
LDA #4
DISP4 STA R10
LDA LHOLD,X
STA R11
LDA LHOLD+1,X
STA PH
LDA LHOLD+2,X
STA FT
LDA LHOLD+3,X
STA PU
JMP TXT3
*****
*
* Nohold.
*
*****
NOHOLD BCLR 2,STAT
LDA ACC
STA R8
CLR R9 ROW 0
LDA #2
STA R10 COLUMN 2
LDA #550
STA R11 P
BSR REL1
BSR REL2
JSR TXT3#
JSR SFND
JMP TRAN2
REL1 LDA ACC
JSR UP
JSR INDX
LDA PAG0,X
RTS
REL2 STA PH
SUB #51#
STA R3
LDA PAG0+1,X
STA FT
STA C1
LDA PAG0+2,X
STA PU
STA C2
JMP TXT1
CPBLF LDA ACC
STA R8
LDA #9
STA R10
LDA #25
JSR R2B
SEC
BRSET 5,IOBUF+1,HIGH
CLC
HIGH RTS

```

```

956
957
958
959
960
961
962 00000631 >0a0004 REVEAL BRSET 5,R7,REV
963 00000634 >1a00 BSET 5,R7
964 00000636 2016 BRA OUT
965 00000638 >1b00 REV BCCLR 5,R7
966 0000063a 2012 BRA OUT
967 0000063c >07000b EXPTB BRCLR 3,R7,EXP
968 0000063f >090004 BRCLR 4,R7,BOT
969 00000642 >1700 BCCLR 3,R7 SINGLE HEIGHT
970 00000644 2008 BRA OUT
971 00000646 >1800 BOT BSET 4,R7 BOTTOM
972 00000648 2004 BRA OUT
973 0000064a >1600 EXP BSET 3,R7
974 0000064c >1900 OUT BCCLR 4,R7 TOP
975 0000064e >cc0000 JMP TXT2
976
977 00000651 >0c00dc TIME BRSET 6,STAT7,HIGH TELETEXT CHIP ?
978 00000654 >010003 BRCLR 0,STAT,CLOCK TELETEXT MODE ?
979 00000657 >cc0000 JMP SUBFC YES
980 0000065a >0a0025 CLOCK BRSET 5,STAT,TAO NO, TIME ALREADY ON ?
981 0000065d >b600 LDA ACC
982 0000065f >b700 STA R4
983 00000661 >cd0000 JSR UCHOLD
984 00000664 >1800 BSET 4,STAT
985 00000666 >1a00 BSET 5,STAT
986 00000668 4f CLR A
987 00000669 ad1c BSR NOBX
988 0000066b a61e LDA #30
989 0000066d ad1e BSR BOX00N
990 0000066f >cd0000 JSR FR0
991 00000672 a609 LDA #509
992 00000674 >b700 STA R7
993 00000676 >cd0000 JSR TXT2 STOP FLASHES ON FIRST PRESS
994 00000679 a64c LDA #546
995 0000067b >b700 STA R5
996 0000067d >b700 STA R6
997 0000067f >cd0000 JSR TXT2
998 00000682 a606 TAO LDA #6
999 00000684 >b700 STA TMR
1000 00000686 81 RTS
1001 00000687 >b700 NOBX STA R10
1002 00000689 a620 LDA #520
1003 0000068b 200a BRA BOX
1004 0000068d >b700 BOX00N STA R10
1005 0000068f a60b LDA #50B
1006 00000691 2004 BRA BOX
1007 00000693 >b700 BOX00F STA R10
1008 00000695 a60a LDA #50A
1009 00000697 >b700 BOX STA R11
1010 00000699 >b700 STA PH
1011 0000069b >b600 LDA R4
1012 0000069d >b700 STA R8
1013 0000069f >3f00 CLR R9
1014 000006a1 a606 LDA #6
1015 000006a3 >cc0000 JMP TXT32

```

```

1017
1018
1019
1020
1021
1022
1023 000006a6 >cd0000
1024 000006a9 >b600
1025 000006ab a010
1026 000006ad >be00
1027 000006af 2704
1028 000006b1 4302
1029 000006b3 260f
1030 000006b5 a107
1031 000006b7 2302
1032 000006b9 8008
1033 000006bb 5d
1034 000006bc 2606
1035 000006be a103
1036 000006c0 2302
1037 000006c2 a004
1038 000006c4 ab30
1039 000006c6 >703
1040 000006c8 a363
1041 000006ca 2714
1042 000006cc a62a
1043 000006ce a301
1044 000006d0 2706
1045 000006d2 4302
1046 000006d4 2704
1047 000006d6 >b704
1048 000006d8 >b705
1049 000006da >b706
1050 000006dc >3c00
1051 000006de 2002
1052 000006e0 >f000
1053 000006e2 >b600
1054 000006e4 >b700
1055 000006e6 4f
1056 000006e7 >b700
1057 000006e9 >cd0000
1058 000006ec a602
1059 000006ee >b700
1060 000006f0 >b603
1061 000006f2 >b700
1062 000006f4 >b604
1063 000006f6 >b700
1064 000006f8 >b605
1065 000006fa >b700
1066 000006fc >b606
1067 000006fe >b700
1068 00000700 >cd0000
1069 00000703 >cd0000
1070 00000706 >b600
1071 00000708 2661
1072 0000070a a606
1073 0000070c >cd0000
1074 0000070e >b603
1075 00000711 >b700
1076 00000713 >b604
1077 00000715 >b700
1078
1079
1080
1081
1082
1083
1084
1085 00000717 >b601
1086 00000719 >b602
1087 0000071b >b602
1088 0000071d >b700
1089 0000071f >b603
1090 00000721 >b700
1091 00000723 >b604
1092 00000725 >b700
1093 00000727 >b605
1094 00000729 >b700
1095 0000072b >b606
1096 0000072d >b700
1097
1098 0000072f >b600
1099 00000731 a018
1100 00000733 >b700
1101 00000735 >b600
1102 00000737 >cd0000
1103 00000739 >cd0000
1104 0000073d >1500
1105 0000073f >cc0000
1106
1107 00000742 >b600
1108 00000744 a130
1109 00000746 2604
1110 00000748 a638
1111 0000074a >b700
1112
1113 0000074c a608
1114 0000074e >b700
1115 00000750 a608
1116 00000752 >b700
1117 00000754 >a600
1118
1119 00000756 a622
1120 00000758 >b700
1121 0000075a >cc0000
1122
1123 0000075d >b600
1124 0000075f >b700
1125 00000761 >b600
1126 00000763 >cd0000
1127 00000766 a604
1128 00000768 >cc0000
1129 0000076b 81

```

```

*****
*
* Sub-page number entry routine.
*
*****

```

```

DIGITS JSR TPSTP
        LDA W2
        SUB #16
SD0     LDX POP
        BEQ THOU
        CFX #2
        BNE SORTD
THOU    CMP #7 THOUSANDS OR TENS
        BLS SOCH NO, SO DON'T CHANGE
        SUB #8 YES, 8->0 & 9->1
        TSTX WAS CFX #0
        BNE SORTD MORE THAN 3 ?
        CMP #3 NO
        BLS SORTD YES, 4->0 THRU 7->3
        SUB #4 CONVERT TO ASCII
        ADD #830 UNITS ?
        STA PAGE+3,X YES, SO CLEAR PDP
        CFX #3 ASTERISK
        BEQ SLRPD HUNDREDS ?
        LDA #S2A YES, SO LEAVE HUNDREDS
        CFX #1 HUNDREDS ?
        BEQ HUMM YES, SO LEAVE TENS & HUNDREDS ?
        STA PAGE+4 CLEAR HUNDREDS
        BEQ HUMM CLEAR TENS
        CFX #2 TENS ?
        BEQ SEN YES, SO LEAVE TENS & HUNDREDS ?
        STA PAGE+4 CLEAR HUNDREDS
        INC PDP CLEAR UNITS
        BRA SPGN
SLRPD   CLR PDP
SPGN    LDA ACC
        STA R8
        CLRA
        STA R9
        JSR BOXOON COLUMN 0
        LDA #2 COLUMN 2
        STA R10
        LDA PAGE+3
        STA R11
        LDA PAGE+4
        STA PH
        LDA PAGE+5
        STA FT
        LDA PAGE+6
        STA PU
        JSR TXT3
        JSR TRAN1
        LDA PDP
        BNE SBO
        LDA #6
        JSR MOBK
        LDA PAGE+3
        STA R11
        LDA PAGE+4
        STA PH

```

```

*****
*
* Get requested sub-page.
*
*****

```

```

SETIT  LDA PAGE+1
        STA C1
        LDA PAGE+2
        STA C2
        LDA PAGE+3
        STA C3
        LDA PAGE+4
        STA C4
        LDA PAGE+5
        STA C5
        LDA PAGE+6
        STA C6
        LDA PAGE
        SUB #S18 PAGE HUNDREDS
        STA R3 PAGE REQUEST HUNDREDS
        LDA ACC
        JSR UP
        JSR TX11 REQUEST IT
        BCLR 2,STAT NOHOLD
        JMP SFND WRITE ONE TO FOUND
        LDA PH
        CMP #S30
        BNE TXT3
        LDA #S38
        STA PH
        LDA #8
        STA W1
        LDA #8 WRITE CCT RAM VIA IIC
        STA SUB3
        LDX #SUB3
        LDA #S22
        STA ADDR
        JMP SEND
        LDA PAGE HOLD DURING
        STA R3 SUB-PAGE NUMBER
        LDA ACC ENTRY
        JSR UP
        LDA #4
        JMP SPM
SBO     RTS

```

```

1131
1132
1133
1134
1135
1136
1137 0000076c >0c002e
1138 0000076f >1c00
1139 00000771 a6ea
1140 00000773 >3f00
1141 00000775 >cd0000
1142 00000778 >e600
1143 0000077a >b700
1144 0000077c >e601
1145 0000077e >b701
1146 00000780 >e602
1147 00000782 >b702
1148 00000784 a62a
1149 00000786 >b700
1150 00000788 >b700
1151 0000078a >b700
1152 0000078c >b700
1153 0000078e >b600
1154 00000790 >b700
1155 00000792 >3f00
1156 00000794 a602
1157 00000796 >b700
1158 00000798 a6b2
1159 0000079a >cc0000
1160
1161 0000079d a60d
1162 0000079f >b600
1163 000007a1 >cd0000
1164 000007a4 >1500
1165 000007a6 >cd0000
1166 000007a9 >cc0000
1167
1168 000007ac >1d00
1169 000007ae >3f00
1170 000007b0 a650
1171 000007b2 >b700
1172 000007b4 >cd0000
1173 000007b7 >e600
1174 000007b9 >b700
1175 000007bb a618
1176 000007bd >b700
1177 000007bf >e601
1178 000007c1 >b700
1179 000007c3 >b700
1180 000007c5 >e602
1181 000007c7 >b700
1182 000007c9 >b700
1183 000007cb >3f00
1184 000007cd a602
1185 000007cf >b700
1186 000007d1 >b600
1187 000007d3 >b700
1188 000007d5 >cc0000
1189
1190
1191
1192
1193
1194
1195
1196 000007d8 >b600
1197 000007da >b700
1198 000007dc >1b00
1199 000007de a602
1200 000007e0 >b700
1201 000007e2 a619
1202 000007e4 >cd0000
1203 000007e7 >b601
1204 000007e9 >b100
1205 000007eb 2704
1206 000007ed >1a00
1207 000007ef >b700
1208 000007f1 >b600
1209 000007f3 >b700
1210
1211 000007f5 a604
1212 000007f7 >b700
1213 000007f9 a619
1214 000007fb >cd0000
1215 000007fe >b601
1216 00000800 >b100
1217 00000802 2704
1218 00000804 >1a00
1219 00000806 >b700
1220 00000808 >b600
1221 0000080a >b100
1222 0000080c 2704
1223 0000080e >1a00
1224 00000810 >b700
1225
1226 00000812 a60c
1227 00000814 >1500
1228 00000816 >1700
1229 00000818 >b600
1230 0000081a >1700
1231 0000081c a606
1232 0000081e >b700
1233 00000820 a619
1234 00000822 >cd0000
1235 00000825 >1700
1236 00000827 >030102
1237 0000082a >1600
1238 0000082c >b600
1239 0000082e >b100
1240 00000830 2704
1241 00000832 >1a00
1242 00000834 >b700
1243
1244 00000836 0f0101
1245 00000839 81

```

```

.....
*
* Sub (timed) pages.
*
.....
SUBPG BSET 6,STAT,OUTSP
      BSET 6,STAT
      BSR TPSTP
      CLR PDP
      JSR INDX
      LDA PAG0,X
      STA PAGE
      LDA PAGE+1,X
      STA PAGE+1
      LDA PAGE+2,X
      STA PAGE+2
      LDA #52A
      STA R11
      STA PH
      STA PT
      STA PU
      LDA ACC
      STA R8
      CLR R9
      LDA #2
      STA R10
      BSR TXT3
      JMP TRAN1
OUTSP BSR RSTR
      LDA ACC
      JSR UP
      BCLR 2,STAT RESET HOLD FLAG
      JSR TXT1
      JMP TRAN1
RSTR BCLR 6,STAT
      CLR PDP
      LDA #550 P
      STA R11
      JSR INDX
      LDA PAG0,X
      STA PH
      SUB #518
      STA R3
      LDA PAGE+1,X
      STA PT
      STA C1
      LDA PAGE+2,X
      STA PU
      STA C2
      CLR R9
      LDA #2
      STA R10
      LDA ACC
      STA R8
      JMP TXT3
.....
*
* Read in Row 25 information.
*
.....
GET25 LDA ACC
      STA R8
      BCLR 5,STAT2 CLEAR DIFFERENCE FLAG
      LDA #2 COLUMN 2 (MINUTES)
      STA R10
      LDA #25 ROW
      JSR R2B
      LDA IOBUF+1
      CMP C6
      BEQ SM5
      BSET 5,STAT2
      STA C6 MINUTES UNITS
SM6 LDA IOBUF
      STA SUB2 MINUTES TENS & CBIT 4
      LDA #4
      STA R10 COLUMN 4 (HOURS)
      LDA #25 ROW
      JSR R2B
      LDA IOBUF+1
      CMP C4
      BEQ SM4
      BSET 5,STAT2
      STA C4 HOURS UNITS
SM4 LDA IOBUF
      CMP C3
      BEQ SM3
      BSET 5,STAT2
      STA C3 HOURS TENS & CBITS 5 & 6
SM3 AND #50C SAVE CBITS 5 & 6 IN STAT7
      BCLR 2,STAT7 CLEAR NEWSFLASH BIT
      BCLR 3,STAT7 CLEAR SUBTITLE BIT
      ORA STAT7
      STA STAT7
      LDA #6 COLUMN 6 (CONTROL BITS)
      STA R10
      LDA #25 ROW
      JSR R2B
      BCLR 3,SUB2 XFER CBITS (UPDATE)
      BRCLR 1,IOBUF+1,TR5 TO BIT 3 OF MINUTES TENS
      BSET 3,SUB2 (REPLACING CBIT4 (ERASE))
      LDA SUB2
      CMP C5
      BEQ CGET26
      BSET 5,STAT2
      STA C5
CGET26 BRCLR 7,PORTB,GET26 PACKET 26 ENABLED ?
      RTS

```



```

1247
1248
1249
1250
1251
1252
1253 0000083a a6ff
1254 0000083c >b700
1255
1256 0000083e >3f01
1257 00000840 >b600
1258 00000842 ab04
1259 00000844 >b700
1260 00000846 >b601
1261 00000848 >b700
1262 0000084a >3c00
1263 0000084c >b600
1264 0000084e a10e
1265 00000850 2303
1266 00000852 >cd0000
1267 00000855 >b600
1268 00000857 >cd0000
1269 0000085a >b601
1270 0000085c >b100
1271 0000085e 26de
1272 00000860 >3a01
1273
1274 00000862 >b600
1275 00000864 ab04
1276 00000866 >b700
1277 00000868 >3c01
1278 0000086a >3c01
1279 0000086c >b601
1280 0000086e >b700
1281 00000870 a126
1282
1283
1284 00000872 230d
1285 00000874 >3f00
1286 00000876 a6ff
1287 00000878 >b700
1288 0000087a >b600
1289 0000087c >cd0000
1290 0000087f 20bd
1291
1292 00000881 >b600
1293 00000883 >cd0000
1294 00000886 >b601
1295 00000888 >b708
1296 0000088a >b600
1297 0000088c >b707
1298 0000088e >3c01
1299 00000890 >b601
1300 00000892 >b700
1301 00000894 >cd0000
1302
1303 00000897 >b600
1304 00000899 >b706
1305 0000089b >b601
1306 0000089d a47c
1307 0000089f 44
1308 000008a0 44
1309 000008a1 >b702
1310 000008a3 >cd0000
1311 000008a6 >b605
1312 000008a8 a128
1313 000008aa 2706
1314 000008ac 250a
1315
1316 000008ae a028
1317 000008b0 2002
1318 000008b2 a618
1319
1320 000008b4 >b704
1321 000008b6 20aa
1322
1323 000008b8 >b604
1324 000008ba >b700
1325 000008bc >b600
1326 000008be >b700
1327 000008c0 >b605
1328 000008c2 >b700
1329
1330 000008c4 >090241
1331 000008c7 >b602
1332 000008c9 a110
1333 000008cb 27f5
1334
1335 000008cd 5f
1336 000008ce >1f06
1337 000008d0 >d60000
1338 000008d3 >b106
1339 000008d5 270a
1340 000008d7 9f
1341 000008d8 ab07
1342 000008da 97
1343 000008db a15b
1344 000008dd 23f1
1345 000008df 2063
1346
1347 000008e1 >b602
1348 000008e3 a40f
1349 000008e5 >b703
1350 000008e7 275b
1351 000008e9 a104
1352 000008eb 2312
1353 000008ed a108
1354 000008ef 2604
1355 000008f1 a003
1356 000008f3 2008

```

```

*****
*
*      Process packet 26 info.
*
*****
GET26  LDA  #8FF
      STA  LIFO
LOOP26 CLR  LIFO+1      START NEW ROW
      LDA  ACC
      ADD  #4          GHOST CHAPTER
      STA  R8
      LDA  LIFO+1
      STA  R10
      LDC  LIFO
      LDA  LIFO
      CMP  #14        STILL PACKET 26 ?
      BLS  OKROW
      JMP  END26
OKROW  LDA  LIFO
      JSR  R2B
      LDA  IOBUF+1
      CMP  LIFO      IS BYTE ZERO OK ?
      BNE  LOOP26    NO, TRY NEXT ROW
      DEC  LIFO+1
LOOP62 LDA  ACC
      ADD  #4
      STA  R8
      INC  LIFO+1
      INC  LIFO+1
      LDA  LIFO+1
      STA  R10
      CMP  #38        PAST END OF ROW 2
      BNE  #
1284   BLS  NXTCH
1285   CLR  R10        YES, BLOW AWAY ROW
1286   LDA  #8FF
1287   STA  R11
1288   LDA  LIFO
1289   JSR  W2B
1290   BRA  LOOP26    NEXT ROW
NXTCH  LDA  LIFO
      JSR  R2B        GET 2 BYTES
      LDA  IOBUF+1
      LDA  IOBUF
      STA  LIFO+7
      INC  LIFO+1
      LDA  LIFO+1
      STA  R10
      JSR  R2B+9     GET THIRD BYTE
1303   LDA  IOBUF
1304   STA  LIFO+6
1305   LDA  LIFO+7
1306   AND  #57C
1307   LSR#A
1308   STA  LIFO+2    SAVE MODE
1309   JSR  EXAD
1311   LDA  LIFO+5
1312   CMP  #40
1313   BEQ  RW24      ROW 24 ?
1314   BLO  NOTROW
1316   SUB  #40      SUBTRACT 40 FOR ROW
1317   BRA  SKIP
RW24  LDA  #24
SKIP  STA  LIFO+4
      BRA  LOOP62
NOTROW LDA  LIFO+4
      STA  R9
      LDA  ACC
      STA  R8
      LDA  LIFO+5
      STA  R10
1330   BRCLR 4, LIFO+2, NOTD  DIACRITICAL ?
1331   LDA  LIFO+2
1332   CMP  #810      NULL ?
1333   BEQ  NULLD     YES, JUST SEND IT (BIT7-1)
1335   CLR# 7, LIFO+6
1337   LDA  CTAB, X
1338   CPE  LIFO+6
1339   BEQ  CHFND
1340   TXA  #7
1341   ADD  #7
1342   TAX  #
1343   CPE  #91
1344   BLS  TRNCH
1345   BRA  CHNF
CHFND  LDA  LIFO+2
      AND  #50F
      STA  LIFO+3
      BEQ  CHNF      NULL DIA.
      CPE  #4
      BLS  CTT
      CPE  #8
      BNE  NOTCF
      SUB  #3
      BRA  UOC

```

1358	000008f5	a10b	NOTCF	CMP	#11	
1359	000008f7	2702		BEQ	CEDI	
1360	000008f9	2049		BRA	CHNF	ILLEGAL MODE
1361	000008fb	4005	CEDI	SUB	#5	
1362	000008fd	>b703	UOC	STA	LIFO+3	
1363	000008ff	9f	GTT	TXA		
1364	00000900	>bb03		ADD	LIFO+3	
1365	00000902	97		TXA		
1366	00000903	>d60000	LDA	CTAB,X		
1367	00000906	203e	BRA	GOTCH		
1368						
1369	00000908	>b602	NOTD	LDA	LIFO+2	
1370	0000090a	a10f		CMP	#50F	
1371	0000090c	271c		BEQ	G2BIT	
1372	0000090e	a102		CMP	#602	
1373	00000910	263e		BNE	END26	
1374						
1375	00000912	>1f06	G3BIT	BCLR	7,LIFO+6	
1376	00000914	5f		CLRX		
1377	00000915	>d60000	TN32	LDA	G3TAB,X	
1378	00000918	2603		BNE	STRM	
1379	0000091a	>cc0000		JMP	LOOP62	
1380	0000091d	>b106	STRM	CMP	LIFO+6	
1381	0000091f	2704		BEQ	G32F	
1382	00000921	5c		INCX		
1383	00000922	5c		INCX		
1384	00000923	20f0		BRA	TN32	
1385	00000925	>d60001	G32F	LDA	G3TAB+1,X	
1386	00000928	201c		BRA	GOTCH	
1387						
1388	0000092a	>1f06	G2BIT	BCLR	7,LIFO+6	
1389	0000092c	5f		CLRX		
1390	0000092d	>d60000	TN23	LDA	G2TAB,X	
1391	00000930	2603		BNE	STMR	
1392	00000932	>cc0000		JMP	LOOP62	
1393	00000935	>b106	STMR	CMP	LIFO+6	
1394	00000937	2704		BEQ	G23F	
1395	00000939	5c		INCX		
1396	0000093a	5c		INCX		
1397	0000093b	20f0		BRA	TN23	
1398	0000093d	>d60001	G23F	LDA	G2TAB+1,X	
1399	00000940	2004		BRA	GOTCH	
1400						
1401	00000942	>1e06	NULD	BSET	7,LIFO+6	
1402	00000944	>b606	CHNF	LDA	LIFO+6	
1403	00000946	>b700	GOTCH	STA	R11	
1404						
1405	00000948	a605		LDA	#5	
1406	0000094a	>cd0000		JSR	TXT32	
1407	0000094d	>cc0000		JMP	LOOP62	
1408						
1409	00000950	81	END26	RTS		
1410						
1411						
1412						
1413						
1414						
1415						
1416						
1417	00000951	202021e02303	G2TAB	FCB	\$20,\$20,\$21,\$E0,\$23,\$83	
1418	00000957	248426932740		FCB	\$24,\$84,\$26,\$93,\$27,\$40	
1419	0000095d	289429a72aa2		FCB	\$28,\$94,\$29,\$A7,\$2A,\$A2	
1420	00000963	2cbcc2d5e2eb8e		FCB	\$2C,\$BC,\$2D,\$5E,\$2E,\$BE	
1421	00000969	2f7630cb37c7		FCB	\$2F,\$76,\$30,\$CB,\$37,\$C7	
1422	0000096f	388a39a73aa2		FCB	\$38,\$8A,\$39,\$A7,\$3A,\$A2	
1423	00000975	3c823d8c3e89		FCB	\$3C,\$82,\$3D,\$8C,\$3E,\$89	
1424	0000097b	3fa1c1f963e5		FCB	\$3F,\$81,\$61,\$F9,\$63,\$E5	
1425	00000981	69fd6be66cfe		FCB	\$69,\$FD,\$6E,\$6E,\$6C,\$FE	
1426	00000987	71f879fc7cff		FCB	\$71,\$F8,\$79,\$FC,\$7C,\$FF	
1427	0000098d	7f7f00		FCB	\$7F,\$7F,\$00	
1428						
1429	00000990	51815b8d5c8b	G3TAB	FCB	\$51,\$81,\$5B,\$8D,\$5C,\$8B	
1430	00000996	5d8e5f2000		FCB	\$5D,\$8E,\$5F,\$20,\$00	
1431						
1432	0000099b	61eae8d2c59261	CTAB	FCB	\$61,\$EA,\$EB,\$D2,\$C5,\$92,\$61	a
1433	000009a2	411f1041d59b41		FCB	\$41,\$F1,\$F0,\$41,\$D5,\$9B,\$41	A
1434	000009a9	65e9ecd665db65		FCB	\$65,\$E9,\$EC,\$DC,\$65,\$DB,\$65	e
1435	000009b0	45290a45454545		FCB	\$45,\$F2,\$90,\$45,\$45,\$45	E
1436	000009b7	6969dde69d469		FCB	\$69,\$69,\$ED,\$DE,\$69,\$D4,\$69	I
1437	000009be	4949f34949f449		FCB	\$49,\$49,\$F3,\$49,\$49,\$F4,\$49	I
1438	000009c5	6fc8eed8c6986f		FCB	\$6F,\$C8,\$EE,\$D8,\$C6,\$98,\$6F	O
1439	000009cc	4ff6f5d8d69c4f		FCB	\$4F,\$F6,\$F5,\$D8,\$D6,\$9C,\$4F	O
1440	000009d3	75c1efd975e275		FCB	\$75,\$C1,\$EF,\$D9,\$75,\$E2,\$75	u
1441	000009da	5555f755559e55		FCB	\$55,\$55,\$F7,\$55,\$55,\$9E,\$55	U
1442	000009e1	6ee6e6ee86e6e		FCB	\$6E,\$6E,\$6E,\$6E,\$6E,\$6E	n
1443	000009e8	4e4e4e4e74e4e		FCB	\$4E,\$4E,\$4E,\$4E,\$E7,\$4E,\$4E	N
1444	000009ef	636363636363e3		FCB	\$63,\$63,\$63,\$63,\$63,\$63,\$E3	C
1445	000009f6	434343434343d7		FCB	\$43,\$43,\$43,\$43,\$43,\$43,\$D7	C
1446						
1447	000009fd	>3f05	EXAD	CLR	LIFO+5	
1448	000009ff	>030702		BRCLR	1,LIFO+7,NO32	
1449	0000a02	>1a05		BSET	5,LIFO+5	
1450	0000a04	>010702	NO32	BRCLR	0,LIFO+7,NO16	
1451	0000a07	>1805		BSET	4,LIFO+5	
1452	0000a09	>0d0802	NO16	BRCLR	6,LIFO+8,NO8	
1453	0000a0c	>1605		BSET	3,LIFO+5	
1454	0000a0e	>0b0802	NO8	BRCLR	5,LIFO+8,NO4	
1455	0000a11	>1405		BSET	2,LIFO+5	
1456	0000a13	>090802	NO4	BRCLR	4,LIFO+8,NO2	
1457	0000a16	>1205		BSET	1,LIFO+5	
1458	0000a18	>050802	NO2	BRCLR	2,LIFO+8,NO1	
1459	0000a1b	>1005		BSET	0,LIFO+5	
1460	0000a1d	81	NO1	RTS		

.....

* Packet 26 character look-up table. *

.....

```

1462
1463
1464
1465
1466
1467
1468 00000a1e >b100
1469 00000a20 >1400
1470 00000a22 >1400
1471 00000a24 >3f00
1472 00000a26 a604
1473 00000a28 <c<d0000
1474 00000a2b a604
1475 00000a2d >b700
1476 00000a2f a660
1477 00000a31 >b700
1478 00000a33 >3a00
1479 00000a35 2735
1480 00000a37 a601
1481 00000a39 <c<d0000
1482 00000a3c >3f00
1483 00000a3e a617
1484 00000a40 <c<d0000
1485 00000a43 >b601
1486 00000a45 26cc
1487 00000a47 a601
1488 00000a49 >b700
1489 00000a4b a630
1490 00000a4d >b700
1491 00000a4f a617
1492 00000a51 <c<d0000
1493 00000a54 >b600
1494 00000a56 >b700
1495 00000a58 >b700
1496 00000a5a >b600
1497 00000a5c >b701
1498 00000a5e >b600
1499 00000a60 >3f00
1500 00000a62 >b600
1501 00000a64 >b700
1502 00000a66 <c<d0000
1503 00000a69 >b7000
1504 00000a6c >b601
1505 00000a6e a110
1506 00000a70 2502
1507 00000a72 <1c00
1508 00000a74 a631
1509 00000a76 >b700
1510 00000a78 4a
1511 00000a79 >b700
1512 00000a7b >b700
1513 00000a7d 20d5
1514
1515
1516
1517
1518
1519
1520
1521 00000a7f a604
1522 00000a81 >b700
1523 00000a83 48
1524 00000a84 >b700
1525 00000a86 a60b
1526 00000a88 >b700
1527 00000a8a >b700
1528 00000a8c a618
1529 00000a8e >b700
1530 00000a90 a606
1531 00000a92 <c<d0000
1532 00000a95 a61f
1533 00000a97 >b700
1534 00000a99 a60a
1535 00000a9b >b700
1536 00000a9d >b700
1537 00000a9f a606
1538 00000aa1 <c<d0000
1539 00000aa4 a614
1540 00000aa6 >b700
1541 00000aa8 >b700
1542 00000aaa a617
1543 00000aac <c<d0000
1544 00000aae >b601
1545 00000ab1 a47f
1546 00000ab3 >b700
1547 00000ab5 >b600
1548 00000ab7 a47f
1549 00000ab9 >b700
1550 00000ab5 <c<d0000
1551 00000abd a00a
1552 00000abf >b700
1553 00000ac1 a618
1554 00000ac3 <c<d0000
1555 00000ac6 >3c00
1556 00000ac8 >3c00
1557 00000aca >b600
1558 00000acc a127
1559 00000ace 23d8
1560 00000ad0 #1
1561
1562 00000ad1 <c<d0000
1563 00000ad4 a614
1564 00000ad6 <c<d0000
1565 00000ad9 a60c
1566 00000adb >b700
1567 00000ade <c<d0000
1568 00000ae0 a606
1569 00000ae2 <c<d0000
1570 00000ae5 <c<d0000

*****
*
* Fetch initial page from 8/30 format 1.
*
*****
GIP BCLR 5,STAT CLEAR TIME HOLD
      BCLR 6,STAT CLEAR SUB-PAGE MODE
      BCLR 6,STAT2 CLEAR NO TXT FLAG
      CLR PDP
      LDA #2
      JSR SPM TXT1 1 BYTE ONLY
      LDA #4
      STA R8 CHAPTER 4 (GHOST)
      LDA #96 96 TRYS
      STA W3
      TRYAG DEC W3
      BEQ IPNF AGAIN ?
      LDA #1
      JSR TPAU2
      CLR R10
      LDA #23
      JSR R2B
      LDA IOBUF+1 8/30 FORMAT 1 FOR INITIAL PAGE
      BNE TRYAG
      LDA #1 COLUMN 1
      STA R10
      LDA #530 RESET PAGE HUNDREDS
      STA PH
      LDA #23 LINE 23 (PACKET 8/30)
      JSR R2BJ2
      LDA PH INITIALISE INDEX (BLACK)
      STA PAGI
      STA PAGE
      LDA PT
      STA PAGI+1
      LDA PU
      STA PAGI+2
      LDA ACC
      STA WACC
      JSR UP
      JMP CLP2
      LDA IOBUF+1
      CMP #510
      BLO P830OK
      LDA #531 REQUEST
      STA PH PAGE 100
      DECA IN CASE
      STA PT INITIAL PAGE
      STA PU NOT FOUND
      BRA GETIND

*****
*
* Row 24 transient.
*
*****
R24T LDA #4 CHAPTER 4
      STA R8
      LSLA
      STA R10 BOX ON AT 8 & 9
      LDA #50B
      STA R11
      STA PH
      LDA #24 ROW 24
      STA R9
      LDA #6
      JSR TXT32 WRITE BOX ON
      LDA #31 BOX OFF AT 31 & 32
      STA R10
      LDA #50A
      STA R11
      STA PH
      LDA #6
      JSR TXT32 WRITE BOX OFF
      LDA #20
      STA W3 START READING @ COLUMN 20
      LDA #23 ROW 23 - PACKET 8/30
      JSR R2B
      LDA IOBUF+1
      LDA #57F
      AND R11
      STA R11
      LDA IOBUF
      AND #57F
      STA PH
      LDA W3
      SUB #10 START WRITING AT COLUMN 10
      STA R10
      LDA #24 WRITE TO ROW 24
      JSR W2B
      INC W3
      INC W3
      LDA W3
      CMP #39 ALL DONE ?
      BLS EA
      RTS

EA STA W3
      LDA #23
      JSR R2B
      LDA IOBUF+1
      AND #57F
      STA R11
      LDA IOBUF
      AND #57F
      STA PH
      LDA W3
      SUB #10
      STA R10
      LDA #24
      JSR W2B
      INC W3
      INC W3
      LDA W3
      CMP #39
      BLS EA
      RTS

NOTR RTS

START2 JSR TXTOF
        LDA #20
        SDLY JSR TPAU2
        NIICD LDA #50C
        STA R8
        JSR TXT3 CLEAR CHAPTER 4
        LDA #6
        JSR TPAU2
        JSR GIP GET PAGE No. FROM 8/30

```


Symbol cross-reference

CNT1	*27																				
CNT3	*27																				
CNT4	*27																				
CNT5	*27																				
COK	*239																				
COR	367	*369																			
COUNT	*27	312	320	325	332	339	341	342	349	353											
CPBLF	613	*945																			
CTAB	1337	1366	*1432																		
CYAN	17	*261																			
CYOK	*297																				
DDI	*574																				
DECODE	372	375	404	407	572	690	*700														
DIFFH	*27																				
DIFFL	*27																				
DIGIT	103	*105																			
DIGIT0	17	*103																			
DIGITS	104	*1023																			
DISC	*27																				
DISP	*27																				
DISP4	893	*898																			
DISP8	867	*890																			
DPGN	128	*130																			
DPNT	*27																				
DRAM	*27																				
EA	*1541	1559																			
END26	1266	1373	*1409																		
EXAD	1310	*1447																			
EXP	967	*973																			
EXPTB	18	*967																			
FILE	758	760	*761																		
FIMI	536	539	*543																		
FND	721	730	739	754	763	772	781	785	*787												
FND2	542	*548																			
FNDJ	704	712	*730																		
FC1	749	751	*752																		
FRO	828	*842	990																		
FTUNE	*27																				
G23F	1394	*1398																			
GBIT	1371	*1388																			
G2TAB	1390	1398	*1417																		
G32F	1381	*1385																			
G3BIT	*1375																				
G3TAB	1377	1385	*1429																		
GCYI	336	*665																			
GET25	21	*1196																			
GET26	1244	*1253																			
GETIND	*1493	1513																			
GETIT	*158																				
GIP	18	259	*1468	1570																	
GLOK	209	*215																			
GLP1	323	*365	667	672																	
GLP2	352	*471	669	674	1503																
GOTCH	1367	1386	1399	*1403																	
GREEN	17	*208																			
GTT	1352	*1363																			
H1	385	*388																			
HAM	702	714	723	732	747	756	765	774	*1609												
HAM8	*1610																				
HIGH	952	*954	977																		
HOLD	19	*858																			
HUN	1044	*1048																			
HVL	*27																				
HVR	*27																				
IAC	258	266	*269																		
IGO	215	*217																			
IG1	221	*223																			
IG2	262	*264																			
INDEX	19	*256																			
INDX	160	*179	281	379	424	618	931	1141	1172												
INDXP	201	211	319	*618																	
INITXT	19	*1570																			
IOBUF	*27	371	374	384	390	395	403	406	571	573	595	597	683	689	952	1203					
IPNF	1208	1215	1220	1236	1269	1294	1296	1303	1485	1504	1544	1547									
IRCMCT	1479	*1504																			
IRCMCT	*27																				
IRCNT	*27																				
IRCODE	*27																				
IRH	*27																				
IRL	*27																				
IRRA1	*27																				
IRRA2	*27																				
IRRA3	*27																				
IRRA4	*27																				
K1	*27																				
K2	*27																				
KEY	*27																				
KOUNT	*27																				
LBAL	*27																				
LDO	*113																				
LHOLD	899	901	903	905	*1608																
LIFO	*27	518	626	1254	1256	1260	1262	1267	1270	1272	1277	1278	1279	1288	1292						
	1295	1297	1298	1299	1304	1305	1309	1311	1320	1323	1327	1330	1331	1336	1338	1347					
	1349	1362	1364	1369	1375	1380	1388	1393	1401	1402	1447	1448	1449	1450	1451	1452					
	1453	1454	1455	1456	1457	1458	1459														
LIND	*27																				
LINK	21	*309																			
LOOP	*341	355																			
LLOP	*320	334																			
LOK	505	*508																			
LOOP26	*1256	1271	1290																		
LOOP62	*1274	1321	1379	1392	1407																
LOOPS	*530	546																			
LPT	203	206	213	218	*224																
LPT2	159	*206																			
LVL	*27																				
LVR	*27																				

Symbol cross-reference	
MATRIX	*27
MIX	19 *800
MRE	*584 603
NEXTC	327 *329
NIICD	*1565
NO1	1458 *1460
NO16	1450 *1452
NO2	1456 *1458
NO32	1448 *1450
NO4	1454 *1456
NO8	1452 *1454
NOBX	148 805 987 *1001 1073
NOCH	114 116 *118
NOHOLD	240 859 *915
NOMIX	800 *812
NOTCF	1354 *1358
NOTD	1330 *1369
NOTFND	324 *328
NOTHLD	239 *241
NOTOK	368 *424
NOTOK2	212 *446
NOTOK3	202 *423
NOTR	*1560
NOTROW	1314 *1323
NOTT	46 48 *50
NOTTH	22 49 61 *411
NOTXTX	1578 *1595
NOV9	432 439 *442 454 461
NOV9A	*441 463
NPAGE	18 *201
NRK27	684 *692
NTSAC	232 *235
NULD	1333 *1401
NUM	787 *1607
NTTCH	1284 *1292
OK0	384 387 *389
OK1	390 393 *395
OK2	395 398 *400 444
OKROW	1265 *1267
OLDIR	*27
ON1	716 718 *719
OSDL	*27
OSDLE	*24 40 63
OUT	964 966 970 972 *975
OUTSP	236 1137 *1161
P8300K	1506 *1508
PAG0	*27 77 82 87 162 164 167 271 274 277 283 285 287 380 425 447
	481 484 486 534 537 540 619 621 932 937 940 1142 1144 1146 1173 1177
	1180
PAG1	*27
PAG2	*27
PAG3	*27
PAGC	*27
PAGE	*27 92 119 125 126 137 139 141 149 161 163 166 169 427 430 435
	436 437 441 442 449 452 457 458 459 620 622 1039 1047 1048 1049 1060
	1062 1064 1066 1074 1076 1085 1087 1089 1091 1093 1095 1098 1123 1143 1145 1147
	1495
PAGI	*27 1494 1497 1499
PANIC	38 58 79 81 84 86 89 91 *94
PDP	*27 113 127 129 145 198 208 238 270 858 1026 1050 1052 1070 1140 1169
	1471
PH	*27 138 150 278 282 381 385 386 388 391 392 394 396 397 399 426
	448 485 517 535 592 598 627 902 934 1010 1063 1077 1107 1111 1150 1174
	1490 1493 1509 1527 1536 1549
PLIHI	*27
PLLL	*626 630
PLOW	*27
PPAGE	18 *211
PROG	*27
PSHL	*517 520
PT	*27 140 275 284 376 408 443 482 538 904 938 1065 1151 1178 1496 1511
PT1	391 *394
PU	*27 142 272 286 378 405 429 434 451 456 479 541 906 941 1067 1152
	1181 1498 1512
PU1	396 *399
PULL	346 *625
PUSH	328 *516
PWR	*27
R1	*27 42 66 803 814
R10	*27 134 295 322 365 499 568 569 583 609 644 645 646 666 671 680
	686 845 898 920 948 1001 1004 1007 1059 1157 1185 1200 1212 1232 1261 1280
	1285 1300 1328 1482 1488 1524 1533 1541 1552
R11	*27 136 292 493 591 596 605 615 848 900 922 1009 1061 1075 1149 1171
	1287 1403 1526 1535 1546
R2	*27 189 471
R24T	22 *1521
R2B	21 370 402 588 *650 682 688 950 1202 1214 1234 1268 1293 1484 1543
R2BJ1	*583 409
R2BJ2	*402 1492
R2BN9	570 *651 1301
R3	*27 105 171 280 413 488 872 936 1100 1124 1176 1582
R4	*27 55 130 172 244 982 1011 1586
R5	*27 51 70 807 816 995 1588
R6	*27 53 71 809 818 996 1589
R7	*27 72 832 962 963 965 967 968 969 971 973 974 992 1591
R8	*27 131 289 311 496 511 586 600 678 847 862 869 917 946 1012 1054
	1154 1187 1197 1259 1276 1326 1475 1522 1566
R9	*27 132 293 497 607 641 650 843 864 871 891 918 1013 1056 1155 1183
	1324 1529
RAD1	*27
RAD2	*27
RAD3	*27
RAD4	*27
RAD5	*27
RAD6	*27
RAD7	*27
RAD8	*27
RADIO	383 389 *568

Symbol cross-reference	
READ	*24 663
READ22	*661
RED	17 *198
REDZ	199 *205
REL1	411 923 *929
REL2	924 *934
REV	962 *965
REVEAL	18 *962
ROW1	*27
ROW24	21 *583
RST	*69
RSTR	237 1161 *1168
RW24	1313 *1318
SAM	*558 562
SBO	1071 *1129
SDO	*1026
SDLY	*1564
SE1	776 778 *779
SEN	1046 *1049
SEND	*24 1121
SEND22	656 839 888 *1119
SETII	*1085
SFND	176 300 513 *605 926 1105
SFND2	*606 616
SHADMAT	*27
SI1	767 769 *770
SKIP	1317 *1320
SKOSP	235 *237
SLAFD	1041 *1052
SH3	1222 *1226
SM4	1217 *1220
SM6	1205 *1208
SNOMD	*27
SOCH	1031 *1033
SORTD	1029 1034 1036 *1038
SP	*27
SPGN	1051 *1053
SPM	75 109 415 824 874 *884 1128 1473 1584
SPM2	877 *883
SR24T	19 *1578
SRCH	158 400 *529
SSUB	711 720 729 738 753 762 771 780 *790
STACK	*27
STAR22	19 *1562
STAT	*27 37 39 43 46 47 48 60 62 64 67 68 877 915 978 980 984
STAT2	*27 44 45 59 800 801 812 1198 1206 1218 1223 1241 1470 1507 1578
STAT3	*27 199 209 216 220 222 261 263 368 593 679 691
STAT4	*27
STAT5	*27
STAT6	*27
STAT7	*27 38 58 977 1227 1228 1229 1230
STR	1391 *1393
STRM	1378 *1380
SUB1	*27 886 887
SUB2	*27 706 707 708 710 715 716 717 719 724 725 726 728 733 734 735
	737 748 749 750 752 757 758 759 761 766 767 768 770 775 776 777
	779 790 837 838 1209 1235 1237 1238
SUB3	*27 652 655 1116 1117
SUBADR	*27 660
SUBPG	979 *1137
TAC	980 *998
TEN	124 *126
TEST	*77 228 257 265
TH1	734 736 *737
THOU	1027 *1030
TIME	19 *977
TMP1	*27
TMP2	*27
THR	*27 830 999 1594
TN23	*1390 1397
TN32	*1377 1384
TOPE	*27
TPAU2	*24 510 1481 1564 1569
TPSTP	1023 *1123 1135
TR5	1236 *1238
TRA	*702 786
TRAN1	144 299 *821 1069 1159 1166
TRAN2	56 810 819 *823 927
TRAN3	*827 875
TRF1	*756
TRFG	*747
TRNCH	*1337 1344
TRON	*714
TRSE	*774
TRSI	*765
TROH	*752
TRTW	*723
TRYAG	*1478 1486
TRZE	*706
TVYX	18 *37
TW1	725 727 *728
TXT1	174 301 *877 943 1103 1165
TXT1L	514 *878
TXT2	21 73 248 *834 975 993 997 1592
TXT3	508 907 1068 1109 *1113 1158 1567
TXT32	611 643 850 1015 *1114 1406 1531 1538
TXT38	143 298 512 925 *1107 1188
TXTOFF	*64 1562
TXTOFF	37 *62
TXTON	*38
UCHOLD	*868 983
WOC	1356 *1362
UP	107 173
UPDATE	18 *58

	Symbol cross-reference												
V5	*648												
W1	*27	654	700	703	791	835	884	1114					
W2	*27	111	373	377	1024								
W2B	602	*641	1289	1554									
W3	*27	224	226	230	243	314	321	329	331	890	894	1477	1478
	1556	1557										1540	1550
WACC	*27	337	347	494	529	530	532	543	544	548	558	560	1501
WRCW	*27												
YELLOW	17	*220											
YIP	*183												
ZE1	707	709	*710										

Using the MC68HC11K4 Memory Mapping Logic

By Steven McAslan
 MCU Applications Engineering
 Motorola Ltd,
 East Kilbride, Scotland

INTRODUCTION

The MC68HC11K4 is provided with memory expansion logic which allows the 64K Byte addressing range of the 68HC11 CPU to be extended to more than 1Mbyte. This application note discusses the operation of this logic and provides examples of memory maps and possible hardware configurations.

THE MC68HC11K4 MEMORY EXPANSION LOGIC

The memory expansion logic extends the addressing range of the 68HC11 CPU by providing two new on-chip blocks. The first new block implements additional address lines which are only made active when required by the CPU. The second block eases interfacing to external memory chips by providing chip select signals. Both of these blocks are fully user programmable.

ADDITIONAL ADDRESS CAPABILITY

If the addressing capability of the 68HC11 CPU is to be extended then the first step involved is to provide additional address lines. The CPU itself provides 16 lines (A0 to A15) which allow up to 64K Bytes of memory to be accessed. Each new address line provided will double that total.

To maintain compatibility with other members of the 68HC11 family the CPU used in the K4 is not functionally changed. The extra addressing capability is provided in

such a way that the CPU itself is unable to distinguish more than 64K Bytes of memory. The extra capacity is provided by switching banks of the extra memory in and out of the 64K Byte range provided. To maximise the flexibility of this approach the size and number of the switched banks is user programmable.

To use extended memory, the programmer must first allocate a range (called a *window*) within the CPU 64K Byte addressing range which will be used when banks are switched in and out. The memory expansion logic allows windows of 8K, 16K or 32K Bytes to be defined and placed at programmable points in the CPU 64K Byte memory. At any time only one bank may be displayed in the defined window and therefore the CPU may only have access to the memory contents of one bank at a time. The bank which is displayed in the window is selected by the additional address lines provided by the memory expansion logic. The process of replacing the active bank with a new bank is called *bank-switching*.

A useful analogy is that of photographic transparencies and a projector. The viewer may have many transparencies but is only able to view one at a time in the projector. If the photographs are numbered then the viewer is able to select precisely which one to view without having to go through all of them. Here the memory banks are akin to the transparencies – many are available but only one is accessible at any time. The number on the transparency can be thought of here as an address. Any transparency which is not being displayed at any one time is still accessible but only when the current one is removed and it is put in its place.

To extend the addressing capability of the CPU, six address lines were added. These are termed XA13, XA14, XA15, XA16, XA17 and XA18. To allow flexibility in the size of the windows, the lower three address lines are only used when determined by the size of the window and replace the CPU's equivalent addresses.

To understand the need for the XA13 to XA15 addresses consider the case when an 8K Byte window is being used. Address lines XA16 to XA18 are only active when the CPU is accessing memory within the window and they provide an extra $8 (2^3)$ times the memory provided by the CPU within that window. If an 8K Byte window is used then a maximum of $8 \times 8K$ Bytes is available. However, for the CPU to uniquely distinguish each location in the 8K Byte window it only requires to use addresses A0 to A12. Changes in address lines A13 to A15 take it outside of the window and are therefore never valid within the window except to identify the starting address of the window. The three new addresses XA13 to XA15 are provided so that a full 512K Byte ($8 \times 8 \times 8K$ Byte) range is realised. For a 16K Byte window only addresses XA14 and XA15 can be used and for 32K Byte only XA15 is usable. Note that the size of the memory window to be used need not necessarily be defined at the hardware design stage since the additional addresses XA13 to XA15 can be programmed to carry the CPU A13 to A15 signals. The situation may be complicated by the need to use differently sized windows (see example 2).

The memory expansion logic actually allows the user to define two independent windows and so more than 1M Byte of memory is accessible.

The hardware required to implement such a large memory range is greatly simplified by the use of the memory expansion's chip select block.

CHIP SELECTS

The memory expansion chip selects are provided to help the user interface the K4 with external memories. Four are provided but only three are of direct importance to the memory expansion logic. These are the two General Purpose Chip Selects and the Program Chip Select. The fourth, I/O Chip Select, is used to simplify the addition of external peripheral chips.

The basic function of a chip select is to provide a logic signal to indicate that the CPU is accessing a certain area of memory. For example, the Program Chip Select is intended to be active in the range of memory where the main program exists. Other chip selects will be active when their respective memory areas are used.

The General Purpose Chip Selects are the most flexible of those provided and their function is closely linked to the memory expansion logic. They can be programmed to be active on an area either within the CPU 64K Byte memory or within either window's 512K Byte range. In both cases the size of memory selected is fully programmable from 2K Bytes to 512K Bytes.

The above paragraphs outline the method by which the memory expansion logic extends the addressing range of the CPU. A detailed description of the internal registers used to implement the new logic is now required. Finally a series of examples are considered.

MEMORY EXPANSION AND CHIP SELECT REGISTERS

This discussion describes the functionality of the internal registers relating to the memory expansion logic and chip selects. Further details on their addresses and specific bit operations may be found in the Technical Summary [1].

The following registers perform the memory expansion function.

The **MMWBR** register allows the starting address of each of the two windows within the CPU 64K Byte address range to be defined. The windows will normally start on a boundary related to their size, for example an 8K Byte window may start on any 8K Byte boundary starting at \$0000, that is, \$2000 \$4000 ... \$E000. A 16K Byte window can only start on 16K Byte boundaries, \$0000 \$4000 ... \$C000. An exception is made for the 32K Byte window. This would normally start at either \$0000 or \$8000. However, the window is also allowed to start at \$4000.

The **MMSIZ** register sets the size of the windows in use and selects whether the chip selects are active for only CPU addresses or for extended addresses.

Each General Purpose Chip Select has two registers called **GP1CSC**, **GP1CSA** and **GP2CSC** and **GP2CSA**.

The *control* register (**GP7CSC**) determines the logical output required when an area of memory is selected (with possible logic combinations with other chip selects) and the range of memory over which the chip select is to be active. Each chip select can be programmed to become active whenever the CPU address enters an memory expansion window (regardless of the actual bank selected); this is known as *following a window*.

The *address* register (**GP7CSA**) allows the starting address of the chip select to be programmed. The bits in this register which are active are determined by the size of the chip select range selected by the control register.

The program and I/O chip selects are programmable via the **CSCTL** register.

Two window registers, **MM1CR** and **MM2CR**, are used to indicate which bank is active in a window. Each contains the values of XA13 to XA18 to be output when the CPU selects addresses within the extended memory window. To change banks the user writes the address of the new bank into the appropriate window register.

The actual memory expansion address lines are multiplexed with PORT G I/O pins. Selecting an address line on one of these pins means that a PORT G pin is lost. For this reason the user need only select those address lines which are needed by the expansion logic. This allows unused lines to be used as general purpose I/O. The register which defines which extended address lines are used is **PGAR**. If an address line is not required then the appropriate bit in **PGAR** should be cleared to 0. A special case exists for two address lines which overlap the CPU address lines (XA13 and XA14). If XA13 or XA14 are selected as address lines in **PGAR**, but are not used in either window, then the appropriate CPU address line will be output on the port.

EXTENDED MEMORY EXAMPLES

The best way to grasp the implications of the K4 extended memory function is to consider some examples. Each example consists of two figures. Figures a are the logical arrangement of the memory and Figures b are a *possible* hardware configuration.

Example 1 shows one window in use. This is an 8K Byte window scheme and provides 8 banks from a single 64K Byte EPROM chip. Note that the logical address of each bank is derived from address lines A0 to A12 then XA13 to XA15. Chip select 1 is used.

Example 2 shows two windows in use. The first window is of 8K Bytes and is organised as in example 1. The second window is of 16K Bytes and is organised as 16 banks in two 128K Byte RAM chips. The logical addresses of the Window 2 banks are determined by A0 to A13 then XA14 to XA17 (XA18 determines start address). Chip select 2 is used for window 2.

Example 3 shows the same two windows as in example 2 except that the logical addressing of the windows are changed. Now Window 1's logical address is determined by A0 to A12 then XA15 to XA17 (XA13 and XA14 ignored) and Window 2's logical address is determined by A0 to A13 then XA15 to XA18 (XA14 ignored). Note that in both windows every bank will be duplicated due to the lack of decoding on certain address lines. In Window 1 each bank is duplicated four times. In Window 2 each bank is duplicated twice.

Example 4 shows the maximum 1Mbyte extended memory possibility in use. Window 1 is 16K Bytes starting at \$0000 and Window 2 is 32K Bytes starting at \$4000. Note that the internal RAM and registers of the K4 are echoed in every page of window 1, but that the internal EPROM in window 2 only occurs in the first 64K Bytes of extended memory. That is, for addresses below \$10000, the internal EPROM is present in the memory map at the relevant address. This currently applies to all window sizes and configurations, however, the user should avoid referring to EPROM at any address beyond page 0 to ensure compatibility with any future upgrades.

Logical addresses of both windows are given by A0 to A13 then XA14 to XA18. Window 2 uses XA14 because it does not start on a 32K Byte boundary and so requires that the XA14 is inverted. Both chip selects are used and follow a window each.

PGAR \$3F - XA13..XA18 MMSIZ \$42 - Window 1 8K
 MMWBR \$04 - window at \$4000 CSCTL \$00 - no I/O or program chip select
 GPCS1A \$00 - from \$00000 GPCS1C \$06 - 64K range (8 x 8K)
 GPCS2A \$00 - not relevant GPCS2C \$00 - disabled

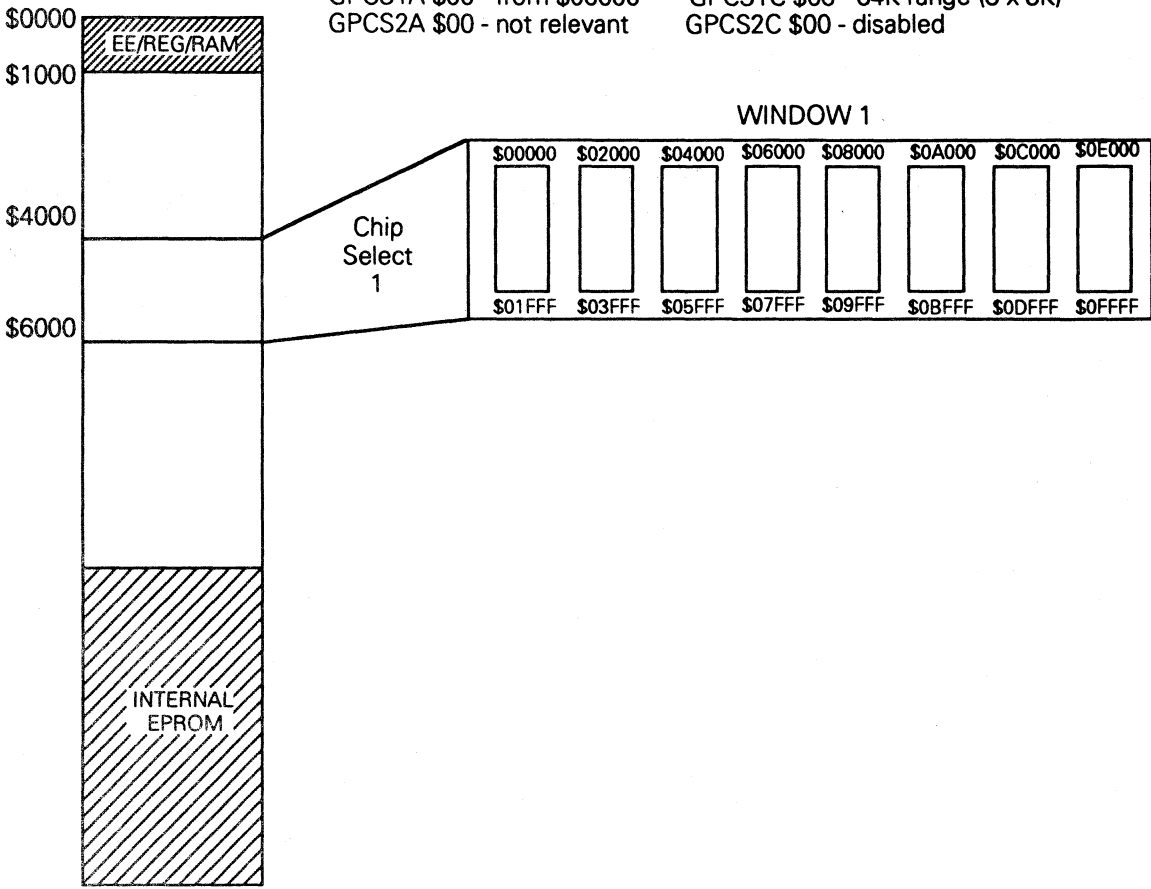


Figure 1a. One 8K Byte window

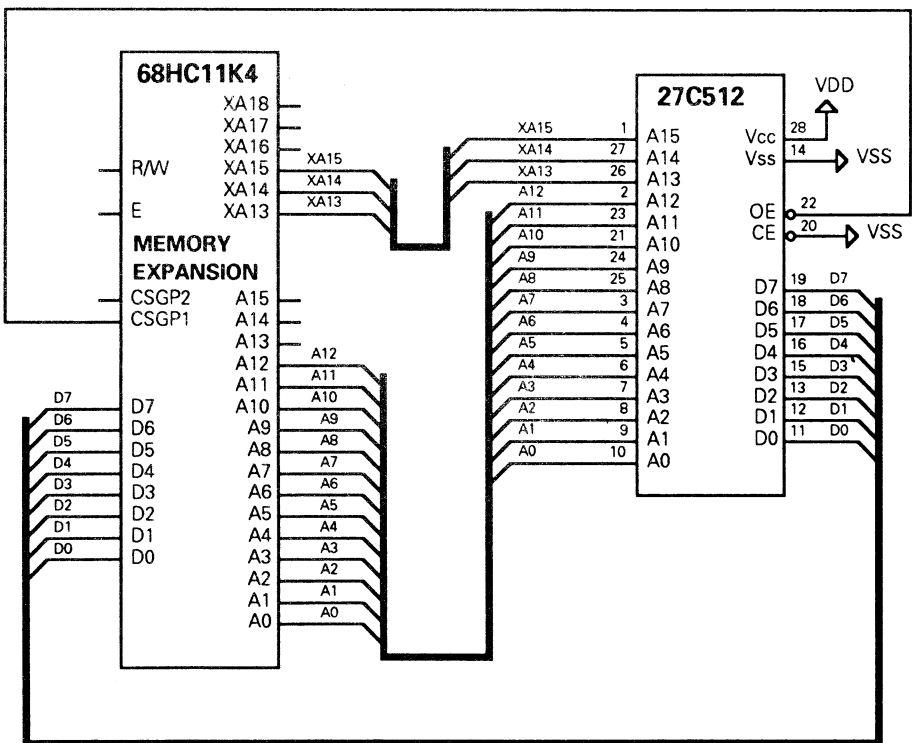


Figure 1b. 64K Bytes in one 8K Byte window

PGAR \$3F - XA13..XA18 MMSIZ \$E2 - Window 1 8K, Window 2 16K
 MMWBR \$84 - Window 1 at \$4000, Window 2 at \$8000 CSCTL \$00 - no I/O or program chip select
 GPCS1A \$00 - from \$00000 GPCS1C \$06 - 64K range (8 x 8K)
 GPCS2A \$80 - from \$40000 GPCS2C \$08 - 256K range (16 x 16K)

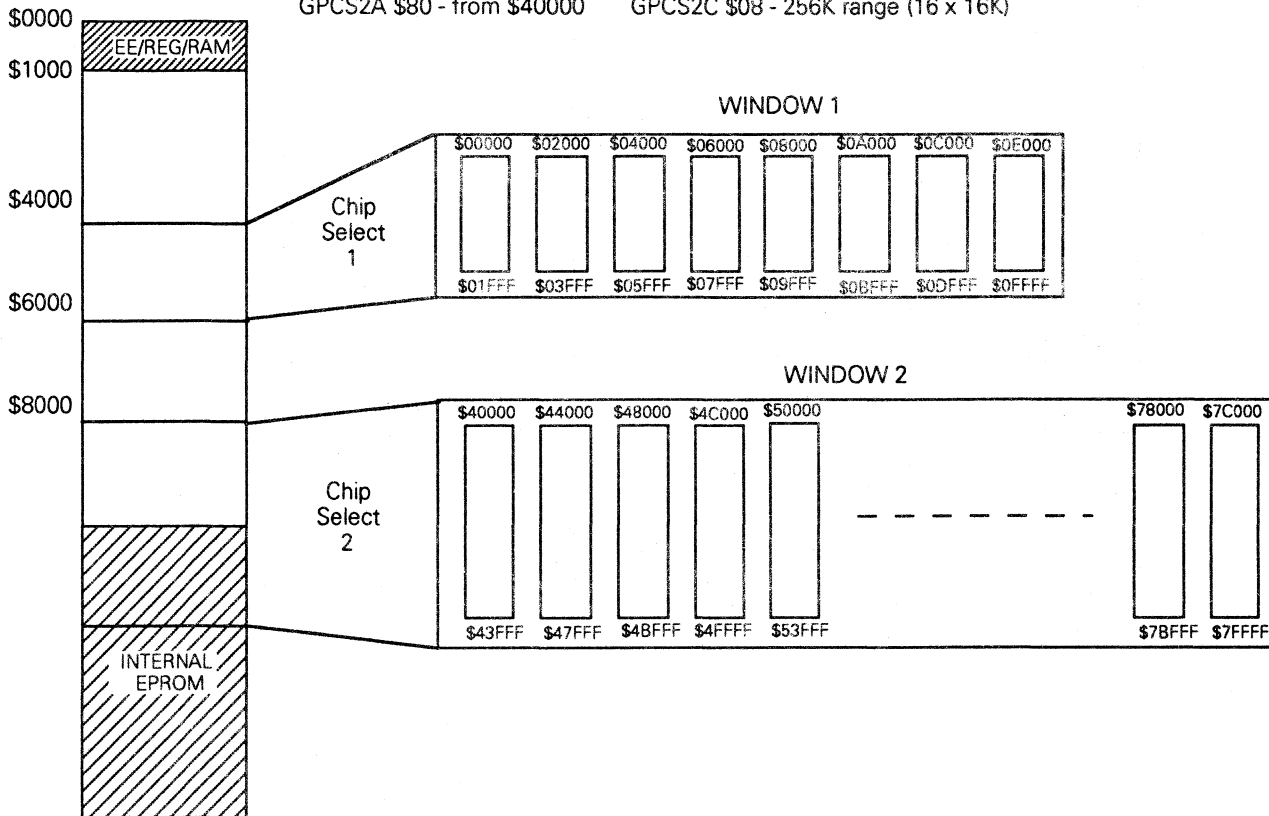


Figure 2a. One 8K Byte window and one 16K Byte window

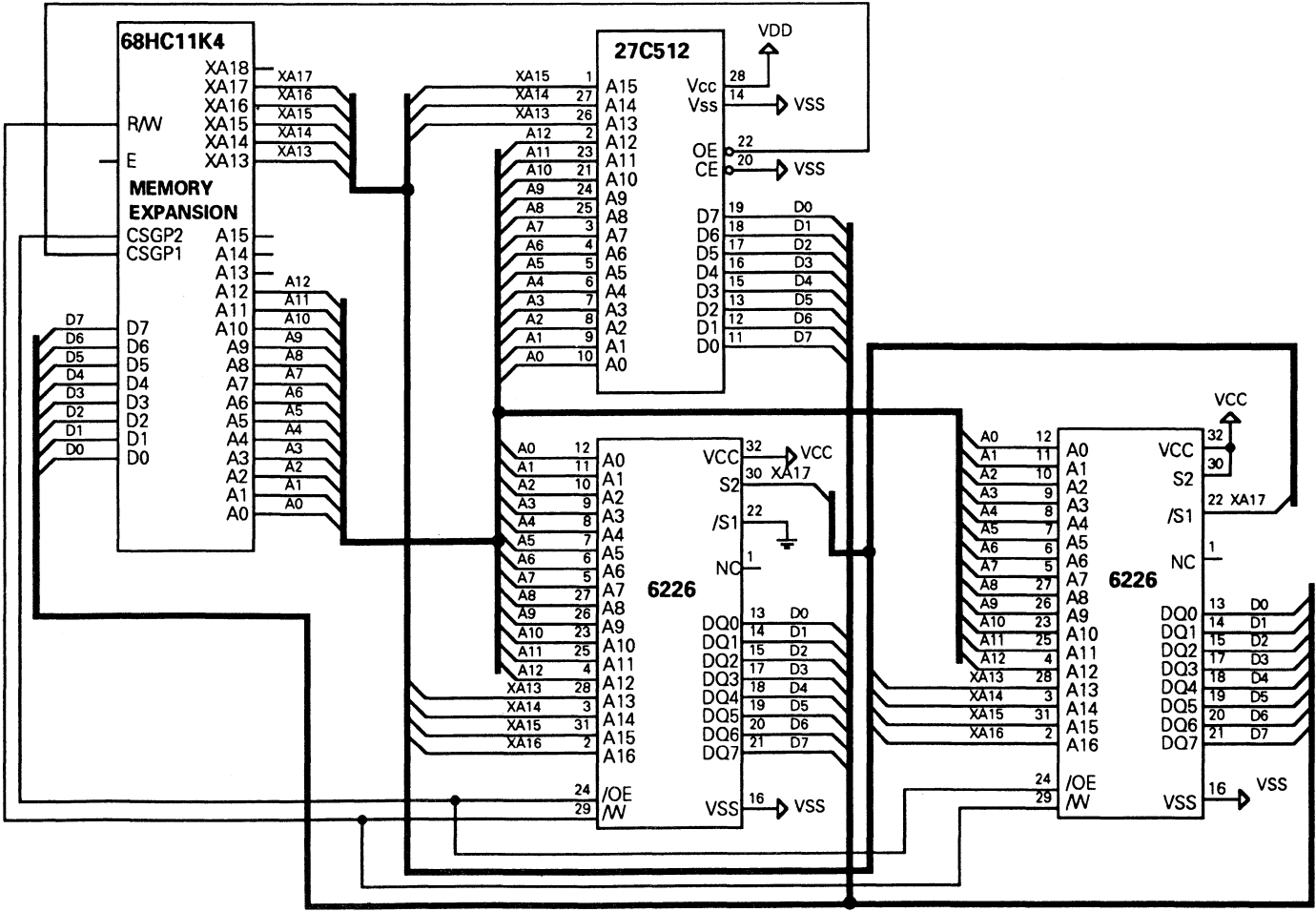
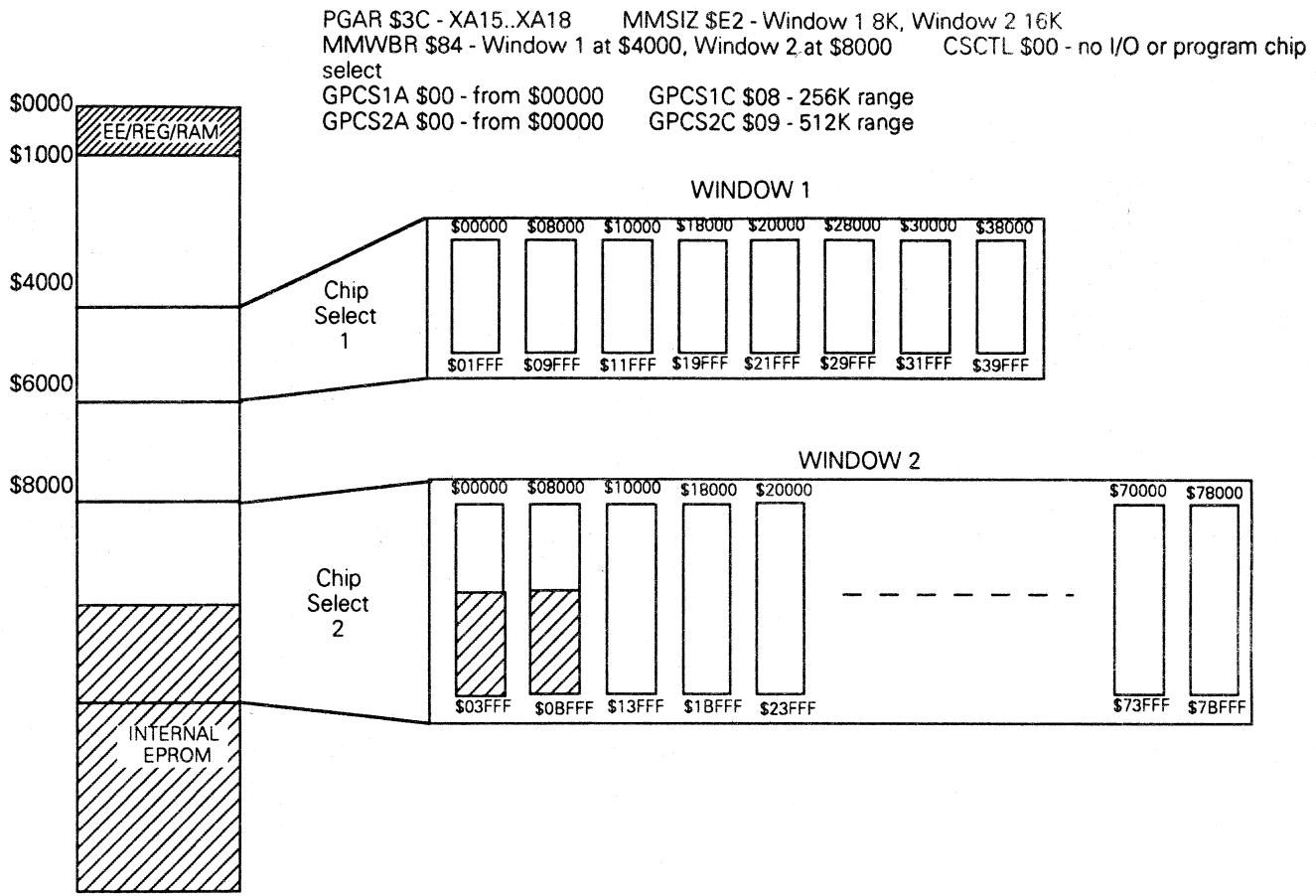


Figure 2b. One 8K Byte window and one 16K Byte window

Figure 3a. 8K Byte window and 16K Byte window with new addressing



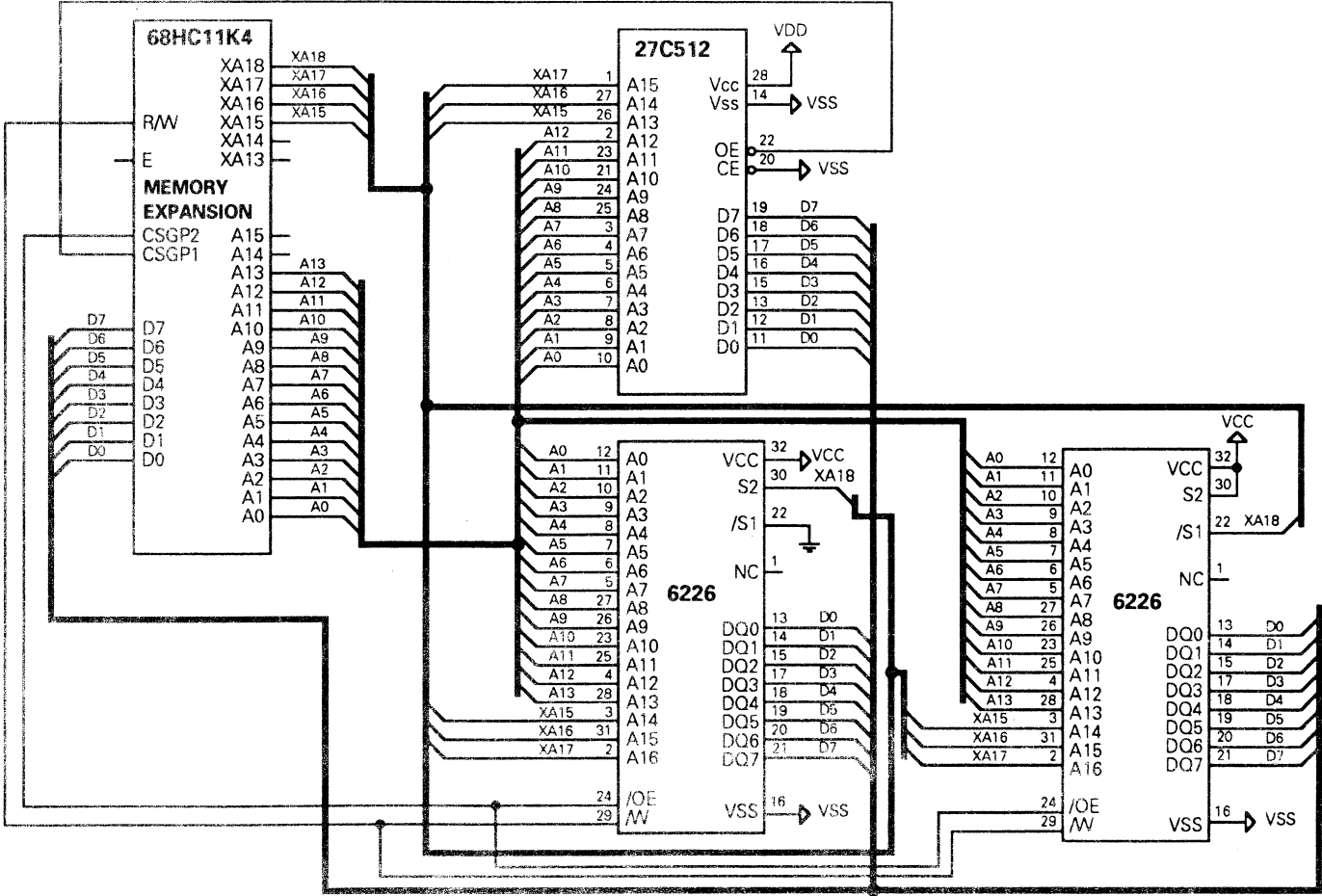


Figure 3b. Alternative 8K Byte window and 16K Byte window

PGAR \$3E - XA14, XA18 MMSIZ \$F2 - Window 1 16K, Window 2 32K
 MMWBR \$40 - one window at \$0000, one at \$4000 CSCTL \$00 - no I/O or program chip select
 GPCS1A \$00 - not relevant GPCS1C \$0A - follow window 1
 GPCS2A \$00 - not relevant GPCS2C \$0B - follow window 2

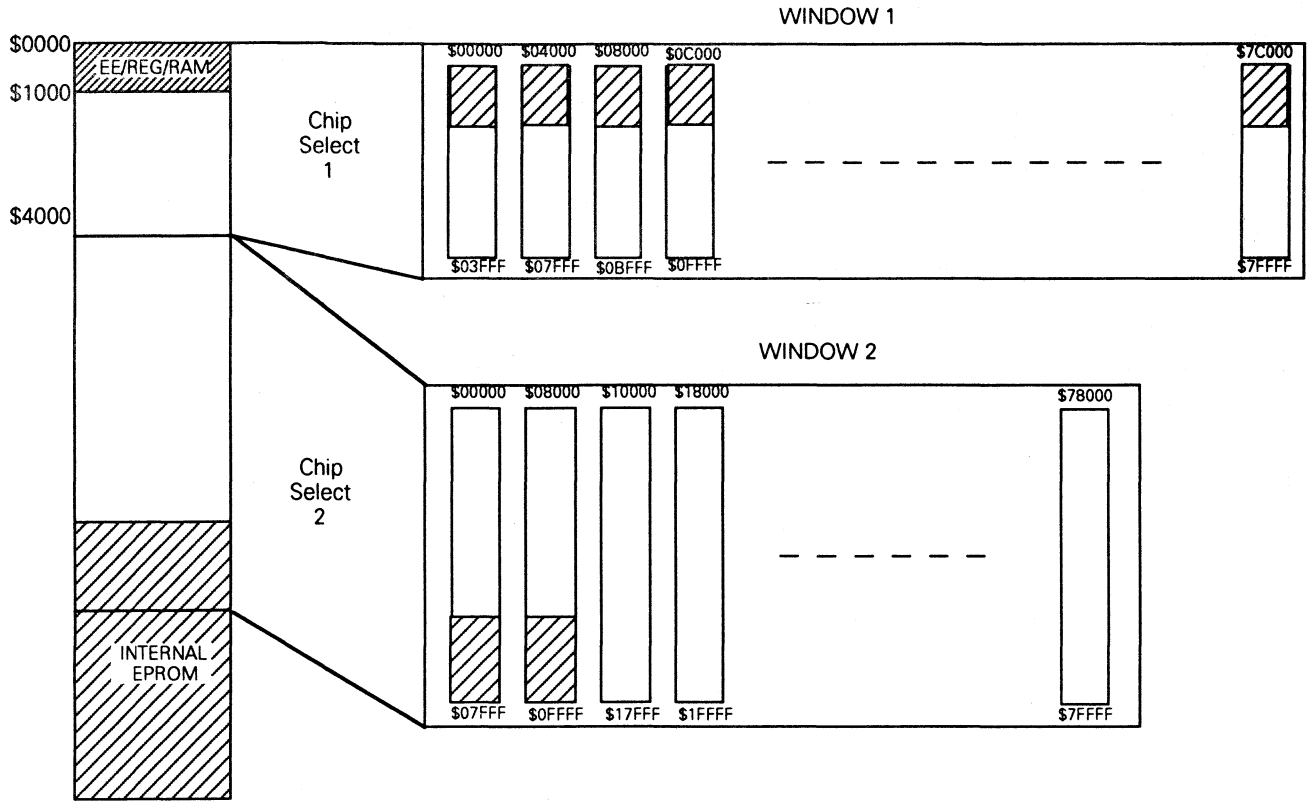


Figure 4a. Possible 1MByte addressing organisation

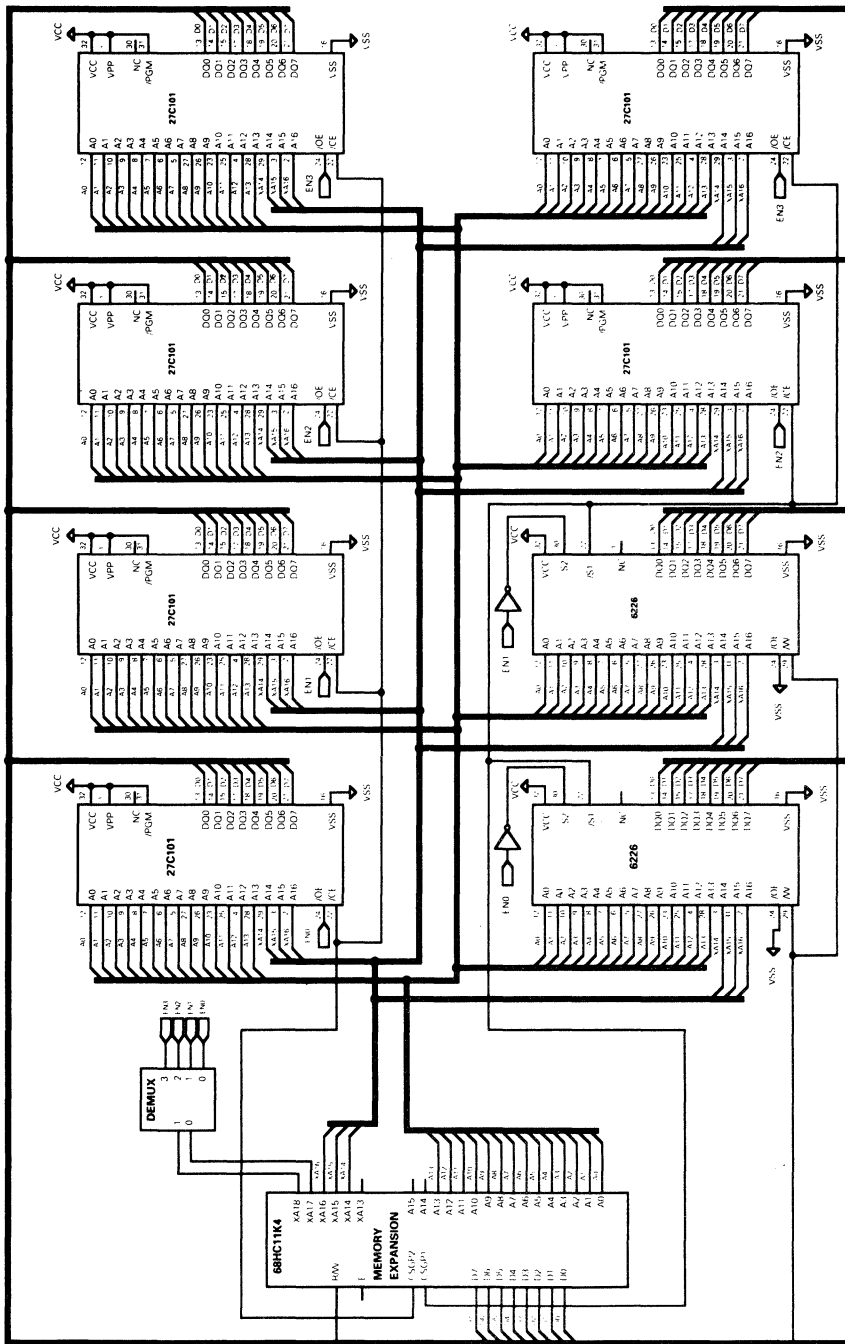


Figure 4b. One MByte in two windows

CONCLUSION

The standard 64K Byte addressing range of the M68HC11 family is easily extended using the MC68HC(7)11K4 memory expansion. This logic also provides programmable chip selects to allow easy interfacing with external memory chips. A similar extended memory system may be implemented on other M68HC11 products by following other systems as described in [2].

REFERENCES

- [1] MC68HC711K4 Technical Summary, Reference BR751/D
- [2] "128K byte addressing with the M68HC11", Reference AN432/D

A Monitor for the MC68HC05E0

Peter Topping,
MCU Applications Group,
Motorola Ltd., East Kilbride

INTRODUCTION

Development systems for single-chip MCUs can be complex and expensive. This can dissuade potential users of this type of microprocessor from designing them into new applications. This application note describes a simple "entry" development system suitable for debugging hardware and software for the M6805 range of microprocessors. The M6805 range includes both CMOS and NMOS parts, most of which are single-chip devices which include mask-programmable ROM, RAM, I/O and one or more timers. The exceptions are the CMOS MC146805E2 and MC68HC05E0 which have no on-chip ROM but instead have data and address buses which enables them to use external memories and peripherals.

The development system uses the MC68HC05E0 processor and can be used to develop applications intended for any M6805 but is particularly suitable for applications which will use the MC68HC05E0 itself. The MC68HC05E0 has a non-multiplexed bus which requires no additional hardware to interface with RAMs, ROMs and EPROMS. It has 480 bytes of RAM, 3 timers, 3 chip-select lines and 4 8-bit ports enabling it to meet many requirements with the addition of only an EPROM containing the application software.

Such a system will be most cost effective in small volume applications not justifying a mask programmed single-chip MCU and also in applications requiring larger programs than can be accommodated in ROMed MCU versions.

For software development, however, RAM and a monitor have to be incorporated so software can be loaded and de-bugged. An MCM6264 8Kbyte (or MCM60L256/MCM6206 32Kbyte) RAM is used. The EBUG05 monitor EPROM, listed at the end of this application note, fulfils the monitor requirement.

In an E0 application the development system can be used as an add-on to the target system hardware, as shown in Figure 1 (components to the left of the dotted line). The application hardware needs very little modification as it already contains the MPU, the interface to the development system being mostly via the socket for the EPROM. Apart from the keyboard and display connections which use port A, only four or five additional connections are required. These can be made available on a small connector on the application PCB. In one of the 8K applications, used to check out the development system (reference 1), a keyboard and display were required, so only four I/O lines (see below) were dedicated to the development system. In a 16K application, A13 (PD5) is also required, so a fifth I/O line would be used. The keyboard provides up to 32 keys using 7 I/O lines and an external 3-line to 8-line decoder. One additional I/O line is used for the display. If the keyboard and/or display is not required in the final application (e.g., reference 2) then the I/O lines, used by them during development, are available for other purposes but their use cannot readily be de-bugged. This is a disadvantage of such a simple development system. There is, however, a major advantage of this system compared with those which use the display and keyboard of a PC. In this system the PC can be used to display the program listing file, obviating the need for printouts during debug.

CIRCUIT

Figure 1 shows the circuit used. The 8K or 16K (half of an MCM60L256) RAM is placed between \$0000 and \$3FFF; this means that the 512 bytes from \$0000 to \$01FF are not used, as they are mapped over the E0's I/O and RAM space. Locations \$0200 to \$021F are used by the monitor, so use of the RAM should start at \$0220 or above. Some RAM may be required to replace E0 page 0 RAM (16 bytes), used by the monitor or for other purposes during de-bug; the application used to check out the monitor used RAM from \$0400. This provides 7K (15K using an MCM60L256) of RAM for the code being de-bugged. The start address chosen for the code being developed can be extended down to \$0220 if this RAM is not required, providing a program space of nearly 15¹/₂Kbytes. The 16K limitation is caused by the split of the memory map into four (see Figure 3), to simplify address decoding and facilitate the ability to load code from an EPROM. If this capability is not required, more complex address decoding would allow up to 60K of RAM to be available for code; the monitor uses less than 4K.

The MC74HC138 provides the chip selects during de-bug as the on-chip chip selects signals are not suitable for the monitor's address map. The final system will usually not require an MC74HC138.

The 28-pin EPROM socket which will contain the final de-bugged code is used during development as the main interface to the monitor and its RAM. Only four or five other signals are required. These are the two or three most significant address lines, A15, A14 and, optionally, A13 and the clock (P02), which are required by the MC74HC138, the R/W line and a port line (PB2) for inputting or outputting code on an RS232 serial link. PB2 can be used in the application as it is only used by the monitor during serial transfers when the application is not running. Care should, however, be taken to avoid contention within the hardware connected to PB2. If only 8K bytes of external RAM are required, then A13 is not required and its pin, PD5, can be used by the application.

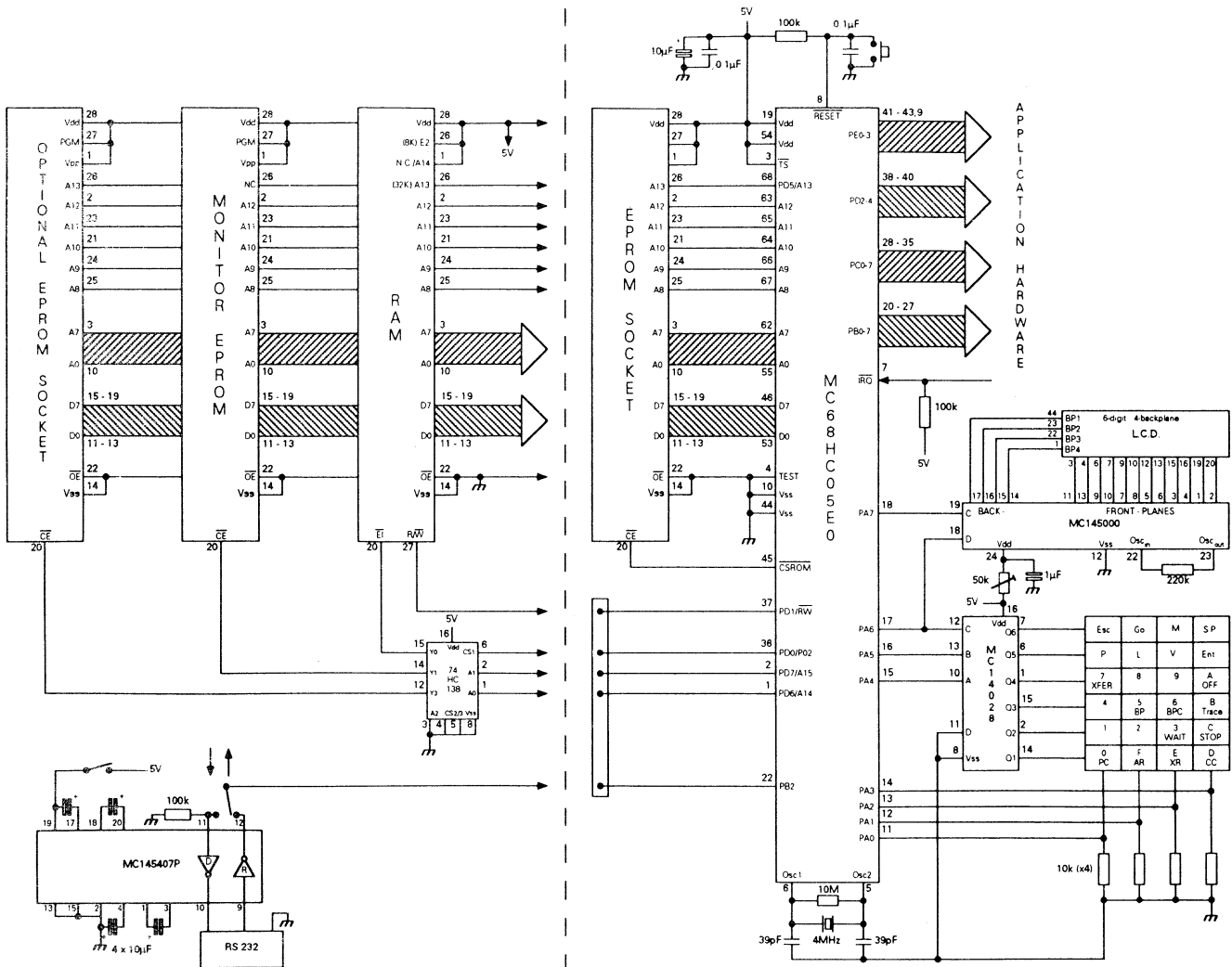
The hardware connected to the EPROM socket can also include a 28-pin socket to accommodate a 27(C)64 or 27(C)128 EPROM. This is addressed between \$4000 and \$7FFF and can be used to introduce software from an EPROM which has been programmed with code for de-bug. The monitor contains a routine which transfers the contents of this EPROM into RAM. Alternatively, code can be loaded serially via the RS232 interface.

The EBUG05 monitor EPROM (27C64) included in the external hardware is enabled from \$C000 to \$FFFF, although its actual start address is \$E000. Figure 3 shows the memory map of the development system.

The monitor display is a 6-digit 4-backplane LCD (e.g., Hamlin type 4200 or the 8-digit GE type LX D69D3F09KG), which is driven by an MC14500P display driver. The driver is controlled by a 2-line serial link from the microprocessor. A single-backplane display can be used as an alternative, as shown in Figure 2. Three MC144115 driver chips are used along with a transistor inverter to supply the additional latch enable signal required by these drivers. This circuit requires more connections to the LCD but allows the use of a more commonly available display.

The optional RS232 interface can be implemented using the single-supply MC145407 driver-receiver chip. If outputting of S-records is not required, then a simple transistor inverter with a pull-up resistor and a reverse polarity protection diode can be used. This interface is shown in Figure 4.

Figure 1. MC68HC05E0-based M6805 development system



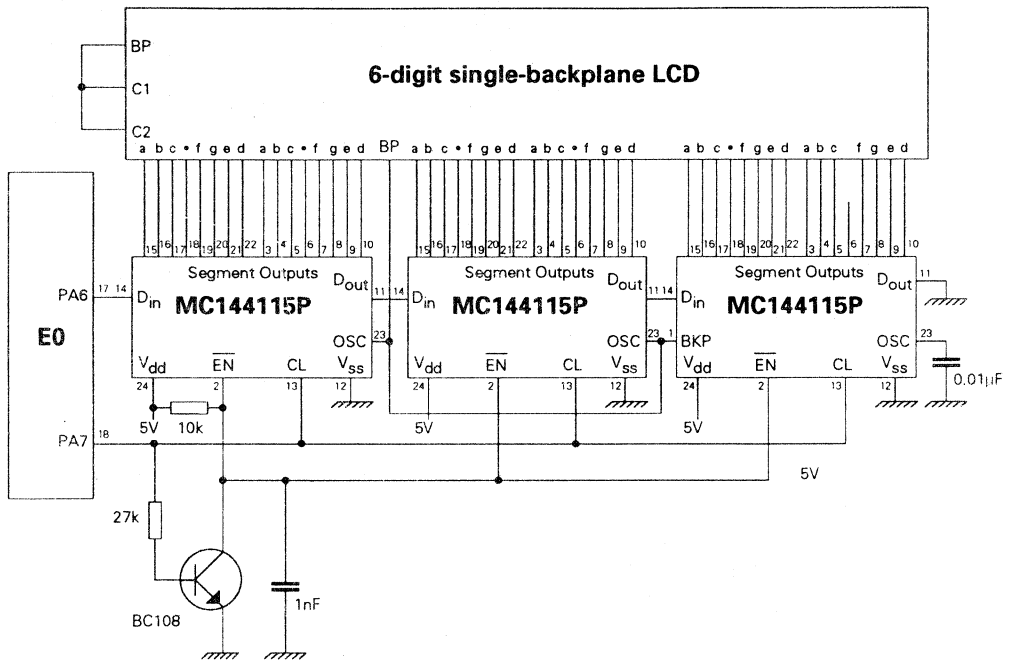


Figure 2. Alternative static LCD display

MC68HC05E0 I/O, timers	0000
	001F
MC68HC05E0 RAM (16 bytes used by EBUG05)	0020
	002F
MC68HC05E0 RAM (464 bytes including stack at 00FF), for use in application	0030
	01FF
MC68HC05E0 RAM (32 bytes used by EBUG05)	0200
	021F
External RAM,* can be used for data and/or program	0220
	3FFF
Optional EPROM socket	4000
	7FFF
not used	8000
	BFFF
EBUG05 monitor	C000
	FFF5
MC68HC05E0 vectors	FFF6
	FFFF

Figure 3. Memory Map

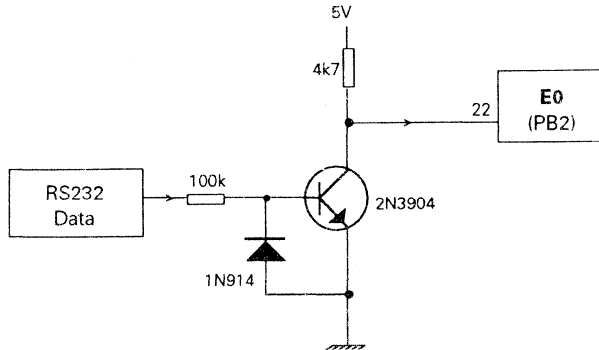


Figure 4. Simple input-only RS232 interface

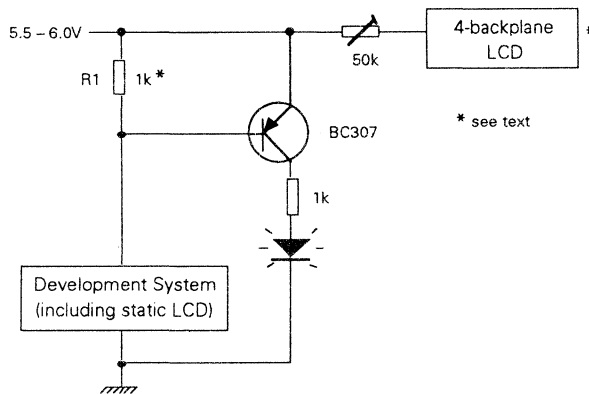


Figure 5. IDD Monitor Circuit

IDD INDICATOR

It is often useful with CMOS circuits to provide a simple IDD monitor which shows via an LED whether or not the IDD is above or below a specific value. In this application it shows whether or not the microprocessor is in the STOP mode. The required circuit is shown in Figure 5. The current threshold can be chosen by selecting the value of R1. A value of 1kohm sets the limit at about $500\mu\text{A}$ which means the LED should be off when the E0 is in STOP mode and on otherwise. The $500\mu\text{A}$ limit allows the LCD and perhaps a CMOS target application to be supplied without switching on the LED. When the microprocessor is not in STOP mode, its IDD is several milliamps and the LED should be lit. If a multiplexed LCD is used, it may be preferable not to supply it via this type of monitor circuit, as a significant change in contrast may occur as the microprocessor goes into its STOP mode (see Figure 5). The monitor drops about 600mV when the microprocessor is running, so the supply voltage should be chosen accordingly. While the microprocessor will operate down to 3V (at an appropriate frequency), the RAM and EPROM chips are specified at $5\text{V} \pm 0.5\text{V}$. For battery operation 4 zinc-carbon or 5 Ni-Cad cells can be used.

EBUG05 KEY FUNCTIONS

The EBUG05 EPROM comprises a monitor, developed from the MC146805E2 CBUG05 program, specifically written to enter and debug 6805 code. The following functions are available using the 24-key keyboard:

Function	Key	Description of function
PC	0	Display program counter.
BP	5	Enable breakpoints. The first breakpoint is displayed if enabled or boff if it isn't. Enter new breakpoint address followed by ENTER to change, or just ENABLE to move to next one, three breakpoints are available. This number can be increased, the only cost being the additional RAM used (3 bytes per breakpoint).
BPC	6	Clear breakpoints. ENTER clears all breakpoints. Entering a number followed by ENTER clears that breakpoint only.
OFF	A	Calculate branch offset. The address of the branch instruction and that of the destination are requested. If a valid branch is calculated, it is written into memory and displayed. If not valid, then "or" for out of range is displayed. A branch of -128 through +127 relative to the start address of the next instruction is allowed.
TR	B	"Trace" one instruction. This is not a true trace as breakpoints are advanced sequentially through the code and do not follow branches or jumps (see below).
STOP	C	Put the E0 into STOP mode, clock stops.
WAIT	3	Put the E0 into WAIT mode.
CC	D	Display/change Condition Code register, new data is followed by ENTER. ESC returns to "□" prompt without changing contents.
XR	E	Display/change index Register, new data is followed by ENTER. ESC returns to "□" prompt without changing contents.
AR	F	Display/change Accumulator, new data is followed by ENTER. ESC returns to "□" prompt without changing contents.
P	P	Output S-records. When this key is pressed, "bA" for begin address is displayed; enter first address (and ENTER), then last address when "EA" is displayed. Another ENTER starts the RS232 S-record output.
L	L	Load S-records from the RS232 interface. Press L and "LOAD" is displayed. S-records sent to 2, PORTB, will be loaded until an S9 record is received. The prompt returns if the load was OK. If not, an error message is displayed (see next section).
V	V	Verify RAM against S-records. "UErIFy" is displayed, otherwise the same as LOAD including error checks.
ENTER	ENT	Enter keyed-in address or data (and move to next address in "M").
ESC	ESC	Escape from current function (except STOP, WAIT, V, L & P).

Function	Key	Description of function
GO	GO	Begin or continue program. When pressed current PC is displayed. To proceed from that location, press ENTER; to proceed from a different address, enter the address followed by ENTER.
M	M	Display/change a location in RAM. When pressed, last address is displayed. Press ENTER to display the contents of this address or enter a new address, followed by ENTER. New contents can be entered (followed by ENTER) if required. ENTER moves to next address; M moves to previous address.
SP	SP	Display stack pointer; cannot be modified.
XFER	7	Transfer code from an EPROM between \$4220 and \$7FFF into RAM (\$0220 to \$3FFF). When pressed, default start address (in RAM) is displayed. Press ENTER to start at this address (\$0400), or enter new start address followed by ENTER. End address (in RAM) is displayed. Press ENTER to transfer code up to this end address (\$1FFF), or enter new end address followed by ENTER. The lowest allowable start address is \$0220 and the highest end address is \$3FFF. During the transfer the code is read from the EPROM at the RAM addresses plus \$4000. The code in EPROM should have been assembled with the start address where it will reside during execution (e.g., \$0400).

USE OF THE MONITOR

The MC68HC05E0's vectors are within the address space of the EBUG05 EPROM. They operate via extended jumps in RAM. This gives the user access to the vectors if these jumps are written to, from within the user's program. The vectors' locations are shown in Table 1. The extended jump instruction (\$CC) is written to RAM by the monitor; all the user has to do is to overwrite the destination addresses at the locations indicated in the Table. This must be done at the start of the user code, before interrupts are enabled. Vectoring the jumps through a table at a fixed location in the user code obviates the need to change this initialisation when re-assembling changes the addresses of the interrupt service routines. Lines of code for this purpose are included as comments in reference 2.

Clearly the RESET vector cannot be altered in this way, so during debug user software should be started using the GO facility in EBUG05. Software interrupts (SWIs) are used by the monitor to facilitate breakpoint and should therefore not be used in the application software.

Software for de-bug should be assembled to reside in RAM, starting at address \$400 (or down to \$220 if more space is required, see above). It can be introduced into the system in an EPROM at \$4000 or loaded serially from a PC via the RS232 interface. The monitor includes a routine to move the code from EPROM to RAM, where it can be executed and debugged using EBUG05. This provides an alternative method of loading code for debug if a serial load is not appropriate. When debug is complete, the code should be re-assembled at \$E000 (or \$C000 if a 16K EPROM is used). When the code is in EPROM, the vectors should be included. The jumps required during debug are then no longer relevant.

CSROM will normally be suitable for selecting this EPROM, so that the MC74HC138 will not be required once the code has been finalised.

Table 1.

Vector	Address	JMP location in RAM
SPI/IIC	FFF4-5	\$0209 (add: \$20A/B)
Timer B	FFF6-7	\$0206 (add: \$207/8)
Timer A	FFF8-9	\$0203 (add: \$204/5)
IRQ	FFFA-B	\$0200 (add: \$201/2)
SWI	FFFC-D	used by monitor
RESET	FFFE-F	use "GO" function

BREAKPOINTS

Up to three breakpoint addresses can be entered; this number can be easily increased if required by changing the EQU on line 25 of EBUG05 source code. To allow the continuation of execution from a breakpoint, a breakpoint at the start address specified by a GO is ignored. This means that a single breakpoint in a loop will only be encountered once if, after it has occurred, execution is re-commenced with a GO from the current address. If it is required to stop at the breakpoint again, then one of the two procedures described here should be used. Two breakpoints can be entered at different places in the loop. They will be encountered alternately; clearly, two GOs are required for each execution of the loop. A second method is to insert only one breakpoint but to trace one instruction before continuing with a GO. This trace takes the program counter past the breakpoint address and so allows the breakpoint to be re-inserted when GO is executed. This will only work reliably if the breakpoint is not on a branch, jump, or any other instruction that could cause the program to proceed to an address other than the next sequential address (see below).

If a single breakpoint has been entered and encountered and execution re-started with a GO from the same address, then pressing RESET is the only method of returning to the monitor. Normally pressing RESET when the program is running with breakpoints valid is not recommended as the application code will be left corrupted with SWIs replacing the instructions at the breakpoint addresses. If, however, control has been lost because execution was recommenced from the address of a single breakpoint, then corruption will not occur as the breakpoint will not have been re-inserted.

Breakpoints are inserted when TRace or GO are executed but removed when a breakpoint is encountered. The trace facility inserts a breakpoint (SWI) at the address of the instruction sequentially after the current instruction and then executes the current instruction. If the current instruction jumps or branches somewhere else then the SWI will not be encountered. This is thus not a true program-following trace but in many cases is more useful as often subroutines do not need to be traced, variables being required to be checked only after each subroutine has been executed. If it is required to trace the program flow to the other address, then another breakpoint should be inserted at that address.

SERIAL LOAD

To load external Motorola S-records, the serial load key (L) should be pressed. The LCD will display "LOAD". S-records should then be supplied at 9600 baud (8-bit, no parity) on the RS232 interface. When an S9 termination record is received, the prompt returns. If an error is detected during a serial load, the load routine stops and displays the address at which the error occurred and the error type. The following error types are possible.

- 1: Checksum error, transmitted data or interface faulty.
- 2: RAM read-back error, RAM faulty or nonexistent.
- 3: ASCII character less than \$30 (0) received.
- 4: ASCII character between \$39 (9) and \$41 (A) received.
- 5: ASCII character more than \$46 (F) received.
- 7: Verify error when comparing S-records with RAM.

The verify function can be used to check that the RAM has not been corrupted. The VERIFY function is used exactly like LOAD except that RAM is compared with, rather than loaded by, the S-records.

The S-record format can be seen in the example below. The number following the S is the record type. The monitor only recognises S1 (8-bit data) and S9 (termination) records. The next byte (23 in most of the records) is the number (in hex) of bytes which follow. This is followed by a 2-byte address (e.g., E000 and FFF4), then the data. The last byte is a checksum byte. The sum of all bytes, including the byte count and the checksum byte, is \$XXFF. This can be easily checked on the S9 record (03+00+00+FC=FF).

```
S123E000CDE05F24FB5FB728D6E09FB128270BC1E0BF273A5C5C5C20EEA601B70E5CDC50
S123E020E09FA6E3B7123F00A6F0B705A658B701A6FBB7063F02A6FFB7073F03A61CB708FF

S123E5E0305A26F8AD3020EAA1102705A11126E29981AE04BF28B72D4444444497D6E4CD2C
S123E600BE28E720B62DA40F97D6E4CDBE28E721CDE1ECB62D81B72EBF2CB6305FADD5B61C
S10DE6202BAE02ADCFB62EBE2C8146
S10FFFF4E022E022E022E04EE022E022C5
S9030000FC
```

SERIAL INTERFACE

Figure 6 shows a suggested method of wiring up the RS232 sockets in a system with both loading and dumping capabilities. This arrangement facilitates use of the serial LOAD and DUMP routines of the monitor either via a PC COM port or between a host and terminal connected by an RS232 link. When using a PC, the "host" socket should be used. As only one pin on the MC68HC05E0 is used, switching is required to make the required connections. S2 can be eliminated (or left at "L") if only loading is required, as will often be the case. To save power in battery applications, the RS232 interface chip can be switched off using S1. Table 2 shows possible methods of use.

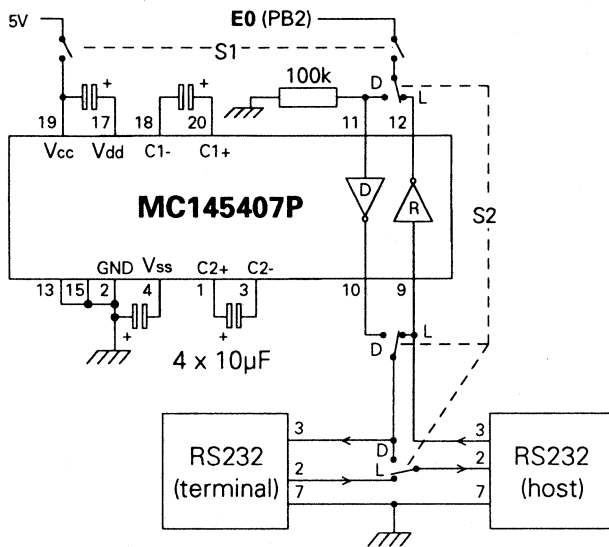


Figure 6. RS232 serial interface with Load/Dump switching

Table 2.

Set-up	Function	S1	S2	Comments
Host & terminal	Load	On	L	Terminal and host connected. Micro looks at data sent from host to terminal (pins 3).
	Dump	On	D	Connection between terminal and host broken, S-records sent to both host (2) and terminal (3).
PC "COM" port	Load	On	L	S-records loaded from pin 3.
	Dump	On	D	S-records sent to pin 2 on "host" socket (and pin 3 on "terminal" socket).

REFERENCES

- 1) AN441/D, An EPROM Emulator using the MC68HC05E0.
- 2) AN460/D, An RDS Decoder using the MC68HC05E0.

APPENDIX – EBUG05 E0 monitor listing

0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011 0020
0012
0013 0000
0014 0001
0015 0002
0016 0003
0017 0004
0018 0005
0019 0006
0020 0007
0021 0008
0022 0009
0023 000c
0024 0012
0025 0003
0026
0027 0020
0028
0029
0030
0031
0032
0033 0021
0034 0022
0035 0023
0036 0024
0037 0025
0038 0026
0039 0027
0040 0029
0041 002a
0042 002b
0043 002c
0044 002d
0045 002e
0046
0047 0200
0048
0049 0200
0050 0203
0051 0206
0052 0209
0053 020c
0054 0212
0055 0213
0056 0214
0057 0215
0058 0216
0059
0060
0061 e000
0062

```

.....
*
*                               EBUG05 E0 monitor.
*
*                               P. Topping                               10th November '91
*
.....
                                ORG      $0020
PORTA EQU 0                      PORT A ADDRESS
PORTB EQU 1                      * B *
PORTC EQU 2                      * C *
PORTD EQU 3                      * D *
PORTE EQU 4                      * E *
PORTAD EQU 5                     PORT A DATA DIRECTION REG.
PORTBD EQU 6                     * B *
PORTCD EQU 7                     * C *
PORTDD EQU 8                     * D *
PORTED EQU 9                     * E *
TCR EQU $0C                      TIMER CONTROL REGISTER
PORTDSF EQU $12                 PORT D ALTERNATIVE FUNCTION REG.
NBKPT EQU 3
STAT RMB 1                       0: Not first SWI after RESET
*                               1: PROCEED (0: GO)
*                               4: No address entered (BLDRNG)
*                               5: Verifying (TLOAD)
*                               6: Register contents being changed
*                               7: Last S1 record (PUNCH)
WORK2 RMB 1                      RAM SUB-ROUTINE LDA/STA
ADDRH RMB 1                      * * ADDRESS H
ADDRL RMB 1                      * * ADDRESS L
WORK3 RMB 1                      * * RTS
WORK5 RMB 1
WORK6 RMB 1
TEMP RMB 2                       TEMP. ADDRESS
PNCNT RMB 1                      No. BREAKPOINTS
CHKSUM RMB 1                     CHECKSUM (SERIAL)
COUNT RMB 1                    BIT COUNTER
TMP1 RMB 1                      TEMP (SERIAL)
TMP2 RMB 1                      * *
BCNT RMB 1                      BYTE COUNT
                                ORG      $0200
IRQ RMB 3                       DE-BUG RAM VECTOR, IRQ
TIRQA RMB 3                     * * * TIMER A
TIRQB RMB 3                     * * * TIMER B
SIRO RMB 3                      * * * SERIAL
DTABL RMB 6                     DISPLAY TABLE
WORK1 RMB 1
WORK4 RMB 1
TMPTCR RMB 1                    TEMP. TCR (XROM USED BY MONITOR)
PROP RMB 1                      INSTRUCTION (PROCEED)
BKPTBL RMB 3*NBKPT             B.P. TABLE
                                ORG      $E000

```



```

0063
0064
0065
0066
0067
0068
0069 e000 a6 f0
0070 e002 b7 05
0071 e004 a6 ff
0072 e006 b7 12
0073
0074 e008 a6 cc
0075 e00a c7 02 00
0076 e00d c7 02 03
0077 e010 c7 02 06
0078 e013 c7 02 09
0079 e016 c6 e0 ac
0080 e019 c7 02 01
0081 e01c c6 e0 ad
0082 e01f c7 02 02
0083 e022 c6 e0 ae
0084 e025 c7 02 07
0085 e028 c6 e0 af
0086 e02b c7 02 08
0087 e02e c6 e0 b0
0088 e031 c7 02 04
0089 e034 c6 e0 b1
0090 e037 c7 02 05
0091 e03a c6 e0 b2
0092 e03d c7 02 0a
0093 e040 c6 e0 b3
0094 e043 c7 02 0b
0095
0096 e046 ae 20
0097 e048 7f
0098 e049 5c
0099 e04a a3 2e
0100 e04c 23 fa
0101
0102 e04e cd e6 1f
0103 e051 a6 ff
0104 e053 d7 02 16
0105 e056 5c
0106 e057 5c
0107 e058 5c
0108 e059 3a 29
0109 e05b 26 f6
0110 e05d 83
0111
0112

```

```

*****
*
*          Reset.
*
*****
RESET          #SFO          SETUP PORT
              STA          CLR KEYPAD
              LDA          SETUP PORTS
              STA          PORT F      FOR ADDRESSES ETC.

              LDA          #SCC          VECTORS IN RAM
              STA          IRQ
              STA          TIRQA
              STA          TIRQB
              STA          SIRQ
              LDA          VECTOR
              STA          IRQ+1
              LDA          VECTOR+1
              STA          IRQ+2
              LDA          VECTOR+2
              STA          TIRQB+1
              LDA          VECTOR+3
              STA          TIRQB+2
              LDA          VECTOR+4
              STA          TIRQA+1
              LDA          VECTOR+5
              STA          TIRQA+2
              LDA          VECTOR+6
              STA          SIRQ+1
              LDA          VECTOR+7
              STA          SIRQ+2

INIT          LDX          #STAT
              CLR          0, X          CLEAR
              INCX          WORKING
              CPX          #BCNT        STORAGE
              BLS          INIT

              JSR          SCNBKP        CLEAR
              LDA          #SFF          ALL
              STA          BRPTBL, X    BREAKPOINTS
              INCX
              INCX
              INCX
              DEC          PNCNT
              BNE          REBCLR
              SWI

```

```

0113
0114
0115
0116
0117
0118
0119 e05e 00 20 04      SWI      BRSET    0, STAT, SWICHK  FROM RESET?
0120 e061 10 20          BSET    0, STAT      YES
0121 e063 20 4f          BRA     GETCMD
0122 e065 b6 0c          SWICHK  LDA     TCR      GET CURRENT TCR
0123 e067 ae 0c          LDX    #SOC          SET XROM SO THAT EXTERNAL RAM
0124 e069 bf 0c          STX    TCR          CAN BE WRITTEN TO
0125 e06b c7 02 14       STA    TMPTR        SAVE CURRENT TCR
0126 e06e cd e6 74       STAY   JSR    KEYSCN
0127 e071 cd e6 1f       JSR    SCNBKP      REMOVE
0128 e074 d6 02 16       SWIREP  LDA    BKPTBL, X  BREAKPOINTS
0129 e077 2b 0d          BMI    SWINOB
0130 e079 b7 22          STA    ADDRH
0131 e07b d6 02 17       LDA    BKPTBL+1, X
0132 e07e b7 23          STA    ADDR1
0133 e080 d6 02 18       LDA    BKPTBL+2, X
0134 e083 cd e7 57       JSR    STORE
0135 e086 5c             SWINOB  INCX      GET NEXT B.P.
0136 e087 5c             INCX
0137 e088 5c             INCX
0138 e089 3a 29          DEC    PNCNT
0139 e08b 26 e7          BNE    SWIREP
0140 e08d cd e1 23       JSR    LOCSTK      FIND STACK
0141 e090 e6 08          LDA    8, X
0142 e092 a0 01          SUB    #1          ADJUST PC
0143 e094 e7 08          STA    8, X        (MINUS 1)
0144 e096 b7 23          STA    ADDR1
0145 e098 e6 07          LDA    7, X
0146 e09a a2 00          SBC    #0
0147 e09c e7 07          STA    7, X
0148 e09e b7 22          STA    ADDRH
0149
0150 e0a0 03 20 06       BRCLR  1, STAT, NOPRO  PROCEED ?
0151 e0a3 c6 02 15       LDA    PROP
0152 e0a6 cd e7 57       JSR    STORE
0153
0154 e0a9 cc e1 35       NOPRO  JMP     PCOUNT    PRINT P.C.
0155
0156
0157
0158
0159
0160
0161
0162 e0ac e8 9c          VECTOR  FDB     IRQV      IRQ
0163 e0ae e8 9c          FDB     TIRQBV     TIMER B
0164 e0b0 e8 9c          FDB     TIRQAV     TIMER A
0165 e0b2 e8 9c          FDB     SIRQV      SERIAL
0166
0167

```

```

0168
0169
0170
0171
0172
0173
0174 e0b4 cd e6 42
0175 e0b7 a6 e4
0176 e0b9 c7 02 0c
0177 e0bc cd e6 4c
0178 e0bf cd e6 74
0179 e0c2 24 fb
0180 e0c4 5f
0181 e0c5 c7 02 12
0182 e0c8 d6 e0 df
0183 e0cb c1 02 12
0184 e0ce 27 0b
0185 e0d0 c1 e1 1f
0186 e0d3 27 df
0187 e0d5 5c
0188 e0d6 5c
0189 e0d7 5c
0190 e0d8 5c
0191 e0d9 20 ed
0192 e0db 5c
0193 e0dc dc e0 df
0194
0195 e0df 11
0196 e0e0 cc e1 35
0197 e0e3 12
0198 e0e4 cc e1 4f
0199 e0e7 14
0200 e0e8 cc e1 6d
0201 e0eb 18
0202 e0ec cc e1 8d
0203 e0ef.28
0204 e0f0 cc e8 9d
0205 e0f3 24
0206 e0f4 cc e8 a3
0207 e0f7 32
0208 e0f8 cc e2 cf
0209 e0fb 34
0210 e0fc cc e3 46
0211 e0ff 38
0212 e100 cc e3 a7
0213 e103 41
0214 e104 cc e8 16
0215 e107 48
0216 e108 cc e2 19
0217 e10b 51
0218 e10c cc e4 c5
0219 e10f 52
0220 e110 cc e3 f1
0221 e113 54
0222 e114 cc e3 eb
0223 e117 62
0224 e118 cc e5 aa
0225 e11b 64
0226 e11c cc e6 e3
0227 e11f 68
0228 e120 cc e6 25
0229
0230
0231
0232
0233
0234
0235
0236
0237 e123 ad 01
0238 00e1
0239 0025
0240 e125 81
0241 e126 ae 7f
0242 e128 a6 e1
0243 e12a 5a
0244 e12b f1
0245 e12c 26 fc
0246 e12e a6 25
0247 e130 e1 01
0248 e132 26 f4
0249 e134 81
0250

*****
*
*      Print logo & scan for keypress.
*
*****

GETCMD JSR   CLR TAB
        LDA   #E4          PRINT
        STA   DTABL
DSCN   JSR   DISTAB      PROMPT
CMDSCN JSR   KEYS CN      CHECK KEYPAD
        BCC   CMDSCN
        CLR X
        STA   WORK1
RJUMP  LDA   PTABL, X    THIS COMMAND?
        CMP   WORK1
        BEQ   PJUMP      YES
        CMP   LAST
        BEQ   GETCMD
        INCX
        INCX
        INCX
        INCX
        BRA   RJUMP
PJUMP  INCX
        JMP   PTABL, X
        NO
        GO TO
        NEXT
        POSSIBLE
        TRY AGAIN
        GO TO
        COMMAND

PTABL  FCB   $11
        JMP   P COUNT    0 PROGRAM COUNTER
        FCB   $12
        JMP   AREG       F ACCUMULATOR
        FCB   $14
        JMP   XREG       S INDEX REGISTER
        FCB   $18
        JMP   C CODE    D CONDITION CODE
        FCB   $28
        JMP   PWRDWN    C STOP
        FCB   $24
        JMP   WDWN      3 WAIT
        FCB   $32
        JMP   BPDIS    8 DISPLAY/SET BP
        FCB   $34
        JMP   BPCLR    9 CLEAR BP
        FCB   $38
        JMP   PROC     B PROCEED TO NEXT INSTRUCTION
        FCB   $41
        JMP   XFER     7 TRANSFER FROM EPROM TO RAM (OFFSET: $3C00)
        FCB   $48
        JMP   OFFSET   A OFFSET CALCULATION
        FCB   $51
        JMP   PUNCH    P OUTPUT S-RECORDS
        FCB   $52
        JMP   TLOAD    L LOAD S-RECORDS
        FCB   $54
        JMP   VERIFY   V VERIFY S-RECORDS
        FCB   $62
        JMP   NEWGO    G NEW GO (SKIP CURRENT BP)
        FCB   $64
        JMP   MEMEX    M MEMORY INSPECT/MODIFY
LAST   FCB   $68
        JMP   STACK    S STACK
        FCB   $68
        JMP   STACK
        FCB   $68
        JMP   STACK
        FCB   $68
        JMP   STACK

*****
*
*      Search for stack pointer, X <- SP-3
*
*****

LOCSTK BSR   LOCST2
STKHI  EQU   -$8000+*/256+$80
STKLOW EQU   -256*STKHI+*
        RTS
LOCST2 LDX   #$7F
LOCLOP LDA   #STKHI
LOC DWN DECX
        CMP   0, X
        BNE  LOCDWN
        LDA  #STKLOW
        CMP  1, X
        BNE  LOCLOP
        RTS

```

```

0251
0252
0253
0254
0255
0256
0257 e135 a6 73
0258 e137 c7 02 10
0259 e13a a6 d1
0260 e13c c7 02 11
0261 e13f ad e2
0262 e141 e6 07
0263 e143 b7 22
0264 e145 e6 08
0265 e147 b7 23
0266 e149 cd e8 02
0267 e14c cc e0 bf
0268
0269
0270
0271
0272
0273
0274
0275 e14f a6 77
0276 e151 c7 02 0c
0277 e154 c7 02 0f
0278 e157 a6 d1
0279 e159 c7 02 0d
0280 e15c c7 02 0e
0281 e15f ad c2
0282 e161 9f
0283 e162 ab 05
0284 e164 3f 22
0285 e166 b7 23
0286 e168 1c 20
0287 e16a cc e6 ea
0288
0289
0290
0291
0292
0293
0294
0295
0296 e16d cd e6 42
0297 e170 a6 06
0298 e172 c7 02 0d
0299 e175 a6 e6
0300 e177 c7 02 0e
0301 e17a a6 60
0302 e17c c7 02 0f
0303 e17f ad a2
0304 e181 9f
0305 e182 ab 06
0306 e184 3f 22
0307 e186 b7 23
0308 e188 1c 20
0309 e18a cc e6 ea
0310
0311
0312
0313
0314
0315
0316
0317 e18d cd e6 42
0318 e190 a6 d1
0319 e192 c7 02 0c
0320 e195 a6 d7
0321 e197 c7 02 0d
0322 e19a a6 e6
0323 e19c c7 02 0e
0324 e19f a6 f1
0325 e1a1 c7 02 0f
0326 e1a4 cd e1 23
0327 e1a7 9f
0328 e1a8 ab 04
0329 e1aa 3f 22
0330 e1ac b7 23
0331 e1ae 1c 20
0332 e1b0 cc e6 ea
0333
0334

```

```

*****
*
*       Display program counter.
*
*****
PCOUNT LDA    #S73          PRINT
        STA    DTABL+4     'PC'
        LDA    #SD1
        STA    DTABL+5
        BSR    LOCSTK      FIND USER PC
        LDA    7,X         HIGH BYTE
        STA    ADDRH
        LDA    8,X         LOW BYTE
        STA    ADDRDL      PRINT IT
        JSR    PRTADR
        JMP    CMDSCN

*****
*
*       Accumulator examine/change.
*
*****
AREG   LDA    #S77          PRINT 'ACCA'
        STA    DTABL
        STA    DTABL+3
        LDA    #SD1
        STA    DTABL+1
        STA    DTABL+2
        BSR    LOCSTK      FIND ACCUM. VALUE
        TXA
        ADD    #5
        CLR    ADDRH       SETUP FOR
        STA    ADDRDL      EXAMINE/CHANGE
        BSET   6,STAT
        JMP    MEMEX3      USING MEMORY ROUTINE

*****
*
*       Index reg. examine/change.
*
*****
XREG   JSR    CLRRTAB
        LDA    #6          PRINT 'Idx'
        STA    DTABL+1
        LDA    #SE6
        STA    DTABL+2
        LDA    #S60
        STA    DTABL+3
        BSR    LOCSTK      FIND INDEX
        TXA               REGISTER VALUE
        ADD    #6
        CLR    ADDRH       SETUP FOR
        STA    ADDRDL      EXAMINE/CHANGE
        BSET   6,STAT
        JMP    MEMEX3      USING MEMORY ROUTINE

*****
*
*       CC reg. examine/change.
*
*****
CCODE  JSR    CLRRTAB
        LDA    #SD1
        STA    DTABL
        LDA    #SD7
        STA    DTABL+1
        LDA    #SE6
        STA    DTABL+2
        LDA    #SF1
        STA    DTABL+3
        JSR    LOCSTK      FIND CONDITION
        TXA               CODES
        ADD    #4
        CLR    ADDRH       SETUP FOR
        STA    ADDRDL      EXAMINE/CHANGE
        BSET   6,STAT
        JMP    MEMEX3      USING MEMORY ROUTINE

```

```

0335
0336
0337
0338
0339
0340
0341
0342
0343 e1b3 19 20
0344 e1b5 cd e6 42
0345 e1b8 a6 f4
0346 e1ba c7 02 10
0347 e1bd a6 77
0348 e1bf c7 02 11
0349 e1c2 cd e6 4c
0350 e1c5 cd e7 a0
0351 e1c8 24 25
0352 e1ca b6 22
0353 e1cc b7 27
0354 e1ce b6 23
0355 e1d0 b7 28
0356 e1d2 cd e7 46
0357 e1d5 b7 26
0358 e1d7 cd e6 42
0359 e1da a6 f1
0360 e1dc c7 02 10
0361 e1df a6 77
0362 e1e1 c7 02 11
0363 e1e4 cd e6 4c
0364 e1e7 cd e7 a0
0365 e1ea 24 03
0366 e1ec b6 22
0367 e1ee 81
0368 e1ef 18 20
0369 e1f1 81
0370
0371
0372
0373
0374
0375
0376
0377 e1f2 bf 26
0378 e1f4 3f 2b
0379 e1f6 be 26
0380 e1f8 d6 e2 0d
0381 e1fb be 2b
0382 e1fd d7 02 0c
0383 e200 3c 26
0384 e202 3c 2b
0385 e204 b6 2b
0386 e206 a1 06
0387 e208 25 ec
0388 e20a cc e6 4c
0389
0390 e20d 00 00 d0 d7 77 e6
0391 e213 d6 f1 60 06 71 b6
0392
0393

```

```

*****
*
*   Build a beginning and ending
*   address range.
*   TEMP,TEMP+1 - ADDRH,ADDRL.
*
*****

```

```

BLDRNG BCLR 4,STAT
        JSR CLRTAB          PRINT
        LDA #SF4           `BA`
        STA DTABL+4
        LDA #S77
        STA DTABL+5
        JSR DISTAB
        JSR BLDADR          GET SOURCE ADDR.
        BCC BLDRN1         VALID?
        LDA ADDRH          YES
        STA TEMP           NO SAVE IT
        LDA ADDRDL
        STA TEMP+1
        JSR LOAD           FETCH OPCODE OF INSTR.
        STA WORK6         SAVE IT
        JSR CLRTAB
        LDA #SF1           PRINT `EA`
        STA DTABL+4
        LDA #S77
        STA DTABL+5
        JSR DISTAB
        JSR BLDADR          GET DESTINATION ADDR
        BCC BLDRN1         VALID?
        LDA ADDRH          YES
        RTS
BLDRN1 BSET 4,STAT        INVALID
        RTS

```

```

*****
*
*   Display message.
*
*****

```

```

DISP STX WORK6
DISLP CLR COUNT
        LDX WORK6
        LDA DLOAD,X
        LDX COUNT
        STA DTABL,X
        INC WORK6
        INC COUNT
        LDA COUNT
        CMP #6
        BLO DISLP
        JMP DISTAB
DLOAD FCB 0,0,$D0,$D7,$77,$E6
VERF FCB $D6,$F1,$60,$06,$71,$B6

```

```

0394
0395
0396
0397
0398
0399
0400 e219 ad 98
0401 e21b 08 20 3e
0402 e21e b6 23
0403
0404 e220 a0 02
0405 e222 b7 23
0406 e224 b6 22
0407 e226 a2 00
0408 e228 b7 22
0409 e22a b6 23
0410 e22c b0 28
0411
0412 e22e b7 23
0413 e230 b6 22
0414 e232 b2 27
0415 e234 b7 22
0416 e236 b6 26
0417 e238 a1 1f
0418 e23a 23 50
0419 e23c b6 22
0420 e23e a1 ff
0421 e240 27 0b
0422 e242 4d
0423 e243 26 7a
0424
0425 e245 b6 23
0426 e247 a1 7f
0427 e249 22 74
0428 e24b 20 0a
0429
0430 e24d b6 23
0431 e24f a1 ff
0432 e251 27 6c
0433
0434 e253 a1 80
0435 e255 25 68
0436
0437 e257 ad 06
0438 e259 cc e0 bf
0439 e25c cc e0 b4
0440
0441 e25f cd e6 42
0442 e262 a6 d6
0443 e264 c7 02 0c
0444 e267 a6 b5
0445
0446

```

```

*****
*
*          Calculate branch offset.
*
*****

```

OFFSET	BSR	BLDRNG	
	BRSET	4,STAT,ORET	
	LDA	ADDRL	NO FIND APPARENT
	SUB	#2	
	STA	ADDRL	
	LDA	ADDRH	
	SBC	#0	
	STA	ADDRH	
	LDA	ADDRL	
	SUB	TEMP+1	OFFSET
	STA	ADDRL	
	LDA	ADDRH	
	SBC	TEMP	
	STA	ADDRH	
	LDA	WORK6	CHECK OP CODE
	CMP	#\$1F	FOR BIT BRANCH
	BLS	OFFST1	
	LDA	ADDRH	
	CMP	#\$FF	+ OR - OFFSET?
	BEQ	OFFST2	
	TSTA		CHECK OFFSET
	BNE	OVRERR	FOR +/- 0
	LDA	ADDRL	
	CMP	#\$7F	
	BHI	OVRERR	
	BRA	OK1	
OFFST2	LDA	ADDRL	
	CMP	#\$FF	
	BEQ	OVRERR	
	CMP	#\$80	
	BLO	OVRERR	
OK1	BSR	USE	PRINT IT IF VALID
	JMP	CMDSCN	
ORET	JMP	GETCMD	
USE	JSR	CLRTAB	
	LDA	#\$D6	PRINT `USED`
	STA	DTABL	
	LDA	#\$B5	

```

0447
0448
0449
0450
0451
0452
0453 e269 7 02 0d          STA DTABL+1
0454 e26. .6 f1           LDA #SF1
0455 e26. -7 02 0e        STA DTABL+2
0456 e271 6 e6           LDA #SE6
0457 e273 7 02 0f        STA DTABL+3
0458 e276 6 23           LDA ADDRL          PRINT OFFSET
0459 e278 d e7 d6        JSR PRDAT
0460 e27b 7              TAX
0461 e27. o6 28          LDA TEMP+1
0462 e27. .1b 01         ADD #1
0463 e280 o7 23          STA ADDRHL
0464 e28  o6 27          LDA TEMP
0465 e284 a9 00          ADC #0             PUT INTO
0466 e28e o7 22          STA ADDRHL        INSTRUCTION
0467 e288 f              TXA
0468 e289 cc e7 57       JMP STORE
0469
0470 e28c o6 23          OFFST1 LDA ADDRHL  ADJUST FOR
0471 e28e a0 01          SUB #1             BIT BRANCH
0472 e290 o7 23          STA ADDRHL
0473 e292 o6 22          LDA ADDRHL
0474 e294 a2 00          SBC #0
0475 e296 o7 22          STA ADDRHL
0476 e298 a1 ff          CMP #FFF           NEG OFFSET?
0477 e29a 27 0b          BEQ OFFST3        YES
0478 e29c 4d             TSTA              CHECK FOR
0479 e29d 26 20          BNE OVRERR        +/- 0 AND -1
0480 e29f o6 23          LDA ADDRHL
0481 e2a1 a1 7f          CMP #S7F
0482 e2a3 22 1a          BHI OVRERR
0483 e2a5 20 0e          BRA OK2
0484
0485 e2a7 b6 23          OFFST3 LDA ADDRHL
0486 e2a9 a1 ff          CMP #FFF
0487 e2ab 27 12          BEQ OVRERR
0488 e2ad a1 fe          CMP #SFE
0489 e2af 27 0e          BEQ OVRERR        BHI ?
0490 e2b1 a1 80          CMP #S80
0491 e2b3 25 0a          BLO OVRERR
0492
0493 e2b5 3c 28          OK2 INC TEMP+1
0494 e2b7 26 02          BNE OFFITS
0495 e2b9 3c 27          INC TEMP
0496 e2bb ad a2          OFFITS BSR USE     PRINT IF VALID
0497 e2bd 20 0d          BRA CMDJMP
0498
0499 e2bf a6 d7          OVRERR LDA #SD7     PRINT OR
0500 e2c1 c7 02 10        STA DTABL+4
0501 e2c4 a6 60          LDA #S60
0502 e2c6 c7 02 11        STA DTABL+5
0503 e2c9 cd e8 02        JSR PRDAT
0504 e2cc cc e0 bf        CMDJMP JMP CMDSCN
0505
0506

```

```

0507
0508
0509
0510
0511
0512
*****
*
*           Display/set breakpoints.
*
*****
0513 e2cf 3f 26      BPDIS  CLR      WORK6
0514 e2d1 3a 26      DEC      WORK6
0515 e2d3 cd e6 1f   JSR      SCNBKP      FIND B.P. TABLE
0516 e2d6 bf 21      STX      WORK2
0517 e2d8 4f         BPDIS1 CLRA
0518 e2d9 c7 02 10   STA      DTABL+4
0519 e2dc d6 02 16   LDA      BKPTBL,X   GET B.P.
0520 e2df 2a 14      BPL      BPDIS2     VALID?
0521 e2e1 a6 f4      LDA      #SF4       NO
0522 e2e3 c7 02 0c   STA      DTABL      PRINT `BOFF`
0523 e2e6 a6 d7      LDA      #SD7
0524 e2e8 c7 02 0d   STA      DTABL+1
0525 e2eb a6 71      LDA      #S71
0526 e2ed c7 02 0e   STA      DTABL+2
0527 e2f0 c7 02 0f   STA      DTABL+3
0528 e2f3 20 0a      BRA      BPDIS4
0529 e2f5 b7 22      BPDIS2 STA      ADDRH     PRINT B.P.
0530 e2f7 d6 02 17   LDA      BKPTBL+1,X
0531 e2fa b7 23      STA      ADDRRL
0532 e2fc cd e8 02   JSR      PRTADR
0533 e2ff 3c 26      BPDIS4 INC      WORK6   PRINT B.P. #
0534 e301 be 26      LDX      WORK6
0535 e303 d6 e7 6e   LDA      CTABL,X
0536 e305 c7 02 11   STA      DTABL+5
0537 e309 cd e6 4c   JSR      DISTAB
0538 e30c cd e7 a0   JSR      BLDADR     NEW B.P.
0539 e30f be 21      LDX      WORK2
0540 e311 25 08      BCS      BPDIS7     YES
0541 e313 a1 10      CMP      #S10       NO,ESC?
0542 e315 27 19      BEQ      BPRET      GET OUT
0543 e317 a1 11      CMP      #S11       ENTER?
0544 e319 27 0a      BEQ      BPDIS5     GET NEXT B.P.
0545 e31b b6 22      BPDIS7 LDA      ADDRH
0546 e31d d7 02 16   STA      BKPTBL,X   STORE NEW B.P.
0547 e320 b6 23      LDA      ADDRRL
0548 e322 d7 02 17   STA      BKPTBL+1,X
0549 e325 5c         BPDIS5 INCX
0550 e326 5c         INCX
0551 e327 5c         INCX
0552 e328 bf 21      STX      WORK2
0553 e32a 3a 29      DEC      PNCNT
0554 e32c 26 aa      BNE      BPDIS1     DONE?
0555 e32e 20 9f      BRA      BPDIS      YES START OVER
0556 e330 cc e0 b4   BPRET   JMP      GETCMD
0557
0558 e333 cd e6 42   ERROR  JSR      CLR TAB
0559 e336 a6 f1      LDA      #SF1
0560 e338 c7 02 0d   STA      DTABL+1
0561 e33b a6 60      LDA      #S60
0562 e33d c7 02 0e   STA      DTABL+2
0563 e340 c7 02 0f   STA      DTABL+3
0564 e343 cc e0 bc   JMP      DSCN
0565
0566

```



```

0567
0568
0569
0570
0571
0572
*****
*
*      Breakpoint clear.
*
*****
0573 e346 cd e6 42      BPCLR JSR CLRTAB          PRINT `BCLR`
0574 e349 a6 f4          LDA #SF4
0575 e34b c7 02 0c      STA DTABL
0576 e34e a6 d1          LDA #SD1
0577 e350 c7 02 0d      STA DTABL+1
0578 e353 a6 d0          LDA #SD0
0579 e355 c7 02 0e      STA DTABL+2
0580 e358 a6 60          LDA #S60
0581 e35a c7 02 0f      STA DTABL+3
0582 e35d cd e6 4c      JSR DISTAB
0583 e360 cd e6 1f      JSR SCNBKP          FIND B.P. TABLE
0584 e363 bf 21          STX WORK2
0585 e365 cd e7 90      JSR GETNYB
0586 e368 25 14          BCS BPCLR1          ENTER?
0587 e36a a1 11          CMP #S11
0588 e36c 26 36          BNE BPCRET          NO
0589 e36e a6 ff          LDA #SFF            YES,CLEAR ALL
0590 e370 be 21          LDX WORK2
0591 e372 d7 02 16      BPCLR2 STA BKPTBL,X
0592 e375 5c             INCX
0593 e376 5c             INCX
0594 e377 5c             INCX
0595 e378 3a 29          DEC PNCNT
0596 e37a 26 f6          BNE BPCLR2
0597 e37c 20 26          BRA BPCRET
0598 e37e a1 03          BPCLR1 CMP #3          VALID B.P. #?
0599 e380 24 b1          BHS ERROR           NO
0600 e382 97             TAX                 YES
0601 e383 d6 e7 6e      LDA CTABL,X         PRINT B.P. #
0602 e386 c7 02 11      STA DTABL+5
0603 e389 4f             CLRA
0604 e38a a0 03          SUB #3              FIND IT
0605 e38c ab 03          BPCLR3 ADD #3
0606 e38e 5a             DECX
0607 e38f 2a fb          BPL BPCLR3
0608 e391 b7 26          STA WORK6
0609 e393 cd e6 4c      JSR DISTAB          PRINT B.P.
0610 e396 cd e6 bb      JSR CHRIN
0611 e399 a1 11          CMP #S11           CLEAR IT?
0612 e39b 26 07          BNE BPCRET          NO
0613 e39d be 26          LDX WORK6          YES
0614 e39f a6 ff          LDA #SFF
0615 e3a1 d7 02 16      STA BKPTBL,X
0616 e3a4 cc e0 b4      BPCRET JMP GETCMD
0617
0618

```

```

0619
0620
0621
0622
0623
0624
0625 e3a7 12 20
0626 e3a9 cd e1 23
0627 e3ac e6 07
0628 e3ae b7 22
0629 e3b0 b7 27
0630 e3b2 e6 08
0631 e3b4 b7 23
0632 e3b6 b7 28
0633 e3b8 cd e7 46
0634 e3bb 44
0635 e3bc 44
0636 e3bd 44
0637 e3be 44
0638 e3bf 97
0639 e3c0 d6 e3 db
0640 e3c3 bb 23
0641 e3c5 b7 23
0642 e3c7 b6 22
0643 e3c9 a9 00
0644 e3cb b7 22
0645
0646 e3cd cd e7 46
0647 e3d0 c7 02 15
0648 e3d3 a6 83
0649 e3d5 cd e7 57
0650
0651 e3d8 cc e5 d3
0652
0653 e3db 03 02 02 01 01 MAT FCB 3,2,2,2,1,1,2,1,1,1,2,2,3,3,2,1
      02 01 01 01 02 02
      03 03 02 01
0654
0655

```

```

*****
*                                     *
*           Proceed.                   *
*                                     *
*****
PROC  BSET  1,STAT      SET PROCEED FLAG
      JSR   LOCSTK     FIND S.P.
      LDA   7,X
      STA   ADDRH
      STA   TEMP
      LDA   8,X
      STA   ADDRHL
      STA   TEMP+1
      JSR   LOAD       GET OPCODE
      LSRA
      LSRA
      LSRA
      LSRA
      TAX
      LDA   MAT,X      X <- MSB
      ADD   ADDRHL     NUMBER OF BYTES THIS INSTRUCTION
                        CALCULATE NEXT ADDRESS AND MOVE
                        CURRENT BREAKPOINT TO IT
      STA   ADDRHL
      ADC   #0
      STA   ADDRHL
      JSR   LOAD
      STA   PROP
      LDA   #$83
      JSR   STORE
      JMP   NCONT
      FCB  3,2,2,2,1,1,2,1,1,1,2,2,3,3,2,1

```

```

0656
0657
0658
0659
0660
0661
0662 e3eb 1a 20
0663 e3ed ae 06
0664 e3ef 20 03
0665 e3f1 1b 20
0666 e3f3 5f
0667 e3f4 a6 81
0668 e3f6 b7 24
0669 e3f8 cd e1 f2
0670
0671 e3fb ad 5c
0672 e3fd a1 53
0673 e3ff 26 fa
0674 e401 ad 56
0675 e403 a1 39
0676 e405 27 6c
0677 e407 a1 31
0678 e409 26 f0
0679
0680 e40b 3f 2a
0681 e40d 3f 2d
0682 e40f cd e4 8a
0683 e412 b7 2e
0684
0685 e414 cd e4 8a
0686 e417 b7 22
0687 e419 cd e4 8a
0688 e41c b7 23
0689
0690 e41e cd e4 8a
0691 e421 27 1e
0692 e423 0b 20 0b
0693 e426 b7 26
0694 e428 cd e5 4b
0695 e42b b1 26
0696 e42d 26 57
0697 e42f 20 08
0698 e431 cd e5 4b
0699 e434 c1 02 13
0700 e437 26 3d
0701 e439 3c 23
0702 e43b 26 02
0703 e43d 3c 22
0704 e43f 20 dd
0705
0706
0707
0708
0709
0710
0711
0712
0713 e441 bb 2a
0714 e443 b7 2a
0715 e445 a1 ff
0716 e447 27 b2
0717
0718 e449 ae 01
0719 e44b 9f
0720 e44c cd e7 d6
0721 e44f 4f
0722 e450 c7 02 10
0723 e453 cd e8 02
0724 e456 cc e0 bf
0725

```

```

*****
*
* RS232 (9600) S-Record receiver (E0 at 4MHz).
*
*****

```

```

VERIFY BSET 5,STAT SERIAL VERIFY
LDX #6
BRA L4
TLOAD BCLR 5,STAT SERIAL LOAD
CLR
L4 LDA #S81
STA WORK3
JSR DISP
INPUT BSR INCHD 7 BIT ASCII INTO A
CMP #'S' S ?
BNE INPUT NO, TRY AGAIN
BSR INCHD YES, GET NEXT CHARACTER
CMP #'9' 9 ?
BEQ NINE YES, FINISH
CMP #'1' NO, 1 ?
BNE INPUT NO, TRY AGAIN
LNKTH CLR CHKSUM YES, CLEAR CHECKSUM
CLR TMP2 AND TEMP. STORE
JSR BYTEI AND GET BYTE COUNT
STA BCNT AND SAVE IT
ADDR JSR BYTEI ADDRESS HIGH
STA ADDRH
JSR BYTEI ADDRESS LOW
STA ADDR
DLOP JSR BYTEI 75 GET A BYTE
BEQ CHCK 3 78 LAST BYTE ?
BRCLR 5,STAT,L5 5 83 NO, VERIFYING ?
STA WORK6 4 87 YES
JSR RAMACC 30 117 READ RAM
CMP WORK6 3 120 SAME ?
BNE ERR7 3 123
BRA L6 3 126
L5 JSR RAMACC 56 139 NO, WRITE TO RAM
CMP WORK4 3 142
BNE ERR2 3 145 READ-BACK OK ?
L6 INC ADDR 5 150 5 131 INCREMENT LS ADDRESS
BNE NOOVR 3 153 3 134 OVERFLOW ?
INC ADDR 5 158 5 139 YES, INC. HIGH BYTE
NOOVR BRA DLOP 3 161 3 142

```

```

*****
*
* Checksum byte & error routine.
*
*****

```

```

CHCK ADD CHKSUM
STA CHKSUM DEBUG
CMP #SFF IS CHECKSUM BYTE OK ?
BEQ INPUT YES, AND AGAIN
ERR LDX #1
TXA
JSR PRTDAT
CLRA
STA DTABL+4
JSR PRTADR
JMP CMDSCN

```

```

0726
0727
0728
0729
0730
0731
0732
0733 e459 ad 60
0734 e45b 05 01 fd
0735 e45e 04 01 fd
0736 e461 ae 07
0737 e463 bf 2b
0738 e465 ad 58
0739
0740 e467 ad 52
0741 e469 05 01 00
0742 e46c 46
0743 e46d 3a 2b
0744 e46f 26 f6
0745
0746 e471 44
0747 e472 81
0748
0749 e473 cc e0 b4
0750
0751 e476 ae 02
0752 e478 20 d1
0753 e47a ae 03
0754 e47c 20 cd
0755 e47e ae 04
0756 e480 20 c9
0757 e482 ae 05
0758 e484 20 c5
0759 e486 ae 07
0760 e488 20 c1
0761
0762

```

```

*****
*
*      Input routine, MC68HC05E0 : 0.5 uS.      *
*      Cycles per bit at 9600 baud : 208.      *
*
*****

```

```

INCHD  BSR      DEL191      191      GET OUT OF BIT 7
INCH   BRCLR   2,PORTB,*    5        IS LINE HIGH ?
        BRSET   2,PORTB,*    5        YES, WAIT FOR START
        LDX     #7           2      6    7 DATA BITS TO READ
        STX     COUNT        4      10  WAIT TILL MIDDLE OF 1st BIT
        BSR     DEL110      110     120  +/-3
INBT   BSR      DEL191      191      +120-208=103
        BRCLR   2,PORTB,ZER  5      196  CYC 2 (105) READ
ZER    RORA     COUNT        3      199  SAVE BIT
        DEC     COUNT        5      204
        BNE     INBT        3      207
        LSRA     COUNT        3      16   MSB A ZERO
        RTS     COUNT        6      22
NINE   JMP      GETCMD
ERR2   LDX     #2           RAM READBACK
        BRA     ERR
ERR3   LDX     #3           LESS THAN ASCII 0
        BRA     ERR
ERR4   LDX     #4           BETWEEN ASCII 9 & A
        BRA     ERR
ERR5   LDX     #5           MORE THAN ASCII F
        BRA     ERR
ERR7   LDX     #7           VERIFY ERROR
        BRA     ERR

```

```

0763
0764
0765
0766
0767
0768
0769 e48a ad cd          BYTEI  BSR    INCHD    22      MS NIBBLE
0770 e48c ad 17          BSR    ASCII    35 57  WHAT WAS IT
0771 e48e 48            LSLA                   3      OK
0772 e48f 48            LSLA                   3      SHIFT
0773 e490 48            LSLA                   3      IT
0774 e491 48            LSLA                   3 69  UP
0775 e492 b7 2c          STA    TMP1     4 73  AND SAVE IT
0776 e494 b6 2d          LDA    TMP2     3 76  RESTORE BYTE
0777 e496 bb 2a          ADD    CHRSUM   3 79  ACCUMULATE
0778 e498 b7 2a          STA    CHRSUM   4 83  IN CHECKSUM BYTE
0779 e49a ad bd          BSR    INCHD    22      LS NIBBLE
0780 e49c ad 07          BSR    ASCII    35 57  WHAT WAS IT
0781 e49e bb 2c          ADD    TMP1     3 60  ADD TO MS NIBBLE
0782 e4a0 b7 2d          STA    TMP2     4 64  SAVE BYTE
0783 e4a2 3a 2e          DEC    BCNT     5 69  DECREMENT BYTE COUNT
0784 e4a4 81            RTS                   6 75
0785
0786 e4a5 a1 30          ASCII  CMP    #$30     2      BEFORE ZERO ?
0787 e4a7 25 d1          BLO   ERR3     3 5    YES, NOT LEGAL
0788 e4a9 a1 39          CMP    #$39     2 7    AFTER NINE
0789 e4ab 22 03          BHI   MT9      3 10   YES TRY A-F
0790 e4ad a0 30          SUB   #$30     3 13   0-9, CONVERT TO HEX
0791 e4af 81            RTS                   6 19
0792
0793 e4b0 a1 41          MT9   CMP    #$41     2 12  BEFORE A ?
0794 e4b2 25 ca          BLO   ERR4     3 15   YES, NOT LEGAL
0795 e4b4 a1 46          CMP    #$46     2 17  AFTER F ?
0796 e4b6 22 ca          BHI   ERR5     3 20   YES, NOT LEGAL
0797 e4b8 a0 37          SUB   #$37     3 23  A-F, CONVERT TO HEX
0798 e4ba 81            NFND  RTS                   6 29
0799
0800 e4bb ae 1d          DEL191 LDX   #29      2
0801 e4bd 20 02          BRA   DELAY    3 5
0802 e4bf ae 10          DEL110 LDX   #16      2
0803 e4c1 5a          DELAY DECX     3
0804 e4c2 26 fd          BNE   DELAY    3 6 x X
0805 e4c4 81            RTS                   6 12+6X (INC BSR)
0806
0807

```

```

0808
0809
0810
0811
0812
0813
*****
*
*          RS232 (9600 @ 4MHz) S-Record output.
*
*****
0814 e4c5 14 06      PUNCH  BSET   2,PORTBD      BIT 2 OUTPUT
0815 e4c7 cd e1 b3   JSR    BLDRNG      BUILD RANGE
0816 e4ca 08 20 a6   BRSET  4,STAT,NINE  NEW ADDRESS ENTERED ?
0817 e4cd be 27      LDY    TEMP        NO, SWAP ADDRESSES
0818 e4cf b7 27      STA   TEMP
0819 e4d1 bf 22      STX   ADDRH
0820 e4d3 b6 23      LDA   ADDRDL
0821 e4d5 be 28      LDY   TEMP+1
0822 e4d7 bf 23      STX   ADDRDL
0823 e4d9 b7 28      STA   TEMP+1
0824 e4db 1f 20      BCLR  7,STAT        CLEAR END FLAG
0825
0826 e4dd b6 28      LOOP1  LDA   TEMP+1      END LSB
0827 e4df b0 23      SUB   ADDRDL          CURRENT LSB
0828 e4e1 b7 26      STA   WORK6          DIFFERENCE LSB
0829 e4e3 b6 27      LDA   TEMP           END MSB
0830 e4e5 b2 22      SBC   ADDRH          CURRENT MSB
0831 e4e7 26 od      BNE   LOTS           MSB ZERO ?
0832 e4e9 b6 26      LDA   WORK6          YES, LOOK AT LSB
0833 e4eb 4c          INCA                ADJUST
0834 e4ec 27 08      BEQ   LOTS           WAS $FF ?
0835 e4ee a1 20      CMP   #$20           MORE THAN 23 ?
0836 e4f0 22 04      BHI   LOTS           IF SO USE 23
0837 e4f2 1e 20      BSET  7,STAT        NO, LAST S1 RECORD
0838 e4f4 20 02      BRA   LTE20          LESS THAN OR EQUAL TO 20
0839
0840 e4f6 a6 20      LOTS   LDA   #$20
0841 e4f8 ab 03      LTE20  ADD   #$03
0842 e4fa b7 2e      STA   BCNT          ADD BYTE COUNT & ADDRESS
0843 e4fc a6 53      LDA   #'S          NO. BYTES THIS S1 RECORD
0844 e4fe cd e5 63   JSR   OUCH          S
0845 e501 a6 31      LDA   #'1          1
0846 e503 ad 5e      BSR   OUCH
0847 e505 3f 2a      CLR   CHKSUM
0848 e507 b6 2e      LDA   BCNT          BYTE COUNT
0849 e509 cd e5 8f   JSR   BYTED
0850 e50c b6 22      LDA   ADDRH          ADDRESS HIGH
0851 e50e cd e5 8f   JSR   BYTED
0852 e511 b6 23      LDA   ADDRDL
0853 e513 ad 7a      BSR   BYTED          ADDRESS LOW
0854
0855 e515 cd e7 46   LOOP2  JSR   LOAD        GET BYTE
0856 e518 3c 23      INC   ADDRDL        INCREMENT ADDRESS
0857 e51a 26 02      BNE   NOVR          OVERFLOW ?
0858 e51c 3c 22      INC   ADDRH          YES, INC. HIGH BYTE
0859 e51e ad 6f      BSR   BYTED        SEND BYTE
0860 e520 26 f3      NOVR   BNE   LOOP2   LAST BYTE ?
0861
0862
0863
0864
0865
0866
0867
0868
*****
*
*          Checksum byte.
*
*****
0869 e522 b6 2a      LDA   CHKSUM        CHECKSUM
0870 e524 43          COMA                REQUIRED CHECKSUM BYTE
0871 e525 ad 68      BSR   BYTED        SEND IT
0872 e527 ad 34      BSR   CRLF          CRLF
0873 e529 0f 20 b1   BRCLR 7,STAT,LOOP1  FINISHED ?
0874

```

```

0875
0876
0877
0878
0879
0880
0881 e52c a6 53
0882 e52e ad 33
0883 e530 a6 39
0884 e532 ad 2f
0885 e534 a6 03
0886 e536 ad 57
0887 e538 a6 00
0888 e53a ad 53
0889 e53c a6 00
0890 e53e ad 4f
0891 e540 a6 fc
0892 e542 ad 4b
0893 e544 ad 17
0894 e546 15 06
0895 e548 cc e0 b4
0896
0897
0898
0899
0900
0901
0902
0903 e54b 0a 20 09
0904 e54e ae c7
0905 e550 bf 21
0906 e552 bd 21
0907 e554 c7 02 13
0908 e557 ae c6
0909 e559 bf 21
0910 e55b bc 21
0911
0912
0913 e55d a6 0d
0914 e55f ad 02
0915 e561 a6 0a
0916
0917
0918
0919
0920
0921
0922
0923
0924 e563 14 01
0925 e565 ae 0a
0926 e567 bf 2b
0927 e569 cd e4 c1
0928 e56c 98
0929 e56d 20 08
0930
0931 e56f ae 1c
0932 e571 cd e4 c1
0933 e574 9d
0934 e575 99
0935 e576 46
0936 e577 25 04
0937 e579 15 01
0938 e57b 20 04
0939 e57d 14 01
0940 e57f 20 00
0941 e581 3a 2b
0942 e583 26 ea
0943 e585 81
0944
0945 e586 ab 30
0946 e588 a1 39
0947 e58a 23 02
0948 e58c ab 07
0949 e58e 81
0950

```

```

*****
*
*           S9 record.
*
*****

```

```

LDA #S' S
BSR OUCH
LDA #'9' 9
BSR OUCH
LDA #S03 3 BYTES
BSR BYTED
LDA #S00
BSR BYTED DUMMY (0)
LDA #S00
BSR BYTED ADDRESS
LDA #SFC
BSR BYTED CHECKSUM
BSR CRLF
BCLR 2,PORTBD BIT 2 INPUT
JMP GETCMD

```

```

*****
*
*           RAM R/W for S-record load.
*
*****

```

```

RAMACC BRSET 5,STAT,L3 5 READING ?
LDX #SC7 2 7 NO, WRITING
STX WORK2 4 11
JSR WORK2 16 27
STA WORK4 4 31
L3 LDX #SC6 2 33 READING
STX WORK2 4 37
JMP WORK2 13 50 (56 WITH JSR)

CRLF LDA #S0D CR
BSP OUCH
LDA #S0A LF

```

```

*****
*
*           Output routine, 208 cycles per bit.
*
*****

```

```

OUCH BSET 2,PORTB 5 MAKE SURE IT'S HIGH
LDX #10 2 7 10 BITS TO SEND
STX COUNT 4 11 START, 8 DATA, STOP
JSR DELAY 72 83
CLC 2 85 START A ZERO
BRA STAR 3 88

OUTBT LDX #28 2
JSR DELAY 180 182
NOP 2 184
DEL3 SEC 2 186 FILL WITH ONES FOR STOP
RORA 3 189 GET A BIT
STAR BCS OU1 3 192 1 ?
BCLR 2,PORTB 5 197 NO
BRA OBD 3 200
OU1 BSET 2,PORTB 5 YES
BRA OBD 3
OBD DEC COUNT 5 205
BNE OUTBT 3 208 DONE ?
RTS

ASCIO ADD #S30 3 CONVERT TO ASCII
CMP #S39 2 5 0-9 ?
BLS NMT9 3 8
ADD #S07 3 8 NO, A-F
NMT9 RTS 6 14

```

```

0951
0952
0953
0954
0955
0956
0957 e58f b7 2c
0958 e591 44
0959 e592 44
0960 e593 44
0961 e594 44
0962 e595 ad ef
0963 e597 ad ca
0964 e599 b6 2c
0965 e59b bb 2a
0966 e59d b7 2a
0967 e59f b6 2c
0968 e5a1 a4 0f
0969 e5a3 ad e1
0970 e5a5 ad bc
0971 e5a7 3a 2e
0972 e5a9 81
0973
0974

```

*	Byte output sub-routine.			*
*				*

BYTEO	STA	TMP1		
	LSRA			SHIFT
	LSRA			DOWN TO
	LSRA			GET MSB
	LSRA			
	BSR	ASCIO		& CONVERT
	BSR	OUCH		
	LDA	TMP1		RESTORE BYTE
	ADD	CHKSUM		ACCUMULATE
	STA	CHKSUM		IN CHECKSUM BYTE
	LDA	TMP1		
	AND	#50F		LSB
	BSR	ASCIO		CONVERT IT
	BSR	OUCH		
	DEC	BCNT		DECREMENT BYTE COUNT
	RTS			


```

0975
0976
0977
0978
0979
0980
0981 e5aa 13 20          NEWGO  BCLR   1,STAT
0982 e5ac cd e1 23      JSR    LOCSTK
0983 e5af e6 08         LDA    8,X
0984 e5b1 b7 23         STA   ADDR1
0985 e5b3 e6 07         LDA    7,X
0986 e5b5 b7 22         STA   ADDRH
0987 e5b7 cd e7 9a      JSR    GETADR
0988 e5ba 25 08         BCS   GOON          ADDR VALID?
0989 e5bc a1 10         CMP   #S10
0990 e5be 27 5c         BEQ   GOBACK
0991 e5c0 a1 11         CMP   #S11
0992 e5c2 26 55         BNE   GOERR
0993 e5c4 cd e1 23      GOON   JSR    LOCSTK          YES PUT IT
0994 e5c7 b6 22         LDA   ADDRH          IN STACK
0995 e5c9 e7 07         STA   7,X
0996 e5cb b7 27         STA   TEMP
0997 e5cd b6 23         LDA   ADDR1
0998 e5cf e7 08         STA   8,X
0999 e5d1 b7 28         STA   TEMP+1
1000
1001 e5d3 ad 4a          NCONT  BSR    SCNBKP          FIND B.P. TABLE
1002 e5d5 d6 02 16      GOINSB LDA   BKPTBL,X          GET ADDRESS MSB
1003 e5d8 2b 0d         BMI   GONOB          VALID ? (<32k ONLY)
1004 e5da b7 22         STA   ADDRH          YES
1005 e5dc d6 02 17      LDA   BKPTBL+1,X      LSB
1006 e5df b7 23         STA   ADDR1
1007 e5e1 cd e7 46      JSR    LOAD          READ INSTRUCTION
1008 e5e4 d7 02 18      STA   BKPTBL+2,X
1009 e5e7 5c           GONOB  INCX          GET NEXT B.P.
1010 e5e8 5c           INCX
1011 e5e9 5c           INCX
1012 e5ea 3a 29         DEC   PNCNT
1013 e5ec 26 e7         BNE   GOINSB          DONE?
1014
1015 e5ee ad 2f           BSR    SCNBKP          FIND B.P. TABLE
1016 e5f0 d6 02 16      GOINSB2 LDA  BKPTBL,X          INSERT B.P.'S
1017 e5f3 2b 17         BMI   GONOB2         VALID MSB ?
1018 e5f5 b7 22         STA   ADDRH          YES
1019 e5f7 b1 27         CMP   TEMP          SAME AS MSB OF NEXT ADDRESS ?
1020 e5f9 26 07         BNE   DOSW          IF NOT THEN INSERT SWI
1021 e5fb d6 02 17      LDA   BKPTBL+1,X      SAME, GET LSB
1022 e5fe b1 28         CMP   TEMP+1         SAME AS THAT OF NEXT ADDRESS ?
1023 e600 27 0a         BEQ   GONOB2         IF SO THEN NO SWI
1024 e602 d6 02 17      DOSW  LDA   BKPTBL+1,X      SAME, GET LSB
1025 e605 b7 23         STA   ADDR1
1026 e607 a5 83         LDA   #S83          INSERT SWI
1027 e609 cd e7 57      JSR    STORE
1028 e60c 5c           GONOB2 INCX          GET NEXT B.P.
1029 e60d 5c           INCX
1030 e60e 5c           INCX
1031 e60f 3a 29         DEC   PNCNT
1032 e611 26 dd         BNE   GOINSB2         DONE?
1033
1034 e613 c5 02 14         LDA   TMPTCR          RESTORE ORIGINAL
1035 e616 b7 0c         STA   TCR            TCR VALUE
1036 e618 80           RTI                  YES
1037
1038
1039 e619 cc e3 33      GOERR  JMP    ERROR
1040 e61c cc e0 b4      GOBACK JMP    GETCMD
1041
1042 e61f a6 03          SCNBKP LDA   #NBKPT
1043 e621 b7 29         STA   PNCNT
1044 e623 5f           CLRX
1045 e624 81           RTS
1046

```

```

1047
1048
1049
1050
1051
1052
1053 e625 a6 b5
1054 e627 c7 02 10
1055 e62a a6 73
1056 e62c c7 02 11
1057 e62f 4f
1058 e630 5f
1059 e631 cd e7 d8
1060 e634 cd e1 23
1061 e637 9f
1062 e638 ab 03
1063 e63a ae 02
1064 e63c cd e7 d8
1065 e63f cc e0 bf
1066
1067
1068
1069
1070
1071
1072
1073 e642 ae 05
1074 e644 4f
1075 e645 d7 02 0c
1076 e648 5a
1077 e649 2a f9
1078 e64b 81
1079
1080
1081
1082
1083
1084
1085
1086 e64c ae 05
1087 e64e d6 02 0c
1088 e651 ad 08
1089 e653 5a
1090 e654 2a f8
1091 e656 81
1092
1093
1094
1095
1096
1097
1098
1099
1100 e657 ad e9
1101 e659 20 f1
1102
1103
1104
1105
1106
1107
1108
1109 e65b cf 02 12
1110 e65e 1d 00
1111 e660 ae 08
1112 e662 48
1113 e663 24 02
1114 e665 1c 00
1115 e667 1e 00
1116 e669 1f 00
1117 e66b 1d 00
1118 e66d 5a
1119 e66e 26 f2
1120 e670 ce 02 12
1121 e673 81
1122

```

```

*****
*
*   Display stack pointer.
*
*****
STACK  LDA   #5B5           PRINT
        STA   DTABL+4      'SP'
        LDA   #573
        STA   DTABL+5
        CLRA
        CLRX
        JSR   PRTBYT
        JSR   LOCSTK       FIND USER
        TXA                   STACK POINTER
        ADD   #3
        LDX   #2
        JSR   PRTBYT       PRINT IT
        JMP   CMDSCN
*****
*
*   Clear display table.
*
*****
CLRRTAB LDX   #5
CLRLOC  CLRA
        STA   DTABL,X      CLEAR SIX
        DECX                   LOCATIONS IN
        BPL  CLRLOC        DISPLAY TABLE
        RTS
*****
*
*   Display table contenst.
*
*****
DISTAB  LDX   #5
DISCHR  LDA   DTABL,X      LOAD DISPLAY
        BSR   DISPLY       TABLE INTO
        DECX                   145000
        BPL  DISCHR
        RTS
*****
*
*   Blank display.
*
*****
CLRDIS  BSR   CLRRTAB      BLANK
        BRA   DISTAB       DISPLAY
*****
*
*   Shift one character into display.
*
*****
DISPLY  STX   WORK1        SAVE INDEX
        BCLR  6,PORTA      CLEAR DATA
        LDX   #8
DIS1    LSLA
        BCC  DIS2         SET UP
        BSET 6,PORTA      BIT OF
        BSET 7,PORTA      ACCUMULATOR
        BCLR 7,PORTA      CLOCK
        BCLR 6,PORTA      IT
        BCLR 6,PORTA      CLEAR DATA
        DECX                   COMPLETE?
        BNE  DIS1         NO
        LDX  WORK1        RESTORE INDEX
        RTS
*****

```

```

1123
1124
1125
1126
1127
1128
1129 e674 98
1130 e675 4f
1131 e676 ae 06
1132 e678 ab 10
1133 e67a b7 00
1134 e67c ad 06
1135 e67e 25 03
1136 e680 5a
1137 e681 26 f5
1138 e683 81
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148 e684 b6 00
1149 e686 c7 02 12
1150 e689 a5 0f
1151 e68b 27 1b
1152 e68d ad 1a
1153 e68f b6 00
1154 e691 c1 02 12
1155 e694 26 12
1156 e696 99
1157 e697 b6 00
1158 e699 a5 0f
1159 e69b 26 fa
1160 e69d ad 0a
1161 e69f b6 00
1162 e6a1 a5 0f
1163 e6a3 26 f2
1164 e6a5 c6 02 12
1165 e6a8 81
1166
1167
1168
1169
1170
1171
1172
1173 e6a9 a6 0a
1174 e6ab b7 25
1175 e6ad a6 ff
1176 e6af 21 fe
1177 e6b1 21 fe
1178 e6b3 4a
1179 e6b4 26 f9
1180 e6b6 3a 25
1181 e6b8 26 f3
1182 e6ba 81
1183
1184
1185
1186
1187
1188
1189
1190
1191 e6bb cd e6 74
1192 e6be 24 fb
1193 e6c0 5f
1194 e6c1 d1 e6 cb
1195 e6c4 27 03
1196 e6c6 5c
1197 e6c7 20 f8
1198 e6c9 9f
1199 e6ca 81
1200

```

```

*****
*                                     *
*      Keypad scan, carry set if valid output.      *
*                                     *
*****
KEYSCN  CLC
        CLRA
        LDX  #6
KEY1    ADD  #S10      SETUP
        STA  PORTA    ROW
        BSR  COLUMN   CHECK COLUMNS
        BCS  KEY2     IF VALID GET OUT
        DECX          ELSE TRY
        BNE  KEY1     NEXT ROW
KEY2    RTS

```

```

*****
*                                     *
*      Check for key closure.      *
*      A contains value, C set if valid output.      *
*                                     *
*****
COLUMN  LDA  PORTA    READ KEYPAD
        STA  WORK1   STORE IT
        BIT  #S0F    KEY CLOSED?
        BEQ  COLRET  NO GET OUT
        BSR  DBOUNC  ELSE DEBOUNCE
        LDA  PORTA   RE-READ KEYPAD
        CMP  WORK1   SAME KEY CLOSED?
        BNE  COLRET  NO GET OUT
        SEC
        COL1  LDA  PORTA    KEY
        BIT  #S0F    RELEASED?
        BNE  COL1   NO TRY AGAIN
        BSR  DBOUNC  YES DEBOUNCE
        LDA  PORTA   STILL
        BIT  #S0F    RELEASED?
        BNE  COL1   NO TRY AGAIN
        LDA  WORK1   RETURN CHAR IN A-REG
COLRET  RTS          YES GO HOME

```

```

*****
*                                     *
*      Pause for 3075 cycles.      *
*                                     *
*****
DBOUNC  LDA  #10      40mS
        STA  WORK5
DLP     LDA  #SFF     PAUSE
DLOOP  BRN  *         256X12
        BRN  *         CYCLES
        DECA          OR AT
        BNE  DLOOP   LEAST 3.7mS
        DEC  WORK5
        BNE  DLP
        RTS

```

```

*****
*                                     *
*      Input one character, A contains value.      *
*                                     *
*****
CHRIN   JSR  KEYSCN   GET KEY
        BCC  CHRIN   IF NOT VALID RETRY
        CLRX
CHRIN1  CMP  STABL,X  CONVERT
        BEQ  CHRIN2  TO HEX
        INCX
        BRA  CHRIN1
CHRIN2  TXA
        RTS          IF CANCEL

```

1201
 1202
 1203
 1204
 1205
 1206
 1207 e6cb 11
 1208 e6cc 21
 1209 e6cd 22
 1210 e6ce 24
 1211 e6cf 31
 1212 e6d0 32
 1213 e6d1 34
 1214 e6d2 41
 1215 e6d3 42
 1216 e6d4 44
 1217 e6d5 48
 1218 e6d6 38
 1219 e6d7 28
 1220 e6d8 18
 1221 e6d9 14
 1222 e6da 12
 1223 e6db 61
 1224 e6dc 58
 1225 e6dd 68
 1226 e6de 64
 1227 e6df 62
 1228 e6e0 54
 1229 e6e1 52
 1230 e6e2 51
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239 e6e3 cd e7 9a
 1240 e6e4 a1 10
 1241 e6e8 27 57
 1242 e6ea c7 02 12
 1243 e6ed cd e7 46
 1244 e6f0 cd e7 d6
 1245 e6f3 cd e7 90
 1246 e6f6 a1 10
 1247 e6f8 27 47
 1248 e6fa a1 11
 1249 e6fc 27 1b
 1250 e6fe a1 13
 1251 e700 27 2f
 1252 e702 a1 0f
 1253 e704 22 08
 1254 e706 cd e7 d6
 1255 e709 cd e7 7e
 1256 e70c 25 f8
 1257
 1258 e70e a1 11
 1259 e710 26 15
 1260 e712 b6 21
 1261 e714 cd e7 57
 1262 e717 25 d1
 1263 e719 0c 20 25
 1264 e71c 3c 23
 1265 e71e 26 02
 1266 e720 3c 22
 1267 e722 cd e8 02
 1268 e725 20 c3
 1269 e727 a1 13
 1270 e729 26 16
 1271 e72b b6 21
 1272 e72d ad 28
 1273 e72f 25 b9
 1274 e731 0c 20 0d
 1275 e734 3d 23
 1276 e736 26 02
 1277 e738 3a 22
 1278 e73a 3a 23
 1279 e73c cd e8 02
 1280 e73f 20 a9
 1281 e741 1d 20
 1282 e743 cc e0 b4
 1283
 1284

```

.....
*
*           Conversion table for keypad.
*
.....

```

STABL	FCB	\$11	0	
	FCB	\$21	1	
	FCB	\$22	2	
	FCB	\$24	3	
	FCB	\$31	4	
	FCB	\$32	5	
	FCB	\$34	6	
	FCB	\$41	7	
	FCB	\$42	8	
	FCB	\$44	9	
	FCB	\$48	A	
	FCB	\$38	B	
	FCB	\$28	C	
	FCB	\$18	D	
	FCB	\$14	E	
	FCB	\$12	F	
	FCB	\$61	10	CANCEL COMMAND
	FCB	\$58	11	ENTER COMMAND
	FCB	\$68	12	STACK POINTER
	FCB	\$64	13	MEMORY
	FCB	\$62	14	GO
	FCB	\$54	15	VERIFY TAPE
	FCB	\$52	16	LOAD TAPE
	FCB	\$51	17	PUNCH TAPE

```

.....
*
*           Memory location examine/change.
*
.....

```

MEMEX	JSR	GETADR	BUILD ADDRESS	
	CMP	#S10		
	BEQ	MEMEX4		
MEMEX3	STA	WORK1		
	JSR	LOAD	LOAD DATA	
	JSR	PRTDAT	PRINT IT	
	JSR	GETNYB	GET NEW NIBBLE	
	CMP	#S10		
	BEQ	MEMEX4		
	CMP	#S11		
	BEQ	ADRINC		
	CMP	#S13		
	BEQ	ADRDEC		
	CMP	#S0F		
	BHI	CMDMDL	IF VALID	
MEMEX1	JSR	PRTDAT	PRINT IT	
	JSR	GETBY2	SHIFT IN NEXT	
	BCS	MEMEX1	IF VALID TRY AGAIN	
	CMDMDL	CMP	#S11	ENTER?
	BNE	MEMEX2	NO	
	LDA	WORK2	RESTORE ACCA	
	JSR	STORE	YES STORE IT	
	BCS	MEMEX3	STORE VALID?	
ADRINC	BRSET	6,STAT,MEMEX4		
	INC	ADDRL	YES GOTO	
	BNE	MEMEX5	NEXT	
	INC	ADDRH		
MEMEX5	JSR	PRTADR	PRINT IT	
	BRA	MEMEX3	REPEAT	
MEMEX2	CMP	#S13		
	BNE	MEMEX4	NO	
	LDA	WORK2		
	BSR	STORE		
	BCS	MEMEX3		
ADRDEC	BRSET	6,STAT,MEMEX4		
	TST	ADDRL	YES THEN	
	BNE	CMDMB2	GET PREVIOUS	
	DEC	ADDRH	ADDRESS	
CMDMB2	DEC	ADDRL		
	JSR	PRTADR	PRINT IT	
	BRA	MEMEX3	REPEAT	
MEMEX4	BCLR	6,STAT	INVALID CHAR	
	JMP	GETCMD		

```

1285
1286
1287
1288
1289
1290
1291 e746 cf 02 12
1292 e749 ae c6
1293 e74b bf 21
1294 e74d ae 81
1295 e74f bf 24
1296 e751 bd 21
1297 e753 ce 02 12
1298 e756 81
1299
1300
1301
1302
1303
1304
1305
1306 e757 cf 02 12
1307 e75a ae c7
1308 e75c ad ed
1309 e75e c7 02 13
1310 e761 cd e7 46
1311 e764 c1 02 13
1312 e767 27 01
1313 e769 99
1314 e76a ce 02 12
1315 e76d 81
1316
1317
1318
1319
1320
1321
1322
1323 e76e d7
1324 e76f 06
1325 e770 e3
1326 e771 a7
1327 e772 36
1328 e773 b5
1329 e774 f5
1330 e775 07
1331 e776 f7
1332 e777 b7
1333 e778 77
1334 e779 f4
1335 e77a d1
1336 e77b e6
1337 e77c f1
1338 e77d 71
1339
1340
1341
1342
1343
1344
1345
1346
1347 e77e b7 21
1348 e780 ad 0e
1349 e782 24 0b
1350 e784 38 21
1351 e786 38 21
1352 e788 38 21
1353 e78a 38 21
1354 e78c ba 21
1355 e78e 99
1356 e78f 81
1357

```

```

*****
*
*      Load byte at ADDRH,ADDRL into A.
*
*****

```

```

LOAD   STX   WORK1      SETUP
LDSTCM LDX   #SC6      ROUTINE
        LDX   WORK2     TO DO
        LDX   #S81     TWO BYTE
        STX   WORK3     LOAD
        JSR   WORK2
        LDX   WORK1
        RTS

```

```

*****
*
*      Store byte in A at ADDRH,ADDRL.
*
*****

```

```

STORE  STX   WORK1      SETUP
        LDX   #SC7      ROUTINE
LDSTCM BSR   LDSTCM     TO DO
        STA   WORK4     TWO BYTE
        JSR   LOAD
        CMP  WORK4     STORE
        BEQ  STRTS
        SEC
STRTS  LDX   WORK1
        RTS

```

```

*****
*
*      Hex. to mux. display conversion.
*
*****

```

```

CTABL  FCB   $D7      0
        FCB   $06      1
        FCB   $E3      2
        FCB   $A7      3
        FCB   $36      4
        FCB   $B5      5
        FCB   $F5      6
        FCB   $07      7
        FCB   $F7      8
        FCB   $B7      9
        FCB   $77      A
        FCB   $F4      B
        FCB   $D1      C
        FCB   $E6      D
        FCB   $F1      E
        FCB   $71      F

```

```

*****
*
*      Build a byte in accumulator.
*
*****

```

```

GETBY2 STA   WORK2
        BSR  GETNYB
        BCC  GETBRT
        ASL  WORK2
        ASL  WORK2
        ASL  WORK2
        ASL  WORK2
        ORA  WORK2
        SEC
GETBRT RTS

```

```

1358
1359
1360
1361
1362
1363
1364
1365
1366 e790 cd e6 bb
1367 e793 98
1368 e794 a1 0f
1369 e796 22 01
1370 e798 99
1371 e799 81
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381 e79a cd e6 42
1382 e79d cd e8 02
1383 e7a0 ad ee
1384 e7a2 25 0a
1385 e7a4 a1 11
1386 e7a6 27 2d
1387 e7a8 a1 10
1388 e7aa 27 29
1389 e7ac 20 ec
1390 e7ae 3f 22
1391 e7b0 b7 23
1392 e7b2 cd e8 02
1393 e7b5 ad d9
1394 e7b7 24 13
1395 e7b9 48
1396 e7ba 48
1397 e7bb 48
1398 e7bc 48
1399 e7bd ae 04
1400 e7bf 48
1401 e7c0 39 23
1402 e7c2 39 22
1403 e7c4 5a
1404 e7c5 26 f8
1405 e7c7 cd e8 02
1406 e7ca 20 e9
1407 e7cc a1 10
1408 e7ce 27 05
1409 e7d0 a1 11
1410 e7d2 26 e1
1411 e7d4 99
1412 e7d5 81
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423 e7d6 ae 04
1424 e7d8 cf 02 12
1425 e7db c7 02 13
1426 e7de 44
1427 e7df 44
1428 e7e0 44
1429 e7e1 44
1430 e7e2 97
1431 e7e3 d6 e7 6e
1432 e7e6 ce 02 12
1433 e7e9 d7 02 0c
1434 e7ec c6 02 13
1435 e7ef a4 0f
1436 e7f1 97
1437 e7f2 d6 e7 6e
1438
1439 e7f5 ce 02 12
1440 e7f8 d7 02 0d
1441 e7fb cd e6 4c
1442 e7fe c6 02 13
1443 e801 81

*****
*
*   Get one character and check for valid   *
*   hex, A contains output, carry set if   *
*   valid.                                  *
*
*****
GETNYB JSR   CHRIN           GET CHARACTER
        CLC
        CMP   #SOF         VALID HEX?
        BHI  GETRET       NO
        SEC           YES
GETRET RTS

*****
*
*   Build address in ADDRH,ADDRL, carry set *
*   if new address.                         *
*
*****
GETADR JSR   CLR TAB       BLANK DISPLAY
BLDA2 JSR   PRTADR
BLDADR BSR   GETNYB       GET CHARACTER
        BCS  GETAD1       VALID HEX?
        CMP  #$11         ENTER ?
        BEQ  GETRTS
        CMP  #$10         NO, CANCEL ?
        BEQ  GETRTS       NO, TRY AGAIN
        BRA  GETADR
GETAD1 CLR  ADDR H        INIT HIGH ADDRESS
        STA  ADDR L        PUT CHAR AWAY
        JSR  PRTADR       PRINT NEW ADDRESS
GETALP BSR   GETNYB       GET ANOTHER CHAR
        BCC  GETARG       VALID?
        ASLA
        ASLA              YES
        ASLA              SHIFT IT IN
        ASLA
        LDX  #4
GETASF ASLA
        ROL  ADDR L
        ROL  ADDR H
        DECX
        BNE  GETASF
        JSR  PRTADR       PRINT NEW ADDR
        BRA  GETALP       GET ANOTHER CHAR
GETARG CMP  #$10         NOT VALID HEX, CANCEL ?
        BEQ  GETRTS
        CMP  #$11         ENTER ?
        BNE  GETALP       NO TRY AGAIN
        SEC           YES SET FLAG
GETRTS RTS

*****
*
*   Print one byte into pair of display     *
*   digits, A contains byte, X points to   *
*   first diget.                          *
*
*****
PRTDAT LDX  #4           PRINT IN LAST TWO DIGITS
PRTBYT STX  WORK1
        STA  WORK4
        LSRA
        LSRA
        LSRA
        LSRA
        TAX
        LDA  CTABL, X
        LDX  WORK1
        STA  DTABL, X
        LDA  WORK4
        AND  #SOF
        TAX
        LDA  CTABL, X
        LDX  WORK1
        STA  DTABL+1, X
        JSR  DISTAB
        LDA  WORK4
        RTS

```

```

1444
1445
1446
1447
1448
1449
1450
1451 e802 b7 25
1452 e804 bf 24
1453 e806 b6 22
1454 e808 5f
1455 e809 ad cd
1456 e80b b6 23
1457 e80d ae 02
1458 e80f ad c7
1459 e811 b6 25
1460 e813 be 24
1461 e815 81
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473 e816 a6 81
1474 e818 b7 24
1475 e81a a6 f4
1476 e81c c7 02 10
1477 e81f a6 77
1478 e821 c7 02 11
1479 e824 cd e6 4c
1480 e827 a6 04
1481 e829 b7 22
1482 e82b a6 00
1483 e82d b7 23
1484 e82f cd e7 9d
1485 e832 25 04
1486 e834 a1 11
1487 e836 26 61
1488 e838 b6 22
1489 e83a b7 27
1490 e83c b6 23
1491 e83e b7 28
1492 e840 a6 f1
1493 e842 c7 02 10
1494 e845 a6 77
1495 e847 c7 02 11
1496 e84a cd e6 4c
1497 e84d a6 f1
1498 e84f b7 22
1499 e851 a6 ff
1500 e853 b7 23
1501 e855 cd e7 9d
1502 e858 25 04
1503 e85a a1 11
1504 e85c 26 3b
1505 e85e b6 22
1506 e860 be 27
1507 e862 b7 27
1508 e864 bf 22
1509 e866 b6 23
1510 e868 be 28
1511 e86a bf 23
1512 e86c b7 28
1513
1514

```

```

*****
*
*      Print address ADDRH,ADDRL.
*
*****
PRTADR  STA      WORK5
        STX      WORK3
        LDA      ADDRH
        CLRX
        BSR      PRTBYT
        LDA      ADDRL
        LDX      #2
        BSR      PRTBYT
        LDA      WORK5
        LDX      WORK3
        RTS

```

```

*****
*
*      Transfer code from EPROM to RAM, default
*      destination address:- $0400-$1FFF, from
*      EPROM at $4400 - $5FFF.
*      (TEMP,TEMP+1 -> ADDRH,ADDRL).
*
*****

```

```

XFER    LDA      #$81          RTS
        STA      WORK3
        LDA      #$F4          `BA`
        STA      DTABL+4
        LDA      #$77
        STA      DTABL+5
        JSR      DISTAB
        LDA      #$04          DEFAULT TO $0400
        STA      ADDRH
        LDA      #$00
        STA      ADDRL
        JSR      BLDA2          GET SOURCE ADDR.
        BCS      SKPC1          VALID ?
        CMP      #$11          NO, ENTER ?
        BNE      XAB           IF NOT THEN ABORT
SKPC1   LDA      ADDRH          YES
        STA      TEMP          NO SAVE IT
        LDA      ADDRL
        STA      TEMP+1
        LDA      #$F1          PRINT `EA`
        STA      DTABL+4
        LDA      #$77
        STA      DTABL+5
        JSR      DISTAB
        LDA      #$1F          DEFAULT TO $1FFF
        STA      ADDRH
        LDA      #$FF
        STA      ADDRL
        JSR      BLDA2          GET DESTINATION ADDR
        BCS      SKPC2          VALID ?
        CMP      #$11          NO, ENTER ?
        BNE      XAB           IF NOT THEN ABORT
SKPC2   LDA      ADDRH          SWAP ADDRESSES
        LDX      TEMP
        STA      TEMP
        STX      ADDRH
        LDA      ADDRL
        LDX      TEMP+1
        STA      ADDRL
        STA      TEMP+1

```

```

1515
1516
1517
1518
1519
1520
1521 e86e b6 22      XLOOP LDA   ADDRH
1522 e870 ab 40      ADD   #$40
1523 e872 b7 22      STA   ADDRH
1524 e874 cd e7 46   JSR   LOAD
1525 e877 9f          TAX
1526 e878 b6 22      LDA   ADDRH
1527 e87a a0 40      SUB   #$40
1528 e87c b7 22      STA   ADDRH
1529 e87e 9f          TAX
1530 e87f cd e7 57   JSR   STORE
1531 e882 24 03       BCC   RBOX
1532 e884 cc e4 76   JMP   ERR2
1533 e887 3c 23       RBOX INC   ADDR
1534 e889 26 02       BNE   XSKP
1535 e88b 3c 22       INC   ADDR
1536 e88d b6 22      XSKP LDA   ADDRH
1537 e88f b1 27       CMP   TEMP
1538 e891 26 db       BNE   XLOOP
1539 e893 b6 23       LDA   ADDR
1540 e895 b1 28       CMP   TEMP+1
1541 e897 26 d5       BNE   XLOOP
1542
1543 e899 cc e0 b4     XAB   JMP   GETCMD
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554 e89c 80          SIRQV
1555
1556 e89d cd e6 57   TIRQAV
1557 e8a0 8e          TIRQBV
1558 e8a1 20 fd       IRQV  RTI
1559
1560 e8a3 cd e6 57   PWRDWN JSR   CLRDIS
1561 e8a6 8f          STP    STOP
1562 e8a7 20 fd       BRA   STP
1563
1564 e8a3 cd e6 57   WDN   JSR   CLRDIS
1565
1566 e8a6 8f          WIT   WAIT
1567 e8a7 20 fd       BRA   WIT
1568
1569 e8a3 cd e6 57   ORG   $FFF4
1570
1571 e8a3 cd e6 57   FDB   RESET      SERIAL
1572 e8a6 8f          FDB   TIRQB      TIMER B
1573 e8a7 20 fd       FDB   TIRQA      TIMER A
1574 e8a8 00 00       FDB   IRQ        EXTERNAL INTERRUPT
1575 e8a9 00 00       FDB   SWI        SWI
1576 e8aa 00 00       FDB   RESET      RESET
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000

```


LOW VOLTAGE INHIBIT (LVI) CAPABILITY OF THE M6805 HMOS MICROCOMPUTER (MCU) FAMILY

Prepared by
Ed Edwards
Microprocessor Applications Engineering
Austin, Texas

INTRODUCTION

The low voltage inhibit (LVI) option, as used with many of the M6805 HMOS Family Microcomputer (MCU) devices (EPROM MCUs excluded), provides a means for the MCU to sense a drop in supply voltage (V_{CC}) and then shut itself down in a well-defined manner. The LVI option may be used in applications which require the correct-output during normal operation and no-output during loss of power. The LVI option is also useful in fail-safe and/or fall-back operating modes for minimum systems operation.

An example of this no-output control scheme is in height-positioning applications. With correct-output control signals from the MCU, the position of "cherry-picker" platforms could be controlled and maintained. With inadvertent loss of power due to power supply/battery deterioration and/or cable disconnects, the no-output signal would permit locking methods to be initiated within microseconds. This would allow the "cherry-picker" to maintain its "last position" (thus preventing disastrous falls for personnel or on-board equipment) or to start a predetermined (hydraulic self-locking or hydraulic bleed-off valves) controlled descent. This LVI capability of transition from correct-output to no-output (i.e., high-Z, 3-state) without an intermediate uncontrolled region is defined in the On-Chip Operation paragraph.

Another example of this correct-output usefulness is in starter-controls of multi-horsepower electric motors. During normal power transitions, the controller signals will start up the motor at prescribed voltage and current versus time relationships, and maintain specified shaft RPM and shaft output power afterwards. At low power conditions (i.e., brownout), the electric current required to maintain this same motor-rpm and shaft output power increases proportional to the voltage decrease; e.g., a 20% drop in powerline voltage causes an equivalent 20% increase in motor current. This 20% increase in motor current produces a 44% increase in internal I^2R loss. The internal I^2R loss may cause the motor to overheat or even burn out. The M6805 HMOS

MCU can detect this "brownout" (low voltage) condition and immediately place the MCU output control lines (normally used to turn on the motor controller) in the high-impedance state. Bipolar driver devices with built-in pull-down resistors are usually used in these applications to turn themselves off when the control input is open-circuited. Restart can be then prohibited until normal power conditions return and the original current versus time relationships are reestablished, within the controller.

The LVI option is provided at the time of manufacture by on-chip circuitry, as a mask option, contained in part of the users ROM pattern. When the LVI option is provided, no additional external parts are required for normal operation. The LVI option will usually provide for an overall product cost reduction by eliminating the external components required to implement this feature off-chip.

ON-CHIP RESET/LVI OPERATION

A simplified equivalent of the internal reset and LVI circuitry is shown in Figure 1. The circuit consists of three basic sections: (1) internal reset generator, (2) an internal Schmitt trigger which is externally activated, plus bias circuits, clamping diodes and current limiting, and (3) a low voltage detector with gating logic.

LVI DISABLED

Without the LVI option (i.e., LVI disabled), the internal reset generator is only actuated by the external $\overline{\text{RESET}}$ pin via the Schmitt trigger. In this case during power-on reset (POR), external capacitor C_R (0.1 to 1.0 microfarad) is charged through an on-chip resistor (R_C) and the current source from V_{CC} . When the $\overline{\text{RESET}}$ pin voltage rises to the Schmitt trigger positive threshold ($V_{\text{IRES}+}$), the on-chip reset generator allows the CPU to begin executing from ROM. The RC delay (TR_{HL}) allows the on-chip oscillator to stabilize prior to start of program execution. During normal power turn-off, the on-chip reset generator is actuated by the

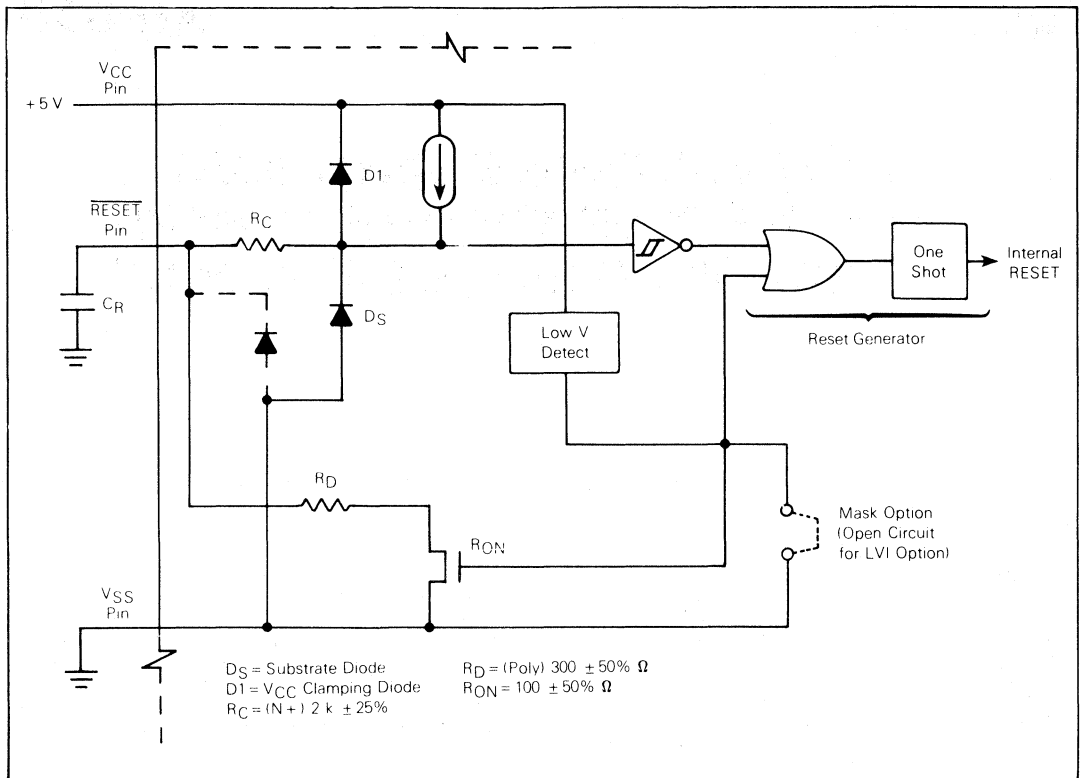


FIGURE 1 — M6805 HMOS Family LVI Simplified Schematic Diagram

Schmitt trigger after the decreasing voltage on the $\overline{\text{RESET}}$ pin falls to the negative threshold voltage ($V_{\text{IRES-}}$). The individual M6805 HMOS Family data sheet and Figure 2 (of this application note) contain information concerning the reset and LVI timing waveforms, and V_{CC} voltage spectrum.

LVI ENABLED

With the LVI mask option (i.e., LVI enabled), the power-on sequence is exactly the same as described above. However, in the power-down condition (resulting from normal power turn off, brownout, or voltage "dip"), the on-chip reset generator is triggered by the low voltage detector before power falls below the reset level. In this case (as shown in Figure 1), the second input to the reset generator OR gate becomes functional and the low voltage detect circuit causes a reset at a voltage point (V_{LVI}) prior to the $\overline{\text{RESET}}$ pin

reaching the Schmitt trigger $V_{\text{IRES-}}$ threshold. The only requirement is that V_{LVI} remains at its threshold for one t_{cyc} (minimum). In typical applications, the V_{CC} bus filter capacitor will eliminate negative-going voltage glitches of less than one t_{cyc} . Once the V_{LVI} threshold is reached for one t_{cyc} , the low voltage detect circuit outputs a logic 1 which is gated to the on-chip reset generator, resetting the CPU within the next t_{cyc} period.

Simultaneous to the activation of the on-chip reset generator, the low voltage detector turns on the internal R_{ON} device. With the R_{ON} device turned on, the external reset capacitor discharges through internal current limiter R_D and continues until the $\overline{\text{RESET}}$ pin voltage falls below the minimum reset voltage, holding the CPU in reset. This condition remains until recovery of V_{CC} , at which time normal power-on reset resumes.

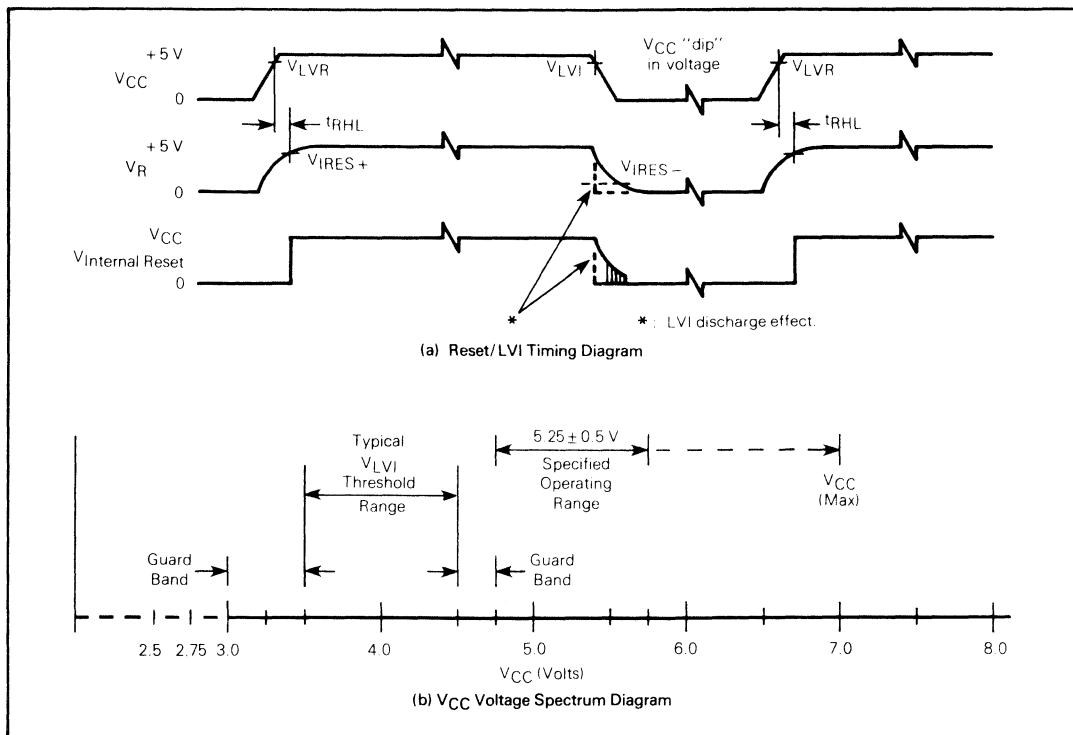


FIGURE 2 — M6805 HMOS Family Reset/LVI Timing and VCC Voltage Spectrum Diagrams

LVI TESTING

Figure 3 shows an LVI test circuit connection for the MC6805P4L1 (or P1). Similar connections for other M6805 HMOS Family MCUs could be made to corresponding pins for LVI testing. This circuit, together with the software in Figure 4, is used to determine the V_{LVI} and V_{LVR} for the MCU under test.

Figure 4 is the test pattern (TEST P) software routine to be entered into RAM in order to generate a continuous output square wave on the PB0 pin. By utilizing the on-chip monitor capability of the MC6805P4L1 demonstration program, the TEST P program can be loaded into on-chip RAM via an RS-232-C terminal. This connection is shown schematically in Figure 3. In typical lab applications, the 15 k pullup resistor is adequate to provide the required operating frequency. Alternatively, an adjustable resistor may be used to set the frequency at 3.58 MHz.

In order to activate the LVI state, the supply voltage (V_{CC}) must drop below V_{LVI} and remain there for one t_{CYC} (internal clock period) plus 250 nanoseconds. The M6805 HMOS Family V_{CC} voltage spectrum is shown in Figure 2b. To determine the V_{LVI} trip point, reduce the V_{CC} from the

normal value (5.25 Vdc) to the point where the TEST P square wave disappears (i.e., PB0 switches to high impedance). This is the V_{LVI} value for this particular device. When V_{LVI} is attained by reducing V_{CC} , the three-state leakage current (I_{TSI}) can then be measured to check that all I/O port pins are in the high-impedance state.

Increase V_{CC} in small increments (10-100 millivolt steps) and reinitiate the test pattern TEST P by keying the Execute address \$040 into the RS-232-C terminal. This determines the V_{LVR} value for the device being tested. (Note that the TEST P program will be retained in the on-chip RAM at the V_{LVI} voltage level.)

TYPICAL APPLICATION

A circuit that can be used to directly control a Darlington bipolar solenoid driver is shown in Figure 5. The MC6805P4L1 was chosen for the circuit because of its larger RAM area (112 bytes versus 64 bytes on other masked ROM devices). As shown in this figure, the port B (PB0) high current drive capability is used for this circuit. When the LVI capability is utilized, the solenoid immediately turns off with loss of power or brownout condition.

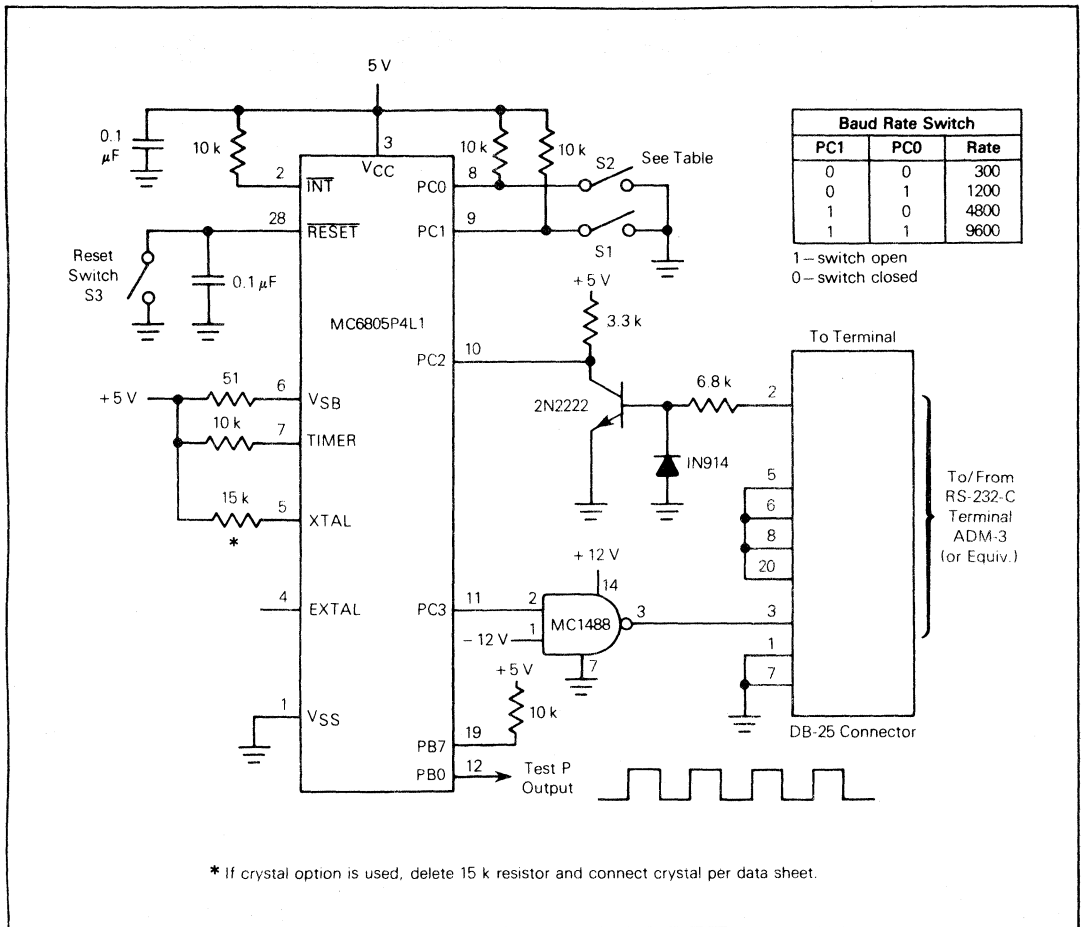


FIGURE 3 - MC6805P4 LVI Test Circuit Schematic Diagram

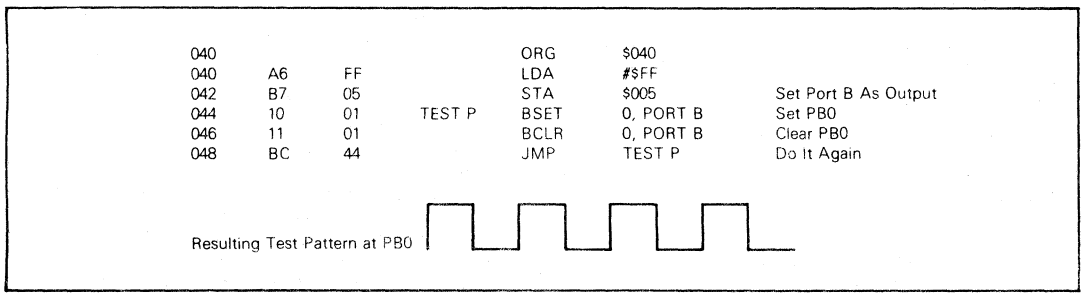


FIGURE 4 - Test Pattern Routine (TEST P)

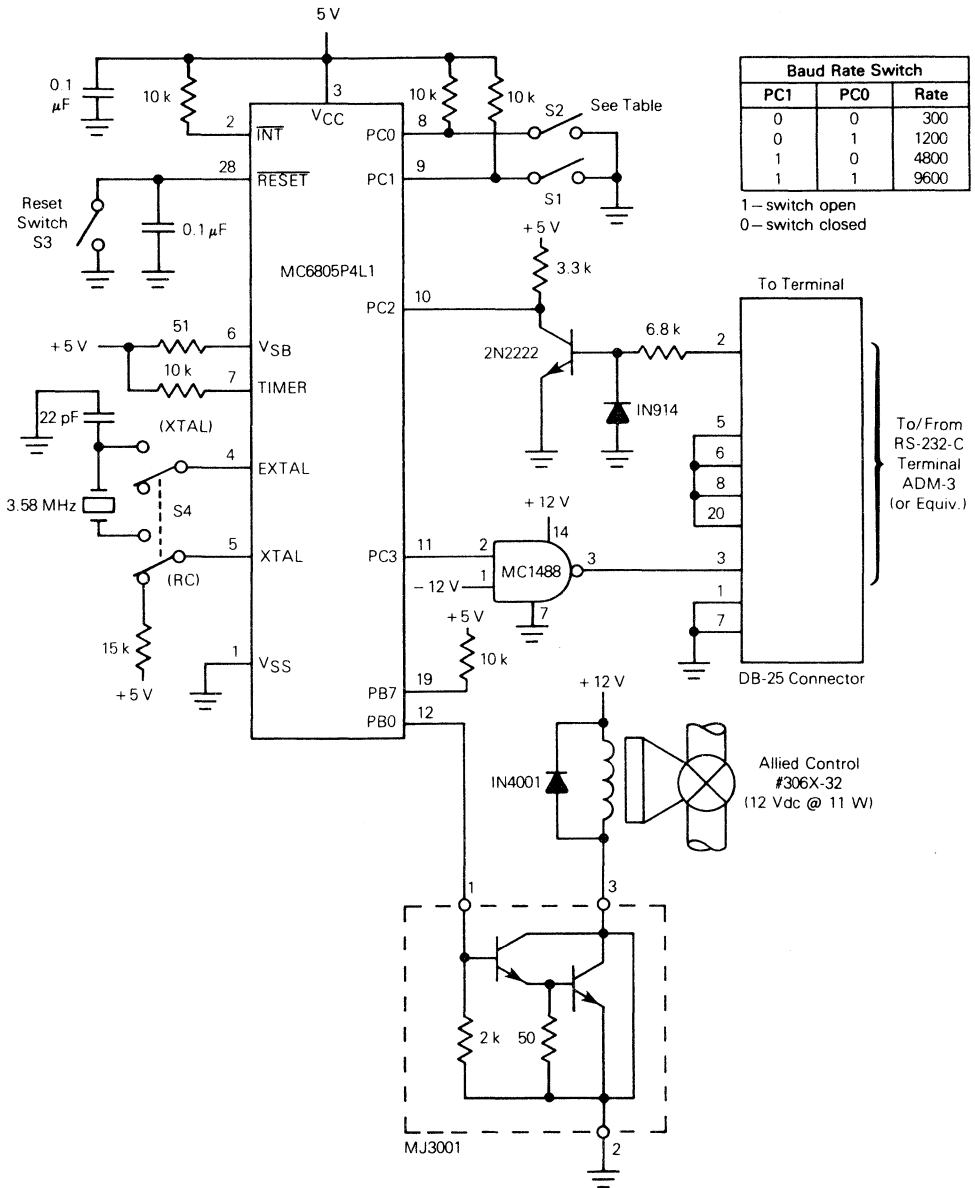


FIGURE 5 — MC6805P4 With Darlington Connected Solenoid Driver

Software to control the solenoid (via port B, PBO) is shown in Figure 6. This software routine may be entered into on-chip RAM via the RS-232-C terminal with the MC6805P4L1 in the monitor mode. The on-time of the solenoid (Allied Controls #306X-32) is set by the value entered (by the RS-232-C terminal) in RAM location \$04F. The on/off times for this particular solenoid and software timing loops are shown below.

\$04F = 04; 9.6 ms/9.6 ms

= 0F; 36 ms/36 ms

= 4F; 188 ms/188 ms

= FF; 620 ms/620 ms

The minimum on/off time of the solenoid shown in Figure 5 is 13 milliseconds. The 9.6 millisecond value (RAM \$04F = 04 in Figure 6) only caused the solenoid to chatter; therefore, a higher hexadecimal value may be required in location \$04F.

MONITOR SOFTWARE

A listing of the monitor (and self-check) program in the MC6805P4L1 (P1) is attached to this application note. This monitor program permits selection (via switch positions) of

four baud rates which allows the user to enter and execute small software programs directly from on-chip RAM, as is done in this application note.

PORT I/O CHARACTERISTICS WITH LVI OPTION

The device operates successfully down to the V_{LVI} threshold. During a lowering of V_{CC} voltage, due to a brownout condition of electrical power or other reason, the LVI option provides the user with reduced drive capability but a stable port configuration. A discussion of these characteristics is provided below.

The LVI threshold range may be between +2.7 V and +4.7 Vdc. The exact level could be determined as described above for the LVI Testing. Between the range of +4.7 Vdc and the actual V_{LVI} voltage, each I/O port configuration remains as programmed. However, the drive capabilities (as a percentage of the specified port dc electrical characteristics) between the 4.7 Vdc and the actual V_{LVI} voltage are as follows:

- (1) I_{LOAD} Sink — 100% of specified current sinking capability for ports A and C, and 50% of specified sinking capability for port B.
- (2) I_{LOAD} Source — 0% of specified drive current. That is, do not depend upon the M6805 HMOS Family MCU to source current below the lower V_{CC} limit of +4.75 Vdc.

00001			0005	A	DDRB	EQU	\$05	
00002			0001	A	PORT B	EQU	\$01	
00003A	0040					ORG	\$040	
00004A	0040	AG	FF	A	TEST01	LDA	#\$FF	
00005A	0042	B7	05	A		STA	DDRB	
00006A	0044	10	01	A		BSET	0,PORT B	
00007A	0046	AD	06	004E		BSR	DELAY	
00008A	0048	11	01	A		BCLR	0,PORT B	
00009A	004A	AD	02	004E		BSR	DELAY	
00010A	004C	BC	40	A		JMP	TEST01	
00011A	004E	AE	04	A	DELAY	LDX	#\$04	LOAD FROM RS-232
00012A	0050	A6	FF	A	DELAY1	LDA	#\$FF	
00013A	0052	4A			DELAY2	DECA		
00014A	0053	26	FD	0052		BNE	DELAY2	
00015A	0055	5A				DEX		
00016A	0056	26	F8	0050		BNE	DELAY1	
00017A	0058	81				RTS		
00018						END		

FIGURE 6 — Software Listing for Controlling Solenoid via PBO

MC6805P4 ROM PATTERN

THIS ROM CONTAINS A CUSTOMER PROGRAM, THE STANDBY MONITOR AND A NEW VERSION OF THE SELF CHECK. THE CUSTOMER ROM OCCUPIES \$80-\$FF AND \$3C0-\$5D8. THIS LEAVES \$5D9-\$783 FOR THE MONITOR. MOST OF THE MONITOR COMMANDS HAVE BEEN DELETED TO MAKE IT FIT WITHIN THIS AREA. THE SELF CHECK IS DIFFERENT FROM THE P2 PATTERN SINCE THERE IS MORE RAM ON THIS PART. THE OLD P2 SELF CHECK WAS VERY TIGHTLY CODED AND DEPENDED ON THERE BEING EXACTLY 64 BYTES OF RAM. THE P4 HAS 112 BYTES OF RAM AND REQUIRES A NEW TEST. THIS MESSES THINGS UP SOMEWHAT SINCE THE SAME GOOD TEST CANNOT BE APPLIED TO THE P4.

THE MONITOR IS SELECTED WHEN BIT 7 OF PORT B IS HIGH DURING RESET. OTHERWISE, THE CUSTOMER PROGRAM WILL BE SELECTED. THE MONITOR MODE IS AN ENHANCED VERSION OF THE ORIGINAL P2 VERSION.

SERIAL I/O IS HANDLED ON PORT C. BIT 3 IS THE SERIAL OUTPUT LINE AND BIT 2 IS THE SERIAL INPUT LINE. THE LOWER TWO BITS OF PORT C SELECT THE BAUD RATE FOR ALL SERIAL I/O.

C1	C0	BIT RATE/SEC
0	0	300
0	1	1200
1	0	4800
1	1	9600

ED RUPP MAY 11, 1981

I/O REGISTER ADDRESSES

0000 00 00	PORTA	EQU	\$000	I/O PORT 0
0000 00 01	PORTB	EQU	\$001	I/O PORT 1
0000 00 02	PORTC	EQU	\$002	I/O PORT 2
0000 00 04	DDR	EQU	4	DATA DIRECTION REGISTER OFFSET
0000 00 08	TIMER	EQU	\$008	8-BIT TIMER REGISTER
0000 00 09	TCR	EQU	\$009	TIMER CONTROL REGISTER
0000 00 10	RAM	EQU	\$010	START OF ON-CHIP RAM AREA
0000 00 80	ZROM	EQU	\$080	START OF PAGE ZERO ROM
0000 03 C0	ROM	EQU	\$3C0	START OF MAIN ROM AREA
0000 08 00	MEMSIZ	EQU	\$800	MEMORY ADDRESS SPACE SIZE
0000 05 D9	MONST	EQU	\$5D9	START OF MONITOR
0000 05 B3	CENTRY	EQU	\$5B3	ENTRY POINT TO CUSTOMER PROGRAM

CHARACTER CONSTANTS

0000 00 0D	CR	EQU	\$0D	CARRIAGE RETURN
0000 00 0A	LF	EQU	\$0A	LINE FEED
0000 00 20	BL	EQU	\$20	BLANK
0000 00 07	BEEP	EQU	\$07	CONTROL-G (BELL)
0000 00 00	EOS	EQU	\$00	END OF STRING

R O M M O N I T O R F O R T H E 6 8 0 5 P 4

THE MONITOR HAS THE FOLLOWING COMMANDS:

M -- MEMORY EXAMINE/CHANGE.
 TYPE M AAA TO BEGIN,
 THEN TYPE: . -- TO RE-EXAMINE CURRENT
 @ -- TO EXAMINE PREVIOUS
 CR -- TO EXAMINE NEXT
 DD -- NEW DATA
 ANYTHING ELSE EXITS MEMORY COMMAND.

E -- EXECUTE FROM ADDRESS. FORMAT IS
 E AAA. AAA IS ANY VALID MEMORY ADDRESS.

SPECIAL EQUATES

0000 00 2E	PROMPT	EQU	'.	PROMPT CHARACTER
0000 00 0D	FWD	EQU	CR	GO TO NEXT BYTE
0000 00 5E	BACK	EQU	'@	GO TO PREVIOUS BYTE
0000 00 2E	SAME	EQU	'.	RE-EXAMINE SAME BYTE

OTHER

0000 00 7F	INITSP	EQU	\$7F	INITIAL STACK POINTER VALUE
0000 00 7A	STACK	EQU	INITSP-5	TOP OF STACK

RAM VARIABLES

0000	ORG	RAM		ON-CHIP RAM (64 BYTES)
0010	GET	RMB	4	NO-MANS LAND, SEE PICK AND DROP SUBROUTINES
0014	ATEMP	RMB	1	ACCA TEMP FOR GETC,PUTC
0015	XTEMP	RMB	1	IX TEMP FOR GETC,PUTC
0016	CHAR	RMB	1	CURRENT INPUT/OUTPUT CHARACTER
0017	COUNT	RMB	1	NUMBER OF BITS LEFT TO GET/SEND

0018	ORG	MONST		
------	-----	-------	--	--

MAIN --- PRINT PROMPT AND DECODE COMMANDS

05D9 CD 06 B1	MAIN	JSR	CRLF	GO TO NEXT LINE
05DC A6 2E		LDA	#PROMPT	
05DE CD 07 28		JSR	PUTC	PRINT THE PROMPT
05E1 CD 06 F6		JSR	GETC	GET THE COMMAND CHARACTER
05E4 A4 7F		AND	##11111111	MASK PARITY
05E6 CD 06 BE		JSR	PUTS	PRINT SPACE (WON'T DESTROY A)
05E9 A1 45		CMP	#'E	EXECUTE
05EB 27 0B		BEQ	EXEC	
05ED A1 4D		CMP	#'M	MEMORY
05EF 27 17		BEQ	MEMORY	
05F1 A6 3F		LDA	#'?	NONE OF THE ABOVE
05F3 CD 07 28		JSR	PUTC	
05F6 20 E1		BRA	MAIN	LOOP AROUND

```

*
*
* EXEC --- EXECUTE FROM GIVEN ADDRESS
*
05F8 CD 06 D8 EXEC JSR GETNYB GET HIGH NYBBLE
05FB 25 DC BCS MAIN BAD DIGIT
05FD 97 TAX SAVE FOR A SECOND
05FE CD 06 C7 JSR GETBYT NOW THE LOW BYTE
0601 25 D6 BCS MAIN BAD ADDRESS
0603 B7 7F STA STACK+5 PROGRAM COUNTER LOW
0605 BF 7E STX STACK+4 PROGRAM COUNTER HIGH
0607 80 RTI

*
* MEMORY --- MEMORY EXAMINE/CHANGE
*
0608 CD 06 D8 MEMORY JSR GETNYB BUILD ADDRESS
060B 25 CC BCS MAIN BAD HEX CHARACTER
060D B7 11 STA GET+1
060F CD 06 C7 JSR GETBYT
0612 25 C5 BCS MAIN BAD HEX CHARACTER
0614 B7 12 STA GET+2 ADDRESS IS NOW IN GET+1&2
0616 CD 06 B1 MEM2 JSR CRLF BEGIN NEW LINE
0619 B6 11 LDA GET+1 PRINT CURRENT LOCATION
061B CD 06 A0 JSR PUTNYB
061E B6 12 LDA GET+2
0620 CD 06 93 JSR PUTBYT
0623 CD 06 BE JSR PUTS A BLANK, THEN
0626 AD 33 BSR PICK GET THAT BYTE
0628 CD 06 93 JSR PUTBYT AND PRINT IT
062B CD 06 BE JSR PUTS ANOTHER BLANK,
062E CD 06 C7 JSR GETBYT TRY TO GET A BYTE
0631 25 06 BCS MEM3 MIGHT BE A SPECIAL CHARACTER
0633 AD 2C BSR DROP OTHERWISE, PUT IT AND CONTINUE
0635 AD 3A MEM4 BSR BUMP GO TO NEXT ADDRESS
0637 20 DD BRA MEM2 AND REPEAT
0639 A1 2E MEM3 CMP #SAME RE-EXAMINE SAME?
063B 27 D9 BEQ MEM2 YES, RETURN WITHOUT BUMPING
063D A1 0D CMP #FWD GO TO NEXT?
063F 27 F4 BEQ MEM4 YES, BUMP THEN LOOP
0641 A1 5E CMP #BACK GO BACK ONE BYTE?
0643 26 94 BNE MAIN NO, EXIT MEMORY COMMAND
0645 3A 12 DEC GET+2 DECREMENT LOW BYTE
0647 B6 12 LDA GET+2 CHECK FOR UNDERFLOW
0649 A1 FF CMP #$FF
064B 26 C9 BNE MEM2 NO UNDERFLOW
064D 3A 11 DEC GET+1
064F B6 11 LDA GET+1 SAME FOR HIGH NYBBLE
0651 A1 FF CMP #$FF
0653 26 C1 BNE MEM2 TO WRAP AROUND
0655 A6 07 LDA #$7 HIGHEST ADDRESS IS $7FF
0657 B7 11 STA GET+1
0659 20 BB BRA MEM2

*
* UTILITIES
*
* PICK --- GET BYTE FROM ANYWHERE IN MEMORY
* THIS IS A HORRIBLE ROUTINE (NOT MERELY
* SELF-MODIFYING, BUT SELF-CREATING)
*

```

```

*      GET+1&2 POINT TO ADDRESS TO READ,
*      BYTE IS RETURNED IN A
*      X IS UNCHANGED AT EXIT
*
065B BF 15      PICK      STX      XTEMP      SAVE X
065D AE D6      LDX      #$D6      D6=LDA 2-BYTE INDEXED
065F 20 04      BRA      COMMON
*
*
*      DROP --- PUT BYTE TO ANY MEMORY LOCATION.
*      HAS THE SAME UNDESIRABLE PROPERTIES
*      AS PICK
*      A HAS BYTE TO STORE, AND GET+1&2 POINTS
*      TO LOCATION TO STORE
*      A AND X UNCHANGED AT EXIT
*
0661 BF 15      DROP      STX      XTEMP      SAVE X
0663 AE D7      LDX      #$D7      D7=STA 2-BYTE INDEXED
*
*
0665 BF 10      COMMON    STX      GET      PUT OPCODE IN PLACE
0667 AE 81      LDX      #$81      81=RTS
0669 BF 13      STX      GET+3    NOW THE RETURN
066B 5F         CLRX     WE WANT ZERO OFFSET
066C BD 10      JSR      GET      EXECUTE THIS MESS
066E BE 15      LDX      XTEMP    RESTORE X
0670 81         RTS      AND EXIT
*
*
*      BUMP --- ADD ONE TO CURRENT MEMORY POINTER
*
*      A AND X UNCHANGED
*
0671 3C 12      BUMP      INC      GET+2    INCREMENT LOW BYTE
0673 26 02      BNE     BUMP2   NON-ZERO MEANS NO CARRY
0675 3C 11      INC      GET+1    INCREMENT HIGH NYBBLE
0677 81         BUMP2     RTS
*
*
*      OUT3HS --- PRINT WORD POINTED TO AS AN ADDRESS, BUMP POINTER
*      X IS UNCHANGED AT EXIT
*
0678 AD E1      OUT3HS   BSR      PICK      GET HIGH NYBBLE
067A A4 07      AND      #$1111  MASK UNUSED BITS
067C AD 22      BSR      PUTNYB   AND PRINT IT
067E AD F1      BSR      BUMP      GO TO NEXT ADDRESS
*
*
*      OUT2HS --- PRINT BYTE POINTED TO, THEN A SPACE. BUMP POINTER
*      X IS UNCHANGED AT EXIT
*
0680 AD D9      OUT2HS   BSR      PICK      GET THE BYTE
0682 B7 10      STA      GET      SAVE A
0684 44         LSRA     SHIFT HIGH TO LOW
0685 44         LSRA
0686 44         LSRA
0687 44         LSRA
0688 AD 16      BSR      PUTNYB
068A B6 10      LDA      GET
068C AD 12      BSR      PUTNYB

```

068E AD E1 BSR BUMP GO TO NEXT
 0690 AD 2C BSR PUTS FINISH UP WITH A BLANK
 0692 81 RTS

*
 * PUTBYT --- PRINT A IN HEX
 * A AND X UNCHANGED
 *

0693 B7 10 PUTBYT STA GET SAVE A
 0695 44 LSRA
 0696 44 LSRA
 0697 44 LSRA
 0698 44 LSRA SHIFT HIGH NYBBLE DOWN
 0699 AD 05 BSR PUTNYB PRINT IT
 069B B6 10 LDA GET
 069D AD 01 BSR PUTNYB PRINT LOW NYBBLE
 069F 81 RTS

*
 * PUTNYB --- PRINT LOWER NYBBLE OF A IN HEX
 * A AND X UNCHANGED, HIGH NYBBLE
 * OF A IS IGNORED.
 *

06A0 B7 13 PUTNYB STA GET+3 SAVE A IN YET ANOTHER TEMP
 06A2 A4 0F AND #\$F MASK OFF HIGH NYBBLE
 06A4 AB 30 ADD #'0 ADD ASCII ZERO
 06A6 A1 39 CMP #'9 CHECK FOR A-F
 06A8 23 02 BLS PUTNY2
 06AA AB 07 ADD #'A-'9-1 ADJUSTMENT FOR HEX A-F
 06AC AD 7A PUTNY2 BSR PUTC
 06AE B6 13 LDA GET+3 RESTORE A
 06B0 81 RTS

*
 * CRLF --- PRINT CARRIAGE RETURN, LINE FEED
 * A AND X UNCHANGED
 *

06B1 B7 10 CRLF STA GET SAVE
 06B3 A6 0D LDA #CR
 06B5 AD 71 BSR PUTC
 06B7 A6 0A LDA #LF
 06B9 AD 6D BSR PUTC
 06BB B6 10 LDA GET RESTORE
 06BD 81 RTS

*
 * PUTS --- PRINT A BLANK (SPACE)
 * A AND X UNCHANGED
 *

06BE B7 10 PUTS STA GET SAVE
 06C0 A6 20 LDA #BL
 06C2 AD 64 BSR PUTC
 06C4 B6 10 LDA GET RESTORE
 06C6 81 RTS

*
 * GETBYT --- GET A HEX BYTE FROM TERMINAL
 *
 * A GETS THE BYTE TYPED IF IT WAS A VALID HEX NUMBER,
 * OTHERWISE A GETS THE LAST CHARACTER TYPED. THE C-BIT IS
 * SET ON NON-HEX CHARACTERS; CLEARED OTHERWISE. X
 * UNCHANGED IN ANY CASE.
 *

```

06C7 AD 0F      GETBYT  BSR      GETNYB  BUILD BYTE FROM 2 NYBBLES
06C9 25 0C      BCS      NOBYT  BAD CHARACTER IN INPUT
06CB 48         ASLA
06CC 48         ASLA
06CD 48         ASLA
06CE 48         ASLA      SHIFT NYBBLE TO HIGH NYBBLE
06CF B7 10      STA      GET      SAVE IT
06D1 AD 05      BSR      GETNYB  GET LOW NYBBLE NOW
06D3 25 02      BCS      NOBYT  BAD CHARACTER
06D5 BB 10      ADD      GET      C-BIT CLEARED
06D7 81         NOBYT  RTS
*
*      GETNYB --- GET HEX NYBBLE FROM TERMINAL
*
*      A GETS THE NYBBLE TYPED IF IT WAS IN THE RANGE 0-F,
*      OTHERWISE A GETS THE CHARACTER TYPED. THE C-BIT IS SET
*      ON NON-HEX CHARACTERS; CLEARED OTHERWISE. X IS
*      UNCHANGED.
*
06D8 AD 1C      GETNYB  BSR      GETC   GET THE CHARACTER
06DA A4 7F      AND      #11111111 MASK PARITY
06DC B7 13      STA      GET+3  SAVE IT JUST IN CASE
06DE A0 30      SUB      #'0    SUBTRACT ASCII ZERO
06E0 2B 10      BMI      NOTHEX WAS LESS THAN '0'
06E2 A1 09      CMP      #9
06E4 23 0A      BLS      GOTIT
06E6 A0 07      SUB      #'A-'9-1 FUNNY ADJUSTMENT
06E8 A1 0F      CMP      #$F    TOO BIG?
06EA 22 06      BHI      NOTHEX WAS GREATER THAN 'F'
06EC A1 09      CMP      #9     CHECK BETWEEN 9 AND A
06EE 23 02      BLS      NOTHEX
06F0 98         GOTIT  CLC      C=0 MEANS GOOD HEX CHAR
06F1 81         RTS
06F2 B6 13      NOTHEX LDA      GET+3  GET SAVED CHARACTER
06F4 99         SEC
06F5 81         RTS      RETURN WITH ERROR
*
*      S E R I A L I / O R O U T I N E S
*
*      DEFINITION OF SERIAL I/O LINES
*
06F6 00 02      PUT     EQU     PORTC  SERIAL I/O PORT
06F6 00 02      IN      EQU     2     SERIAL INPUT LINE#
06F6 00 03      OUT     EQU     3     SERIAL OUTPUT LINE#
*
*      GETC --- GET A CHARACTER FROM THE TERMINAL
*
*      A GETS THE CHARACTER TYPED, X IS UNCHANGED.
*
*      INTERRUPTS ARE MASKED ON ENTRY
*      AND UNMASKED ON EXIT.
*
06F6 BF 15      GETC   STX     XTEMP  SAVE X
06F8 A6 08      LDA     #8     NUMBER OF BITS TO READ
06FA B7 17      STA     COUNT
06FC 04 02 FD   GETC4  BRSET  IN,PUT,GETC4 WAIT FOR HILO TRANSITION
*
*      DELAY 1/2 BIT TIME

```

```

*
06FF B6 02          LDA    PUT
0701 A4 03          AND    #111  GET CURRENT BAUD RATE
0703 97             TAX
0704 DE 07 6D       LDX    DELAYS,X GET LOOP CONSTANT
0707 A6 04          GETC3  LDA    #4
0709 4A             GETC2  DECA
070A 26 FD          BNE    GETC2
070C 4D             TSTA   LOOP PAD
070D 5A             DECX
070E 26 F7          BNE    GETC3  MAJOR LOOP TEST

*
*
*
NOW WE SHOULD BE IN THE MIDDLE OF THE START BIT

0710 04 02 E9       BRSET  IN,PUT,GETC4 FALSE START BIT TEST
0713 7D             TST   ,X    MORE TIMING DELAYS
0714 7D             TST   ,X

*
*
*
MAIN LOOP FOR GETC

0715 AD 3E          GETC7  BSR   DELAY  COMMON DELAY ROUTINE
0717 05 02 00       BRCLR  IN,PUT,GETC6  TEST INPUT AND SET C-BIT
071A 7D             GETC6  TST   ,X    TIMING EQUALIZER
071B 36 16          ROR   CHAR  ADD THIS BIT TO THE BYTE
071D 3A 17          DEC   COUNT
071F 26 F4          BNE    GETC7  STILL MORE BITS TO GET(SEE?)

*
0721 AD 32          BSR   DELAY  WAIT OUT THE 9TH BIT
0723 B6 16          LDA   CHAR  GET ASSEMBLED BYTE
0725 BE 15          LDX   XTEMP RESTORE X

*
*
*
0727 81             RTS    AND RETURN

*
*
*
PUTC --- PRINT A ON THE TERMINAL
*
*
*
X AND A UNCHANGED
*
*
*
SAME GAMES ARE PLAYED WITH THE I-BIT AS IN GETC
*
0728 B7 16          PUTC  STA   CHAR
072A B7 14          STA   ATEMP  SAVE IT IN BOTH PLACES
072C BF 15          STX   XTEMP  DON'T FORGET ABOUT X
072E A6 09          LDA   #9    GOING TO PUT OUT
0730 B7 17          STA   COUNT  9 BITS THIS TIME
0732 5F             CLRX  FOR VERY OBSCURE REASONS
0733 98             CLC   THIS IS THE START BIT
0734 9B             SEI   MASK INTERRUPTS WHILE SENDING
0735 20 02          BRA   PUTC2  JUMP IN THE MIDDLE OF THINGS

*
*
*
MAIN LOOP FOR PUTC
*
0737 36 16          PUTC5  ROR   CHAR  GET NEXT BIT FROM MEMORY
0739 24 04          PUTC2  BCC   PUTC3  NOW SET OR CLEAR PORT BIT
073B 16 02          BSET  OUT,PUT
073D 20 04          BRA   PUTC4
073F 17 02          PUTC3  BCLR  OUT,PUT
0741 20 00          BRA   PUTC4  EQUALIZE TIMING AGAIN
0743 DD 07 55       PUTC4  JSR   DELAY,X MUST BE 2-BYTE INDEXED JSR

```

```

*
* THIS IS WHY X MUST BE ZERO
0746 3A 17      DEC    COUNT
0748 26 ED      BNE    PUTC5  STILL MORE BITS
074A 14 02      BSET   IN,PUT  7 CYCLE DELAY
074C 16 02      BSET   OUT,PUT  SEND STOP BIT
*
*
074E AD 05      BSR    DELAY  DELAY FOR THE STOP BIT
0750 BE 15      LDX    XTEMP  RESTORE X AND
0752 B6 14      LDA    ATEMP  OF COURSE A
0754 81        RTS
*
* DELAY --- PRECISE DELAY FOR GETC/PUTC
*
0755 B6 02      DELAY  LDA    PUT    FIRST, FIND OUT
0757 A4 03      AND    #111   WHAT THE BAUD RATE IS
0759 97        TAX
075A DE 07 6D   LDX    DELAYS,X LOOP CONSTANT FROM TABLE
075D A6 F8      LDA    #F8    FUNNY ADJUSTMENT FOR SUBROUTINE OVERHEAD
075F AB 09      DEL3  ADD    #S09
0761 4A        DEL2  DECA
0762 26 FD      BNE    DEL2
0764 5D        TSTX   LOOP PADDING
0765 14 02      BSET   IN,PUT  DITTO
0767 5A        DECX
0768 26 F5      BNE    DEL3    MAIN LOOP
076A A6 00      LDA    #0      FINAL TINY DELAY
076C 81        RTS    WITH X STILL EQUAL TO ZERO
*
* DELAYS FOR BAUD RATE CALCULATION
*
* THIS TABLE MUST NOT BE PUT ON PAGE ZERO SINCE
* THE ACCESSING MUST TAKE 6 CYCLES.
*
076D 20      DELAYS  FCB    32    300 BAUD
076E 08      FCB    8     1200 BAUD
076F 02      FCB    2     4800 BAUD
0770 01      FCB    1     9600 BAUD
*
* RESET --- POWER ON RESET ROUTINE
*
* CHECK FOR CUSTOMER OR MONITOR MODE AND BRANCH
* ACCORDINGLY.
*
0771        RESET
0771 0E 01 03  BRSET  7,PORTB,MONIT
0774 CC 05 B3  JMP    CENTRY  CUSTOMER PROGRAM ENTRY POINT
0777        MONIT
0777 A6 08      LDA    #1000  SETUP PORT FOR SERIAL IO
0779 B7 02      STA    PUT    SET OUTPUT TO MARK LEVEL
077B B7 06      STA    PUT+DDR  SET DDR TO HAVE ONE OUTPUT
077D B6 02      LDA    PUT
077F 83        SWI    GO TO MONITOR ROUTINE
0780 20 EF      BRA    RESET  LOOP AROUND
*
*
*****
*
* 6 8 0 5   S E L F T E S T

```

GENERAL:

SELFTEST PERFORMS THE FOLLOWING TESTS:

- PORTC -- TEST PORTC FOR GOOD INPUT
- I/O -- PORT A AND B FOR INPUT AND OUTPUT
- RAM -- WALKING BIT MEMORY TEST
- ROM -- EXCLUSIVE OR WITH ODD 1'S PARITY RESULT
- INTERRUPT -- INT AND TIMER INTERRUPTS

PORTC TEST IS DONE ONLY ONCE JUST AFTER RESET. THE OTHERS ARE REPEATED AS LONG AS NO ERRORS ARE FOUND.

THE TEST STATUS IS AVAILABLE ON PORTC. AFTER THE PORTC TEST, PORTC IS USED AS AN OUTPUT TO INDICATE WHICH TEST IS RUNNING. IF ANY TEST FAILS, THE OUTPUT LINES REMAIN STABLE AND THE LOWER 2 BITS OF PORTC INDICATE THE TEST THAT FAILED (ASSUMING PORTC WORKS!). SINCE THERE ARE ONLY 4 'CHECKPOINTS' IN THE MAIN LOOP, IT IS NOT POSSIBLE TO ALWAYS DETERMINE EXACTLY WHAT IS WRONG WITH THE PART. HERE ARE THE PROBABLE ERRORS IF THE TEST HAS STOPPED:

B1	B0	PROBLEM
0	0	INTERRUPT FAILURE
0	1	BAD PORTA OR PORTB
1	0	BAD RAM
1	1	BAD ROM

THE TIME REQUIRED FOR ONE CYCLE OF THE PROGRAM IS APPROXIMATELY 800 MILLISECONDS. AFTER EACH TEST, PORTC IS INCREMENTED. THEREFORE, BIT 1 SHOULD APPEAR TO OSCILLATE WITH A PERIOD OF ABOUT 800MS. IT MAY BE DIFFICULT TO DISCERN THIS SINCE THE DUTY CYCLE ON BIT 1 IS NOT ANYWHERE NEAR A SQUARE WAVE. BIT 2 AND BIT 3 WILL HOWEVER HAVE SQUARE WAVE OUTPUTS. BIT 2'S PERIOD WILL BE TWICE AS LONG AS BIT 1, AND BIT 3 SHOULD BLINK AT ABOUT 0.3HZ.

THROUGHOUT THIS PROGRAM, IT MAY SEEM THAT THE PROGRAMMERS HAVE GONE OUT OF THEIR WAY TO USE STRANGE INSTRUCTIONS AND UNUSUAL TECHNIQUES. THIS IS TRUE. THE PURPOSE OF ALL THIS IS TO 1) REDUCE THE PROGRAM SIZE AS MUCH AS POSSIBLE AND 2) TO EXECUTE A BROAD RANGE OF INSTRUCTION TYPES AND ADDRESSING MODES.

0782

PAG

EQUATES

0782 00 00	ANY	EQU	0	RANDOM BIT IN A BYTE
0782 00 3F	RAMSUB	EQU	\$3F	START OF BUILT SUBROUTINE
0782 00 20	LOC	EQU	\$20	USED IN INTERRUPT TEST
0782 00 7C	STACKA	EQU	\$7C	A WHEN STACKED BY INTERRUPT
0782 00 7D	STACKX	EQU	\$7D	X THE SAME


```

0782                *          ORG      MEMSIZ-116-8
                    *
                    *          BEGIN OF SELF TEST
                    *
0784 9C            START  RSP          RESET JUST IN CASE (AVOID STACK INIT PROBLEM)
0785 33 02        COM     PORTC      SHOULD HAVE READ $FF
0787 26 FE        BNE     *          PORTC BAD ON INPUT

                    *
                    *          COM SET PORTC TO ZERO
                    *
0789 10 06        BSET   ANY,PORTC+DDR AND PROGRAM DATA DIRECTION

                    *
                    *          MAIN LOOP (REPEATED)
                    *
078B 3F 09        LOOP   CLR          TIMER+1 RESET TIMER INTERRUPTS
                    *
078D 83          SWI     PASSED FIRST TEST

                    *
                    *          INPUT/OUTPUT PORT TESTS
                    *
078E AE 01        IOTST  LDX      #PORTB POINT TO PORTB FIRST
0790 A6 F0        IOTST2 LDA      #$F0
0792 AD 44        BSR     IOSUB    RETURNS WITH A=$54
0794 A9 BA        ADC     #$0F-$54-1 A IS SET TO $0F
0796 AD 40        BSR     IOSUB    AGAIN WITH NYBBLES REVERSED
0798 5A          DECX
0799 27 F5        BEQ     IOTST2   AGAIN FOR PORTA
079B 4F          CLRA    CLEAR A FOR NEXT TEST

                    *
079C 83          SWI     PASSED I/O TEST

                    *
                    *          RAM TEST
                    *
                    *          ENTER WITH C=1, A=0.
                    *
                    *          THIS TEST IS MODIFIED FROM THE ORIGINAL P2 RAM TEST.  A
                    *          NEW TEST IS NECESSARY BECAUSE THERE ARE 112 BYTES OF RAM
                    *          ON THE P4.  THE OLD RAM TEST WORKED BECAUSE THE TEST
                    *          SEQUENCE WAS 9 BYTES LONG, AND THERE WERE 64 BYTES OF
                    *          RAM.  64 MOD 9 IS 1, SO A SINGLE ROTATE WAS SUFFICIENT TO
                    *          RESTORE THE SEED.  112 MOD 9 IS 4 WHICH REQUIRES 4
                    *          ROTATES TO GET BACK TO THE INITIAL PATTERN.  THIS
                    *          EXCEEDS THE AVAILABLE ROM.
                    *
                    *          THE NEW RAM TEST IS MUCH SIMPLER, BUT WILL TEST ALL THE
                    *          BYTES.  THIS CHANGE AFFECTS THE SUBSEQUENT TESTS SINCE
                    *          THEY MUST NOW INITIALIZE THEIR OWN VARIABLES INSTEAD OF
                    *          RELYING ON THE RAM TEST TO DO IT FOR THEM.
                    *
                    *          RAM WILL BE ALL ZEROES AFTER THIS TEST.
                    *
079D          RAMTST
079D AE 10        LDX     #RAM
079F F7          RAM2   STA     ,X      CLEAR OUT A BYTE
07A0 F1          RAM3   CMP     ,X      COMPARE WITH ACCUMULATOR VALUE
07A1 26 FE        BNE     *          MEMORY MISMATCH
07A3 4C          INCA
07A4 7C          INC     ,X      BUMP BOTH VALUES

```

PAGE 011 MONIT .P4:1

```
07A5 26 F9          BNE      RAM3
07A7 5C              INCX     ADVANCE TO NEXT RAM BYTE
07A8 2A F5          BPL      RAM2      CONTINUE TILL ALL TESTED
*
*          EXIT WITH A=0, X=$80
*
07AA 83              SWI          PASSED RAM TEST
*
*          ROM TEST
*
*          FORCE A RESULT OF $FF (1'S PARITY). THE CHECKSUM HAS BEEN
*          CHOSEN TO FORCE THIS RESULT IF THE ROM ITSELF IS GOOD.
*          ADDRESS AND DATA LINES STUCK HIGH, LOW OR TO EACH OTHER
*          ARE DETECTED WITH HIGH PROBABILITY.
*
*          FOR A DISCUSSION OF THE RELIABILITY OF THIS METHOD OF
*          TESTING THE ROM, SEE "MICROPROCESSOR BASED DESIGN" BY DR.
*          J. B. PEATMAN, PAGES 315-316.
*
*          ENTER WITH A=$00,X=$80,RAM=00
*
ROMTST
07AB              STX      RAMSUB+2
07AB BF 41          LDX     #$C8      =EOR EXTENDED
07AD AE C8          STX     RAMSUB
07AF BF 3F          LDX     #$81      =RTS
07B1 AE 81          STX     RAMSUB+3
07B3 BF 42          JSR     RAMSUB
07B5 BD 3F          INC     RAMSUB+2 ADVANCE TO NEXT ADDRESS
07B7 3C 41          BHI     SUM
07B9 22 FA          INC     RAMSUB+1
07BB 3C 40          BRCLR  3,RAMSUB+1,SUM LOOK FOR $8 IN RAMSUB+1
07BD 07 40 F5       COMA    A SHOULD HAVE BEEN $FF,
07C0 43              BNE     *          HANG HERE FOR BAD ROM
07C1 26 FE
*
07C3 83              SWI          PASSED ROM TEST
*
*          INTERRUPTS TEST
*
*          ENTER WITH: X=$81
*                   A=$00
*                   LOC=$80
*
*          INTERRUPT TEST ALLOWS INTERRUPTS LONG ENOUGH TO GET ONE
*          TIMER INTERRUPT AND ONE INT INTERRUPT. THE INTERRUPT
*          SERVICE ROUTINES SHIFT A BYTE IN MEMORY TO A KNOWN PATTERN
*          WHICH IS CHECKED AFTER THE INTERRUPTS SHOULD HAVE OCCURED.
*          FURTHER, THE A AND X REGISTER ARE COMPARED WITH WHAT WAS
*          STACKED DURING THE INTERRUPTS.
*
*          A TIMER INTERRUPT IS GUARANTEED PENDING BY ALLOWING ENOUGH
*          TIME TO ELAPSE DURING THE OTHER TESTS TO UNDERFLOW THE
*          COUNTER (EVEN WITH MAXIMUM PRE-SCALE). THE INT INTERRUPT
*          IS ALSO GUARANTEED SINCE THE INT LINE IS TIED TO THE PORTA
*          LINE WHICH WIGGLES UP AND DOWN DURING THE I/O TEST.
*
07C4              INTTST
```

```

07C4 1E 20      BSET    7,LOC    SET FLAG BIT IN LOC
07C6 9A         CLI      ALLOW INT AND TIMER INTERRUPT
07C7 9B         SEI      INTERRUPTS SHOULD BE DONE NOW
07C8 2C FE      BMC      *      I-BIT NOT WORKING
07CA B0 7C      SUB     STACKA  COMPARE A WITH STACKED VALUE
07CC 26 FE      BNE     *      INTERRUPT FAILED
07CE B3 7D      CPX     STACKX  ALSO CHECK X
07D0 26 FE      BNE     *
07D2 0B 20 FD    BRCLR   5,LOC,*  BIT SHOULD SHIFT HERE
*
*      END OF TESTS, DO IT AGAIN
*
*      NOTE THE USE OF A 2-BYTE INDEXED JUMP WHICH FORCES A CARRY
*      FROM THE LOWER BYTE IN COMPUTING THE JUMP ADDRESS.
*
07D5 DC 07 0A    JMP     LOOP-$81,X DO IT AGAIN
*
*      I/O TEST SUBROUTINE
*
*      ENTERED WITH X POINTING TO THE PORT
*      TO TEST. A HAS THE DDR PATTERN TO USE.
*
*      RETURNS WITH A=$54 AND C=1.
*
07D8 E7 04      IOSUB   STA     DDR,X   SETUP DATA DIRECTION
07DA A6 55      LDA     #$55   ALTERNATE 1'S AND 0'S
07DC F7         AGAIN  STA     ,X     SAVE IT AND SEE
07DD F1         CMP     ,X     IF IT STAYS THERE
07DE 26 FE      BNE     *      BAD I/O PORT(S)
07E0 48         LSLA   SHIFT PATTERN TO $AA
07E1 2B F9      BMI     AGAIN  AND REPEAT TEST
07E3 81         RTS
*
*      INTERRUPT ROUTINES
*
*      SOFTWARE INTERRUPT IS RESPONSIBLE FOR ADVANCING THE VALUE
*      STORED IN PORTC.
*
07E4 3C 02      SOFT   INC     PORTC  ADVANCE TO NEXT TEST
07E6 80         RTI     AND EXIT
*
*      INT AND TIMER INTERRUPTS
*      SHIFT LOC ONE PLACE TO THE RIGHT.
*
07E7 1F 09      TIMEUP BCLR   7,TIMER+1 HANDLE INTERRUPT
07E9 37 20      INTR   ASR     LOC     SHIFT FLAG AROUND
07EB 80         RTI
*
*
07EC 00 00 00   CHKSUM FCB     0,0,0   SPARE BYTES
07EF 9B         FCB     $9B   ROM CHECKSUM (ODD 1'S PARITY)
*
*      SELFTEST ALTERNATE INTERRUPT VECTORS
*
07F0 07 E7      FDB     TIMEUP
07F2 07 E9      FDB     INTR
07F4 07 E4      FDB     SOFT

```

PAGE 013 MONIT .P4:1

07F6 07 84	*	FDB	START	
07F8	*	END RESET WAZU		
	*			
	*	INTERRUPT VECTORS		
	*			
07F8	*	ORG	MEMSIZ-8	START OF VECTORS
07F8 05 6C		FDB	\$56C	CUSTOMER TIMER INTERRUPT
07FA 05 B3		FDB	\$5B3	CUSTOMER INT VECTOR
07FC 05 D9		FDB	MAIN	SWI TO MAIN ENTRY POINT OF MONITOR
07FE 07 71		FDB	RESET	POWER ON VECTOR

USING THE M6805 FAMILY ON-CHIP 8-BIT A/D CONVERTER

Prepared By:
Microcontroller Unit (MCU) Systems Application Engineering
Austin, Texas

INTRODUCTION

This application note covers some factors which should be considered when using on-chip analog-to-digital (A/D) converters. It is intended for the digital designer with little or no programming experience.

The task of converting analog signals into digital information is becoming easier using today's technology. The MC6805R3 and MC6805S2 microcomputer (MCUs) have an on-chip 8-bit A/D converter with four user channels to help accomplish this task. These versatile MCU devices are easy to use and serve well in automotive, industrial, medical or environmental systems where analog information is monitored, controlled, or stored.

The merging of analog and digital circuits does present a problem to the designer who is most familiar and comfortable using only one of the two disciplines. The problem arises when the analog designer needs to write a program to process the analog information or the digital designer has to contend with the analog hardware characteristics with which he is not totally familiar.

This application note defines the terminology used when discussing A/D converters, describes the circuit elements pertinent to the user, illustrates a self-test hardware/software technique, and gives an example on how to manipulate the converted analog data from a temperature sensor. Program listings are provided at the end of this application note.

DEFINITION OF TERMS

For a better understanding of the characteristics of the M6805 Family 8-bit A/D converter, the key terms used to describe the capabilities and limitations of an A/D converter, are presented first. Also included in this presentation is a brief interpretation of the specifications in the current MC6805R2 data sheet.

Conversion Range

The voltage reference high (V_{RH}) and the voltage reference low (V_{RL}) pins establish the voltage range that is recognized by the converter. The sum of V_{RH} and V_{RL}

should never exceed the V_{CC} of the MCU. If $V_{RH}-V_{RL}$ is below 4.000 volts, the specification accuracy can not be maintained. Operating below the minimum V_{RH} limit of 4.000 volts may cause the converter to miss steps or lose accuracy.

Conversion Time

The A/D converter in an M6805 Family MCU is a successive approximation type. A hardware binary-search method is used to determine the unknown input voltage. This method significantly reduces the time (total of 30 machine cycles) needed to analyze the input voltage and generate a hexadecimal value accurately. An M6805 Family MCU with a full-speed, 4-MHz crystal has a 1-microsecond machine cycle, and thus a 30 microsecond conversion time.

Monotonicity/Missing Codes

Monotonicity is a function of the integral non-linearity of the converter. An A/D converter is monotonic if the output digital value always increases or remains the same when the analog input voltage is increasing. A non-monotonic A/D converter is one that skips steps or decreases in output digital value when the input is increasing.

Non-Linearity

Studying the relationship of the analog input voltage with respect to the digital value that is derived after a conversion on an ideal A/D converter, yields a linear graph similar to the one shown in Figure 1. Integral non-linearity is defined as the deviation from an ideal straight-line, transfer function. Differential non-linearity is the amount that a particular step differs from one least significant bit (LSB). An ideal A/D converter has equal steps, and thus no differential non-linearity. Total non-linearity error is then the sum of the differential and integral errors.

Quantizing Error

The converter's inability to recognize voltage changes of less than one quantum step is referred to as the quantizing error. Since one increment or quantum is the smallest voltage

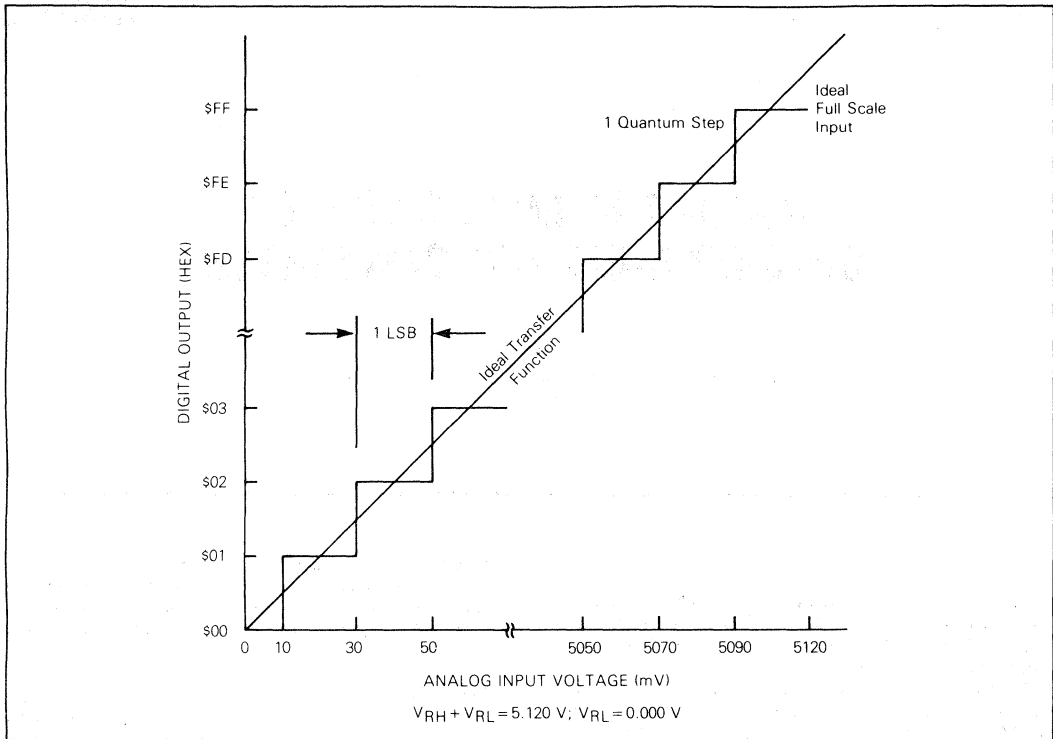


FIGURE 1 — Ideal A/D Conversion Points

value that the A/D converter recognizes, there is an inherent offset designed into the circuit. The M6805 Family A/D converter minimizes the impact of the offset error by centering it about the ideal conversion point.

Figure 1 shows the quantizing error to be ± 0.5 LSB. This offset enables the converter to detect input voltage changes ± 0.5 LSB from the center of a step. Thus, the first step occurs at 0.5 LSB and all subsequent steps occur at one LSB increments from this first 0.5 LSB step (that is, 1.5 LSB, 2.5 LSB, etc.). Also, the last step (\$FE to \$FF) is 1.5 LSB long because of this shift in the transfer function.

Ratiometric Reading/Gain Error

The gain error of the A/D comparator is the deviation from the ideal full scale input voltage and the full scale digital output value. The specification limit can be adjusted by trimming the V_{RH} reference voltage when the input voltage is at the transfer point between \$FE and \$FF (5.090 volts for a nominal $V_{RH} = 5.120$ volts system) and looking for an output value of \$FF.

Reference Voltage (V_{RH}/V_{RL})

The voltage reference high and voltage reference low pins provide the voltage limits to the digital-to-analog (D/A) converter resistor/capacitor chain in the A/D circuit. The nominal resistance value between these two pins is approx-

imately 6 kilohms. The reference voltage variation during a conversion should not exceed 0.25 LSB.

Resolution

When referring to an analog voltage range, there are an infinite number of steps that can be realized. Since it is unrealistic to try and cover all the possible steps, a method of approximating the unknown value using a digital circuit was developed. If a data converter has 8-bits, the 8 refers to the number of steps in powers of two. For example, an 8-bit A/D converter has 256 steps (2^8).

The number of steps is used to determine the A/D resolution. Resolution is defined as the full scale input voltage divided by the total number of steps. This can be represented by the equation:

$$\text{Resolution} = (V_{RH} - V_{RL}) / \text{Number of Steps}$$

The V_{RH} (voltage reference high) is the highest voltage that can be converted, and V_{RL} (voltage reference low) is the low end of the conversion range.

This means that the converter recognizes input voltage changes no smaller than one incremental value. One increment can be expressed as a fraction of the full scale voltage or one least significant bit (LSB) of the N-bit digital word.

The 8-bit A/D converter implemented in an M6805 Family MCU uses a unipolar binary code. A value of \$00 represents an analog input voltage of 0.000 to 0.020 volts and a value of

\$FF represents an analog input voltage of 5.100 to 5.120 volts, provided $V_{RH} = 5.120$ volts and $V_{RL} = 0.000$ volts. A dollar sign (\$) in front of a symbol identifies it in this document as a hexadecimal number (base 16 number).

Sample Time/Sample and Hold Capacitance

The sample time of the M6805 Family A/D converter is the first five machine cycles. This is the actual aperture window during which the selected analog channel coupler connects the sample and hold capacitor to the input voltage. The internal sample and hold capacitor charges for five machine cycles, and then holds that charge on the comparator input during the remaining 25 cycles of the conversion period.

The analog input voltage ideally should stay constant during the sample period. Any variation contributes to the conversion error. It is desirable to hold the input voltage within 0.5 LSB variation during the sample window. Under extreme operating conditions, the first conversion obtained after selecting a new channel may be less accurate than later conversions of the channel. Input channel changes can affect the stored sample and charge.

The input voltage on the channel coupler should not exceed V_{RH} nor be less than V_{RL} . A voltage greater than V_{RH} converts to \$FF, but no overflow indication is provided. Since the maximum rate of change of the input signal is dependent upon the converter sample time, the maximum frequency of a sine wave input is: $F = 1/(TS \times \pi \times 2(N + 1))$ where TS is the sample time and N is the number of bits. With this equation, the slew rate of the signal does not exceed 0.5 LSB during a sample of the input voltage.

Zero Input Reading/Offset Error

The input offset voltage of the A/D comparator causes a shift in its transfer function. The quantizing error inherent in the design of the converter causes the 0.5 LSB shift in the step function. Since a 0.5 LSB error is also allowed in linearity error, an input of 0.000 volts can convert to \$00 or \$01.

THE M6805 FAMILY A/D CONVERTER

Figure 2 is a block diagram of the successive approximation A/D converter implemented in an M6805 Family MCU. There are two unique registers which are addressed under program control. They are used to select a channel, start a conversion, and read the 8-bit result after a conversion has been completed.

There are four separate external input channels which can be individually coupled to the comparator input by writing to the three lower order bits in the A/D control register (ACR). There are also four internal input channels which are coupled to the V_{RH}/V_{RL} resistor chain and can be used as calibration references. Also shown in Figure 2 are the address locations for the external and internal channels.

The converter operates continuously producing a new result every 30 machine cycles. Bit 7 of the ACR flags the user when a conversion has been completed and the digital value of the analog input can be read from the A/D result register (ARR).

When the aperture window closes, the successive approximation register (SAR) addresses the D/A converter (DAC) with a binary word value of \$80. Thus, only the most significant bit (MSB) of the word is set. This value causes the DAC to generate an analog voltage equal to the midway value of the conversion range.

At the comparator, the DAC output is compared with the input voltage stored in the sample and hold capacitor. The comparator then signals the SAR whether the DAC value is greater than the sampled value. If the sampled voltage is greater than the DAC voltage, the SAR binary value is increased to \$C0 by setting the bit that is adjacent to the already set MSB. In turn, this increased value increases the DAC output voltage. If the input voltage is less than the DAC voltage, the SAR MSB is cleared and the bit adjacent to the MSB is set, resulting in less DAC voltage to the comparator. This binary-search method continues until all eight SAR bits have been determined.

At the end of the conversion, the SAR value is transferred to the A/D result register (ARR). This binary value is the approximate digital representation of the analog input within one LSB of error. A conversion complete flag is set in the ACR indicating to the user program that a new result may be retrieved.

The equivalent analog input circuit for one channel entering the multiplexer is illustrated in Figure 3. The worst case instantaneous current draw is at the beginning of the sample period. Since the RC time constant of the sample and hold circuit is approximately 500 nanoseconds, the five machine-cycle sample time is more than sufficient to charge the sample and hold capacitor.

The average analog channel input current is less than one microampere. Thus, no additional buffering of external transducer signals is required. The analog channel input impedance should not be greater than 1K ohms to prevent capacitor leakage currents from producing unwanted voltage drops.

A/D CONVERTER SELF-TEST

Designing a test capability into a system simplifies maintenance, decreases the need for costly test equipment in the field, and provides an excellent way to learn the system characteristics. Using a minimum amount of hardware an M6805 Family MCU can test its own A/D converter in less than one second for 8-bit accuracy. The objective of this example is to self-adjust the A/D converter under test for full scale error within two LSB and then to test for a correct conversion at the center of each step. The MCU used in this example is an MC68705R3.

The hardware schematic for the A/D converter self-test shown in Figure 4 consists of a voltage stimulus, a programmable voltage reference, and a program storage unit. A 12-bit resolution monolithic current output DAC (U2) presents the voltages to the MCU A/D converter input. An operational amplifier converts the DAC current output to a voltage and is fed into the device under test (DUT).

In order to download the test program to the MCU RAM, a 12-bit binary counter (U7) is used to address an EPROM (U3). Another 12-bit counter (U1) addresses the DAC in order to ramp the test voltages to the DUT. The two 12-bit counters are used to minimize the MCU pins needed for calibration in user systems. A variable-precision voltage reference (U5) and a quad analog switch (U6) are used to derive the voltage reference to the DUT.

BOOTSTRAP PROGRAM (BOOT)

To minimize the amount of ROM occupied by the self-test program, a 28-byte bootstrap loader routine is suggested. The bootstrap loader is a short program which is executed

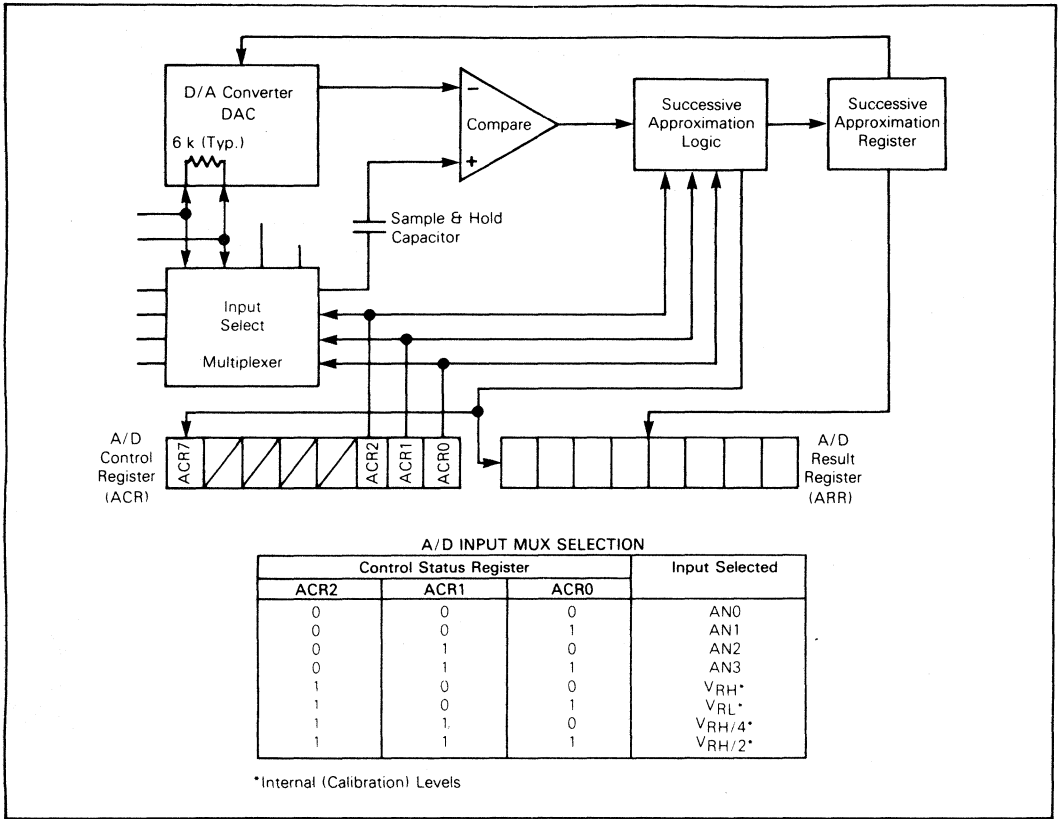


FIGURE 2 — A/D Converter Block Diagram and Input Select Code

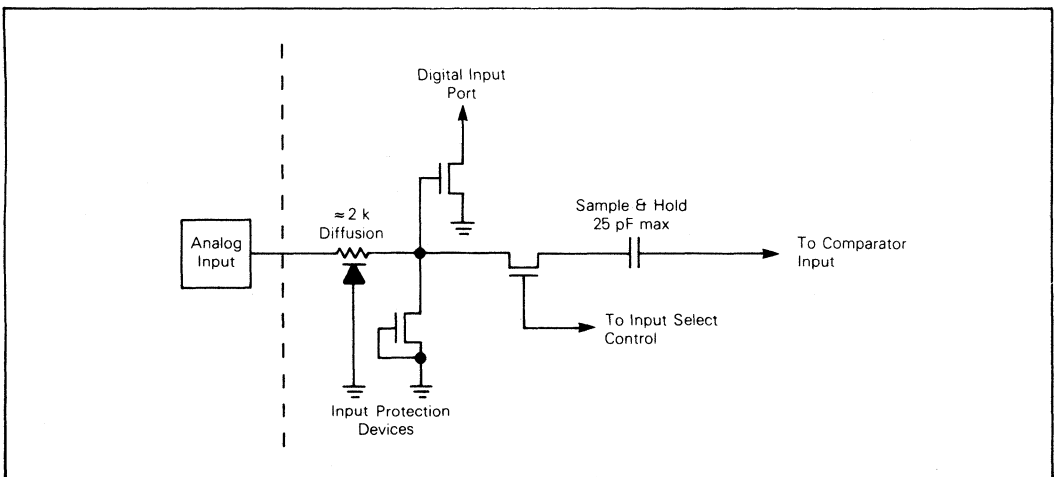


FIGURE 3 — Analog Channel Input Circuit

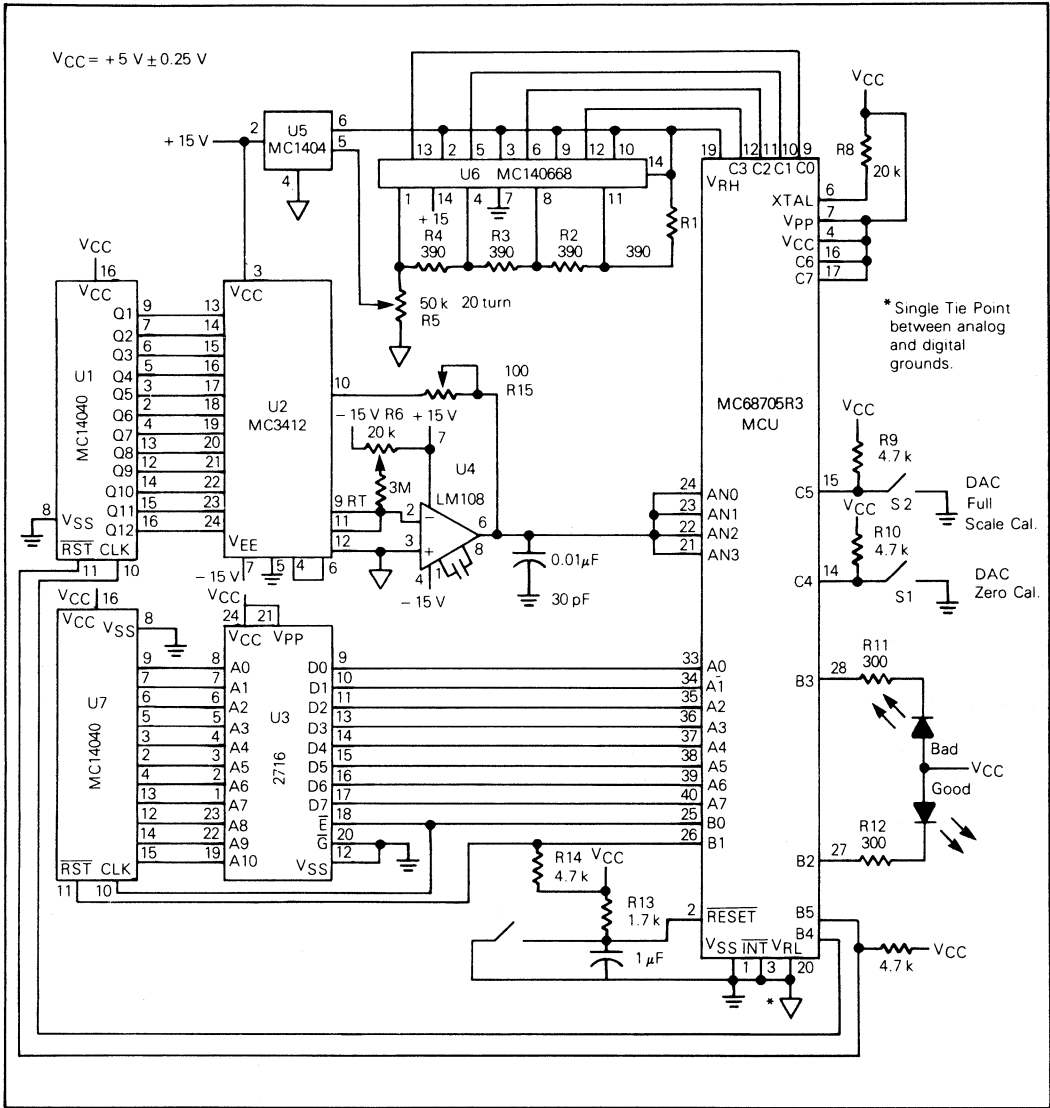


FIGURE 4 — On-Chip A/D Converter Self-Test and Self-Calibration-Schematic Diagram

when a particular condition is met after the MCU comes out of reset. The bootstrap program loads a short test program from an external memory device into on-chip RAM and executes the program.

The condition selected in order to execute the bootstrap loader program is a logic low level on the interrupt pin when coming out of reset. When the interrupt pin has a logic high level, the user main program is entered. The address of the label "load" in the bootstrap program is used in the self-test program to jump back to load new routines. The bootstrap loader approach is not limited to the A/D converter self-test program. It can be used to examine or test other system features.

SELF-TEST SOFTWARE (A/DTST)

The self-test software consists of three sequential loads. This requires the bootstrap loader to jump to RAM after downloading, execute a program from RAM, then return to the bootstrap loader to download the next routine. When the MCU comes out of reset, the interrupt pin is held low in order to enter the bootstrap loader routine.

The bootstrap program uses port B bit 1 to clear the counter and port B bit 0 to increment the count on a negative transition. The program is fed into port A and stored to RAM.

The index register is used as the RAM pointer and is tested for a negative value after each RAM byte load. Since the negative status bit in the condition code register is set when the accumulator holds a value between \$80 and \$FF, the loading routine ends when the index register reaches a value of \$80. The Figure 5 flow chart illustrates the interaction between the bootstrap loader routine (BOOT) and the A/D converter self-test routines (A/DTST).

12-Bit DAC Calibration

The first routine is used to calibrate the 12-bit DAC (U2) and the precision voltage reference (U5). The ideal V_{RH} level selected is 5.120 volts for 0.020-volt steps. Therefore, the first step is to preset U5 with R5 to 5.160 volts (2 LSB above the ideal V_{RH}) while the MCU is held in reset.

The next step is to calibrate the 12-bit DAC zero output for 0.000 volts and the full scale output for 5.100 volts. This routine is executed only if port C bits 4 and 5 are held low. This is done by closing S1 (zero calibrate select) and S2 (full scale calibrate select) before bringing the MCU under test out of reset.

The 12-bit DAC zero offset adjustment is set by varying R6 while monitoring the DAC output for a value of 0.000 volts. Opening the bit 4 switch causes the DAC to be incremented to \$FF0 and held there for the full-scale error adjustment provided the bit 5 switch is closed.

The DAC should then be calibrated to 5.100 volts by varying R15. Opening the bit 5 switch causes the calibration routine to terminate, and the second routine is then downloaded. The DAC calibration routine is normally done once after initial power-up and is skipped if the port C switches are open.

On-Line A/D Calibration

The second routine in the self-test exercise is the A/D converter automatic full-scale error adjust. The DAC is incremented to \$FE8 (5.090 V), which is the A/D converter switch point between \$FE and \$FF. All analog switches on U6 are off at this time so the voltage reference to the DUT is 5.160 volts (V_{RH} ideal + 2 LSB). The A/D converter should

output a value of \$FE since the analog input switch point from \$FE to \$FF would be 5.140 volts using this reference value. Table 1 represents the voltage inputs that cause a switch in the digital output of the A/D converter when using a $V_{RH} + V_{RL}$ of 5.120 V.

The analog switch is controlled by the four lower order bits of port C. As port C is incremented, it causes a decrease of 10 millivolts on the output of the voltage reference, U5. An A/D conversion is done after each port C increment to check for a value of \$FF.

When the A/D converter recognizes the input voltage as an \$FF, port C is left constant at that value for the next test. This self-adjusting routine compensates for the variation of full-scale error from one device to another and eliminates having to manually adjust each unit before test.

A/D Linearity Test

The third and last routine is the actual A/D converter voltage ramp test. The 12-bit DAC is cleared then ramped in 20-millivolt increments to check the center of each A/D step for a correct conversion. At the end of the 256 voltage steps, the program outputs a logic low level on the port B bit 2 to turn on an LED, indicating a good device.

If at any point during the 256 voltage steps a conversion is incorrect, testing is stopped and the program outputs a logic low on port B bit 3 to signal that the A/D converter failed the test. The A/D converter linearity is thus confirmed to be within one LSB (20 millivolts) of the proper reading.

TEMPERATURE SENSOR CONVERSION

Monitoring temperature for display or control is a key factor for energy conservation or process control in closed loop systems. The objective in this portion of the application note is to demonstrate how to manipulate the converted analog signal from a temperature sensor for display.

By monitoring the base-emitter voltage variations on the Motorola MTS-102 silicon temperature sensor, the MCU converts that analog information into an equivalent digital value in degrees fahrenheit and displays it on three 7-segment displays.

Many tasks can be performed after the conversion is complete. One typical application for this circuit is a ceiling fan control to increase room air flow for a period of time before turning on the main central air conditioning unit. If the temperature does not decrease after a timeout period, then the central air is turned on for a period of time.

The Figure 6 schematic shows how the sensor voltage is amplified to give 20-millivolt steps per degree fahrenheit. A dual differential amplifier (U2) buffers the sensor voltage and then inverts and amplifies the signal before entering the A/D converter on the MCU. An amplifier gain of eight produces 10 millivolts per degree fahrenheit, and a gain of 4.444 produces 10 millivolts per degree centigrade.

In order for the A/D converter to recognize one degree steps, the gain must be doubled. The maximum output voltage on the differential amplifier is approximately 3.8 volts using a V_{CC} of 5 volts. Therefore, the temperature sensing range is from -40 degrees to +140 degrees fahrenheit.

The Monsanto LED displays have a common anode and are driven directly by the MCU port B. To update the displays, port C bits 0, 1 and 2 enable each display sequentially every 2048 machine cycles. The regulated voltage from U1 is tied to the MCU V_{CC} as well as to the displays and is used ratiometrically with respect to the V_{RH} pin and the

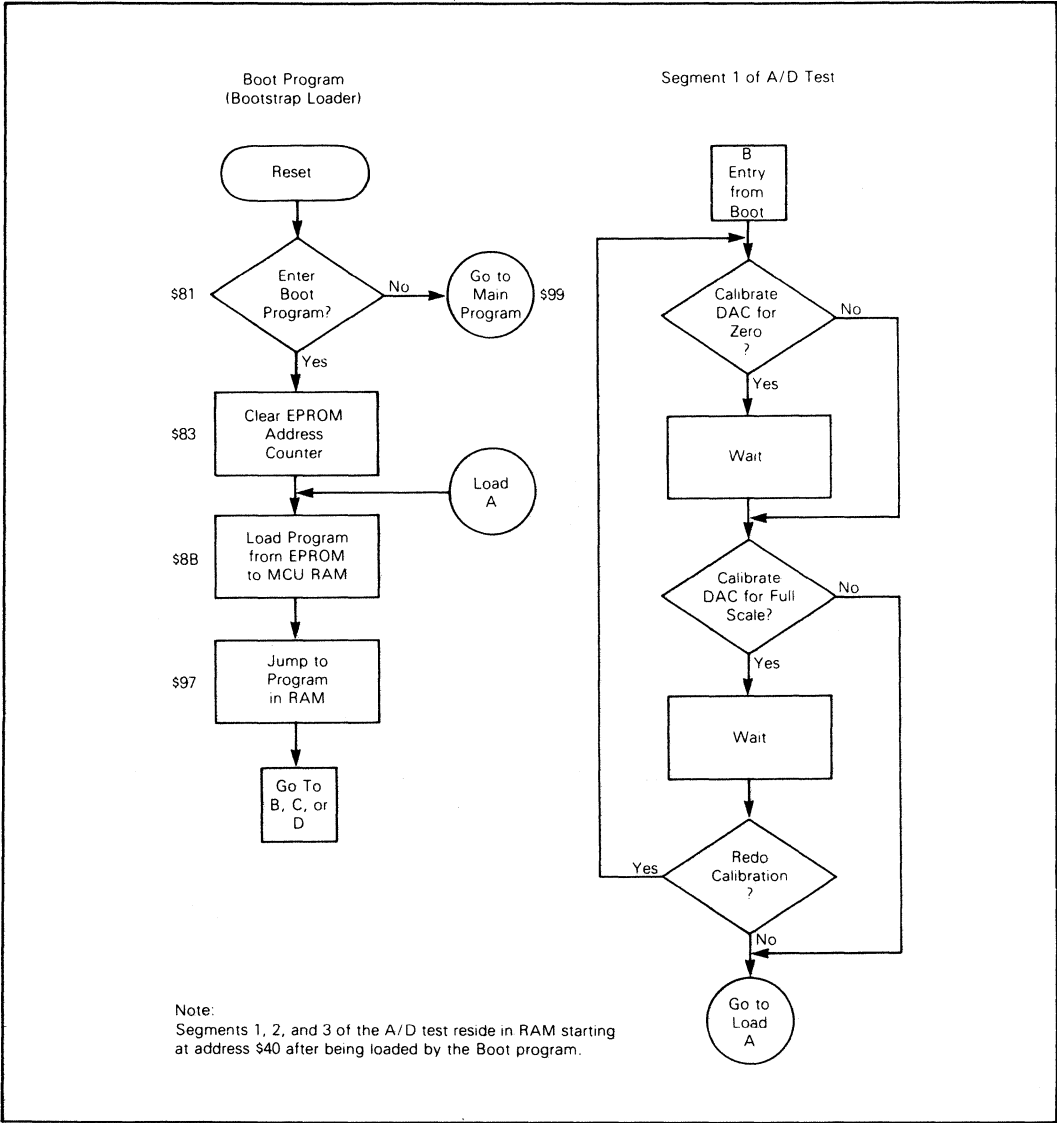


FIGURE 5 – Boot and A/D TST Program Flow Chart (Sheet 1 of 2)

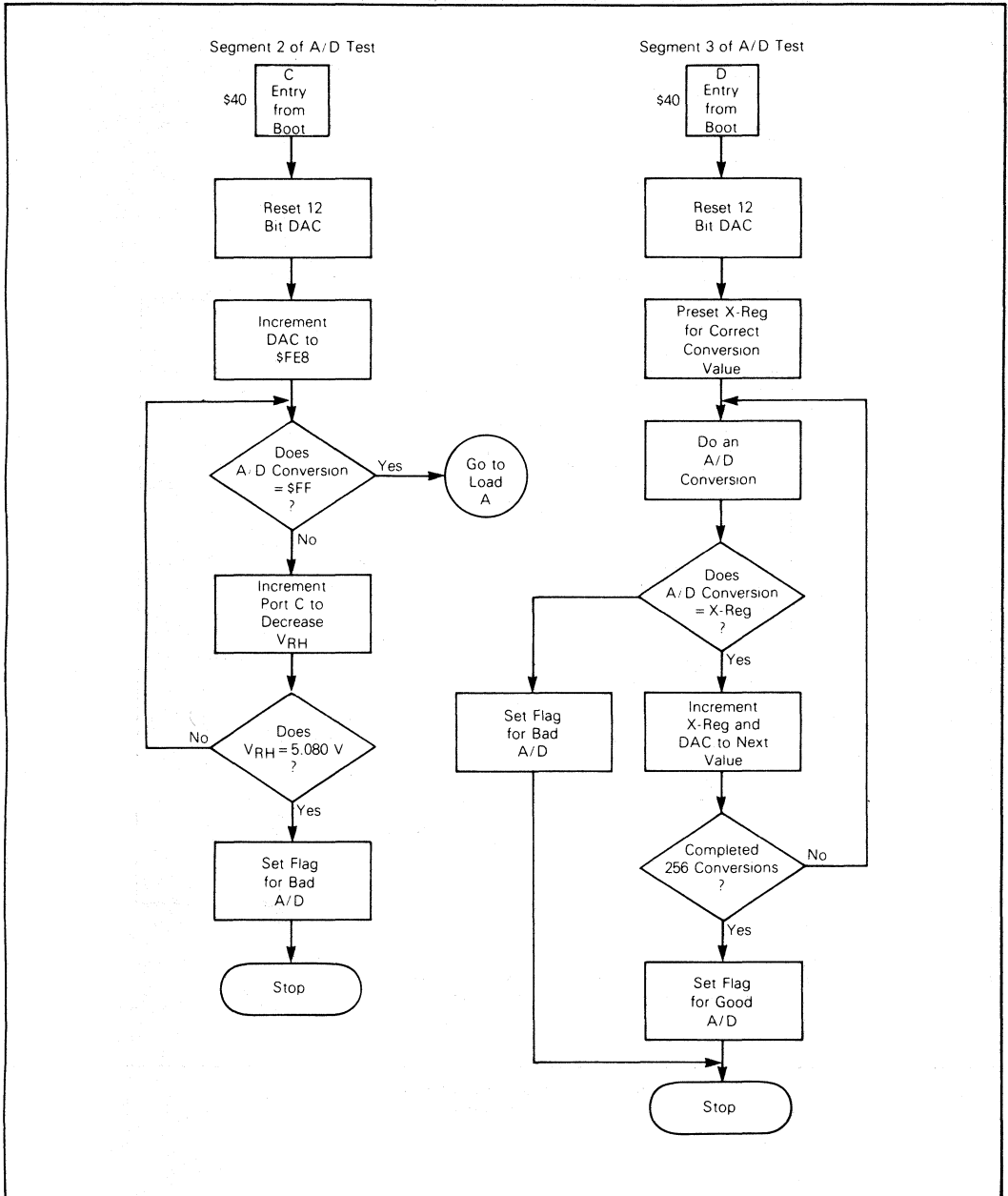


FIGURE 5 — Boot and A/D TST Program Flow Chart
(Sheet 2 of 2)

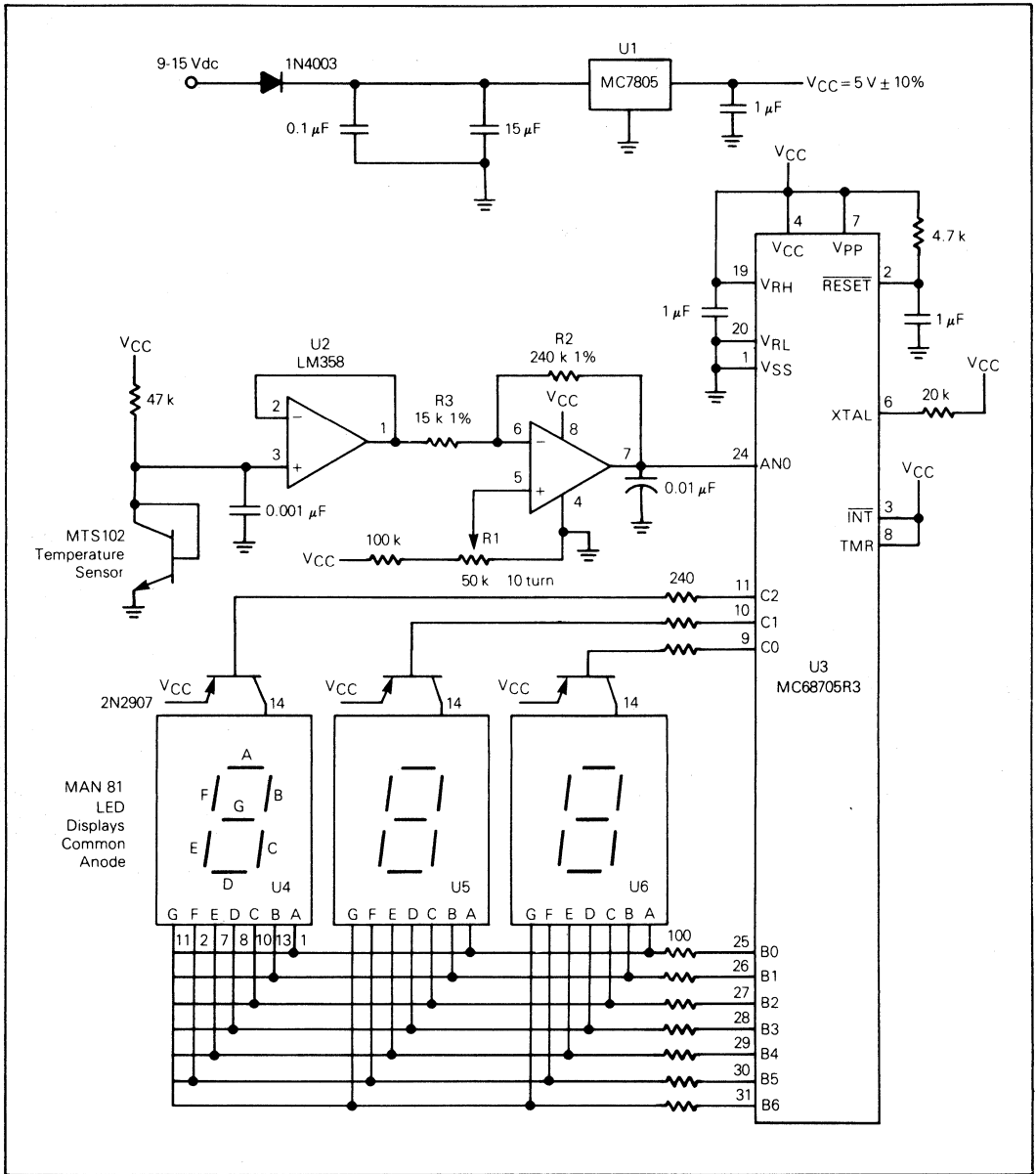


FIGURE 6 — Temperature Sensing A/D Demonstration-Schematic Diagram

sensor supply. Thus, any variation on the amplifiers also occurs on the V_{RH} pin.

Since the MC68705R3 has an EPROM erasing window, it is necessary to cover it after programming so that the A/D converter will maintain its 8-bit accuracy. It is also recommended that the MTS102 sensor leads be moisture-sealed with epoxy. About 12 to 18 inches of twisted, stranded, 22-gauge wire was used to connect the sensor to the amplifier input.

The Figure 7 flow chart describes the sequence of events in the temperature sensor program (TMPSNS). After port initialization, the timer is enabled to periodically update the 7-segment displays. An A/D conversion is performed and the result is converted to binary-coded-decimal (BCD) format for the display routine.

Zero degrees fahrenheit is equal to a conversion of \$30 so the A/D conversion result must be offset before converting to a BCD number. Since the displays are updated during a timer interrupt, the MCU goes into a wait loop before doing

another A/D conversion. This is a good entrypoint for appending new tasks to the basic program such as time of day or controlling a heater or air conditioner.

Calibration of the temperature sensor is performed by varying the 50 kilohms potentiometer (R1) on the differential amplifier for a reading of 32 degrees after a piece of ice has been placed on the sensor for approximately one minute. Refer to the MTS-102 specification for further details.

SUMMARY

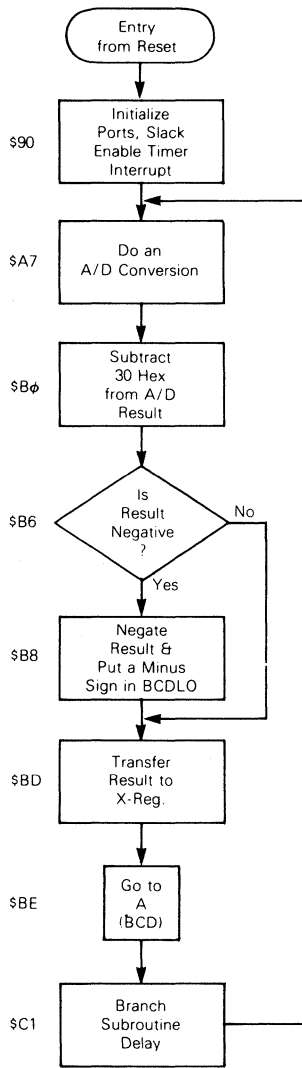
The terminology used to describe an A/D converter characteristics has been discussed and related to the electrical characteristics in the M6805 Family A/D converter specifications. A circuit and program which tests the on-chip A/D converter was also discussed to familiarize the reader with a method of calibrating the A/D converter. A typical temperature sensor application was covered with the intent of showing the sequence of events that take place when converting the A/D result to a displayable value.

TABLE 1 — Voltage Versus Output for 8-Bit A/D Converter
 $V_{RH} = 5.120 \text{ V}$ $V_{RL} = 0.000 \text{ V}$

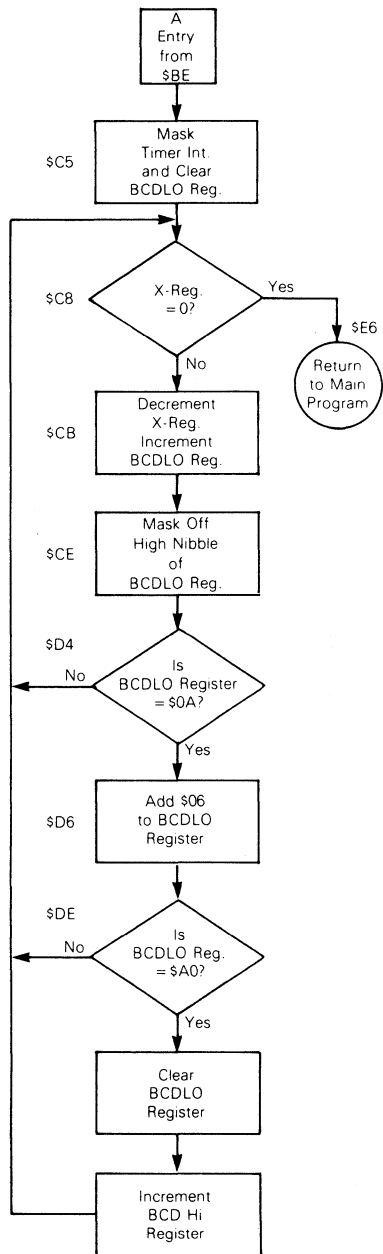
Hi/Lo	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
*0	0.010	0.330	0.650	0.970	1.290	1.610	1.930	2.250	2.570	2.890	3.210	3.530	3.850	5.170	4.490	4.810
1	0.030	0.350	0.670	0.990	1.310	1.630	1.950	2.270	2.590	2.910	3.230	3.550	3.870	4.190	4.510	4.830
2	0.050	0.370	0.690	1.010	1.330	1.650	1.970	2.290	2.610	2.930	3.250	3.570	3.890	4.210	4.530	4.850
3	0.070	0.390	0.710	1.030	1.350	1.670	1.990	2.310	2.630	2.950	3.270	3.590	3.910	4.230	4.550	4.870
4	0.090	0.410	0.730	1.050	1.370	1.690	2.010	2.330	2.650	2.970	3.290	3.610	3.930	4.250	4.570	4.890
5	0.110	0.430	0.750	1.070	1.390	1.710	2.030	2.350	2.670	2.990	3.310	3.630	3.950	4.270	4.590	4.910
6	0.130	0.450	0.770	1.090	1.410	1.730	2.050	2.370	2.690	3.010	3.330	3.650	3.970	4.290	4.610	4.930
7	0.150	0.470	0.790	1.110	1.430	1.750	2.070	2.390	2.710	3.030	3.350	3.670	3.990	4.310	4.630	4.950
8	0.170	0.490	0.810	1.130	1.450	1.770	2.090	2.410	2.730	3.050	3.370	3.690	4.010	4.330	4.650	4.970
9	0.190	0.510	0.830	1.150	1.470	1.790	2.110	2.430	2.750	3.070	3.390	3.710	4.030	4.350	4.670	4.990
A	0.210	0.530	0.850	1.170	1.490	1.810	2.130	2.450	2.770	3.090	3.410	3.730	4.050	4.370	4.690	5.010
B	0.230	0.550	0.870	1.190	1.510	1.830	2.150	2.470	2.790	3.110	3.430	3.750	4.070	4.390	4.710	5.030
C	0.250	0.570	0.890	1.210	1.530	1.850	2.170	2.490	2.810	3.130	3.450	3.770	4.090	4.410	4.730	5.050
D	0.270	0.590	0.910	1.230	1.550	1.870	2.190	2.510	2.830	3.150	3.470	3.790	4.110	4.430	4.750	5.070
E	0.290	0.610	0.930	1.250	1.570	1.890	2.210	2.530	2.850	3.170	3.490	3.810	4.130	4.450	4.770	5.090
F	0.310	0.630	0.950	1.270	1.590	1.910	2.230	2.550	2.870	3.190	3.510	3.830	4.150	4.470	4.790	5.110*

NOTES: (1) Voltages in the chart represent the switch point between steps. For example, 0.010 volts is the switch point between \$00 and \$01.
 (2) Due to quantizing error, the center of each step is the chart value minus 10 millivolts. The last step does not overflow.
 (3) The first step (*) equals 10 millivolts; the last step (*) equals 30 millivolts.

Main Program Flow



Binary to BCD Subroutine Conversion



NOTE:

When the timer interrupts the main program flow, go to the timer interrupt routine (next page). When the timer interrupt routine is finished, the MCU will automatically return to the next operation in the main program flow.

FIGURE 7 – Temperature Sensor (TEMPSNS) Program Flow Chart (Sheet 1 of 2)

TIMER INTERRUPT ROUTINE
(BCD to 7-Segment Display)

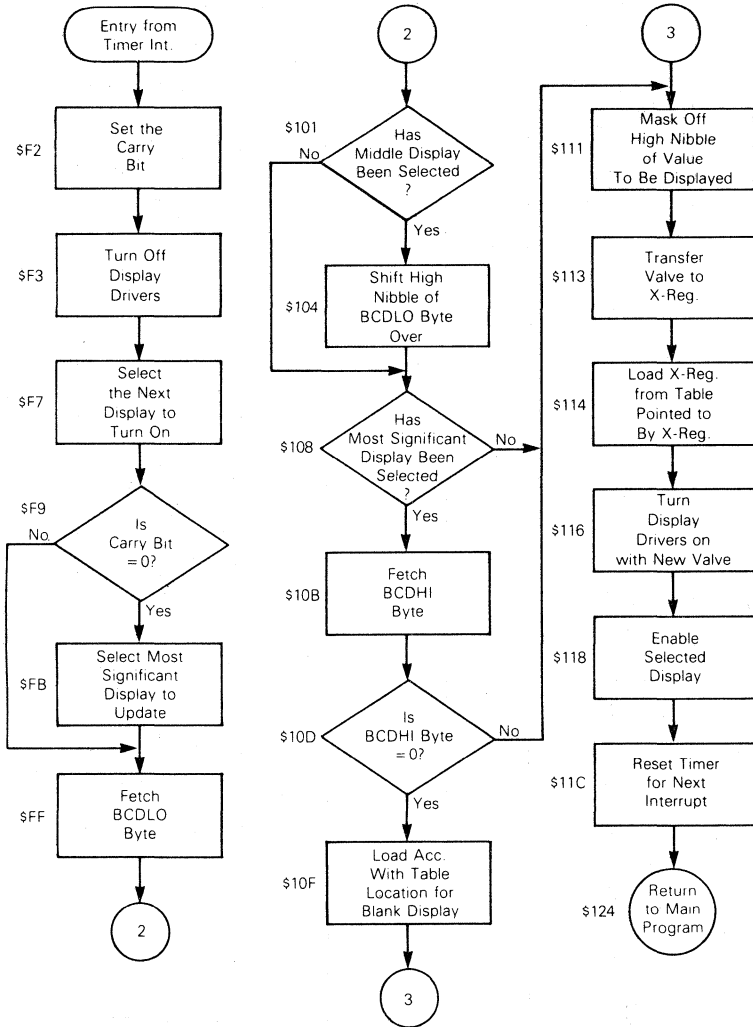


FIGURE 7 — Temperature Sensor (TEMPSNS) Program Flow Chart
(Sheet 2 of 2)

PAGE 001 BOOT .SA:1 BOOT

```
00001          NAM    BOOT
00002          OPT    LLEN=100
00003
00004
00005          *****LABEL DEFINITION*****
00006
00007          0000    A PORTA EQU 0
00008          0001    A PORTB EQU 1
00009          0005    A DDRB EQU 5
00010
00011          *****GENERAL PURPOSE BOOTSTRAP LOADER ROUTINE*****
00012
00013A 0080          ORG    $80      NORMALLY THE START LOCATION WILL BE
00014A 0080 9C          RSP      THE USER RESET VECTOR.
00015A 0081 2F 16    0099    BIH    EXIT    TEST INTERRUPT PIN FOR BOOTSTRAP ENTRY.
00016A 0083 A6 03          A      LDA    #503    MAKE PORTB BITS 0 AND 1 OUTPUTS.
00017A 0085 B7 01          A      STA    PORTB   RESET 12-BIT COUNTER.
00018A 0087 B7 05          A      STA    DDRB
00019A 0089 13 01          A      BCLR   1,PORTB  ENABLE 12-BIT COUNTER.
00020A 008B AE 40          A LOAD  LDX    #540
00021A 008D B6 00          A      LDA    PORTA   PORTA IS INPUT WHEN COMING OUT OF RESET.
00022A 008F F7          STA    ,X      STORE CONTENTS OF PORTA TO RAM.
00023A 0090 11 01          A      BCLR   0,PORTB  INCREMENT 12-BIT COUNTER.
00024A 0092 10 01          A      BSET   0,PORTB
00025A 0094 5C          INCX
00026A 0095 2A F6    008D    BPL    LOAD+2   CONTINUE LOADING TILL X REG=80.
00027A 0097 BC 40          A      JMP    $40     JUMP TO RAM AND EXECUTE PROGRAM.
00028A 0099 CC 0100    A EXIT  JMP    $100    JUMP TO USER MAIN PROGRAM OR EXECUTE FROM HERE.
00029          END
TOTAL ERRORS 00000--00000
```

PAGE 001 SLFTST .SA:1 A/DTST

```
00001          NAM      A/DTST
00002          OPT      LLEN=100
00003A 0000     ORG      S00
00004          *****LABEL DEFINITION*****
00005          0000     A PORTA EQU 0
00006          0001     A PORTB EQU 1
00007          0002     A PORTC EQU 2
00008          0003     A PORTD EQU 3
00009          0004     A DDRA EQU 4
00010          0005     A DDRB EQU 5
00011          0006     A DDRC EQU 6
00012          0007     A NC7 EQU 7
00013          0008     A TDR EQU 8
00014          0009     A TCR EQU 9
00015          000A     A MSR EQU 10
00016          000B     A PCR EQU 11
00017          000C     A NC2 EQU 12
00018          000D     A NC3 EQU 13
00019          000E     A ACR EQU 14
00020          000F     A ARR EQU 15
00021          008B     A LOAD EQU S8B      EXAMPLE LOCATION OF BOOTSTRAP LOADER
00022
00023          *****SEGMENT 1 DAC CALIBRATION ROUTINE*****
00024
00025A 0000 9C          RSP          INITIALIZE STACK
00026A 0001 AE FD      A          LDX          #SFD          INITIALIZE PORTB BUT INSURE
00027A 0003 BF 01      A          STX          PORTB        BITS 0 AND 1 DO NOT CHANGE
00028A 0005 3F 02      A          CLR          PORTC
00029A 0007 AE 0F      A          LDX          #SOF          PORTC 0-3=OUTPUTS
00030A 0009 BF 06      A          STX          DDRC
00031A 000B AE 3F      A          LDX          #S3F          PORTB 0-5=OUTPUTS
00032A 000D BF 05      A          STX          DDRB
00033A 000F 1A 01      A CAL   BSET          5,PORTB    CLEAR DAC 12-BIT COUNTER
00034A 0011 1B 01      A          BCLR          5,PORTB
00035A 0013 09 02 FD 0013 BRCLR          4,PORTC,* STOP HERE AND CAL. 0.000V IF PC4=0
00036A 0016 0A 02 14 002D BRSET          5,PORTC,END1 EXIT CAL. ROUTINE IF PC5=1
00037A 0019 AE FF      A          LDX          #SFF
00038A 001B A6 10      A          LDA          #S10
00039A 001D 19 01      A          BCLR          4,PORTB    TOGGLE DAC 12-BIT COUNTER
00040A 001F 18 01      A          BSET          4,PORTB    16 X 255 TIMES
00041A 0021 4A          DECA
00042A 0022 26 F9      001D    BNE          *-5
00043A 0024 5A          DECX
00044A 0025 26 F4      001B    BNE          *-10
00045A 0027 0B 02 FD 0027 BRCLR          5,PORTC,* STOP HERE AND CAL. 5.100V IF PC5=0
00046A 002A 09 02 E2 000F BRCLR          4,PORTC,CAL RECAL. ZERO OFFSET?
00047A 002D BC 8B      A END1    JMP          LOAD          RETURN TO BOOTSTRAP TO LOAD NEXT ROUTINE
00048
```

```

00050
00051          ****SEGMENT 2  FULL SCALE ERROR ADJUST*****
00052
00053
00054A 0040          ORG  $40          SECOND ROUTINE MUST START 64 BYTES LATER
00055
00056A 0040 1A 01      A      BSET  5,PORTB  RESET DAC 12-BIT COUNTER
00057A 0042 1B 01      A      BCLR  5,PORTB
00058A 0044 AE 08      A      LDX   #S08
00059A 0046 19 01      A      BCLR  4,PORTB
00060A 0048 18 01      A      BSET  4,PORTB
00061A 004A 5A          A      DECX
00062A 004B 26 F9      0046   BNE   *-5
00063A 004D AE FE      A      LDX   #SFE
00064A 004F A6 10      A  LOOP1 LDA   #S10  INCREMENT DAC 12-BIT COUNTER
00065A 0051 19 01      A      BCLR  4,PORTB  UP TO FE8 (5.090V)
00066A 0053 18 01      A      BSET  4,PORTB
00067A 0055 4A          A      DECA
00068A 0056 26 F9      0051   BNE   *-5
00069A 0058 5A          A      DECX
00070A 0059 26 F4      004F   BNE   LOOP1
00071A 005B 5A          A      DECX
00072A 005C 3F 0E      A  ADJUST CLR   ACR   MAKE X-REG=FF
00073A 005E B6 0E      A      LDA   ACR   CLEAR CONVERSION COMPLETE FLAG
00074A 0060 2A FC      005E   BPL   *-2      WAIT FOR CONVERSION (30CYCLES)
00075A 0062 B3 0F      A      CPX   ARR   TEST RESULT FOR FF
00076A 0064 27 0A      0070   BEQ   END2   AND EXIT ADJUST ROUTINE WHEN ARR=FF
00077A 0066 3C 02      A      INC   PORTC ELSE DECREMENT VRH VOLTAGE
00078A 0068 B3 02      A      CPX   PORTC
00079A 006A 26 F0      005C   BNE   ADJUST  CONTINUE SEARCH TILL PORTC=0F
00080A 006C 17 01      A  FAIL1 BCLR  3,PORTB IF UNSUCCESSFUL THEN STOP
00081A 006E 20 FE      006E   BRA   *      AFTER SETTING FLAG
00082A 0070 3A 02      A  END2  DEC   PORTC  READJUST FOR LAST PORTC INCREMENT
00083A 0072 BC 8B      A      JMP   LOAD   RETURN TO BOOTSTRAP LOADER
00084

```

PAGE 003 SLFTST .SA:1 A/DTST

```
00086
00087          *****SEGMENT #S3 A/D TEST ROUTINE*****
00088
00089A 0080          ORG      S80
00090
00091A 0080 AE 01     A       LDX   #S01   INITIALIZE X-REG AS COUNTER
00092A 0082 1A 01     A       BSET   5,PORTB  RESET DAC 12-BIT COUNTER
00093A 0084 1B 01     A       BCLR   5,PORTB
00094A 0086 3F 0E     A       CLR    ACR    CLEAR CONVERSION FLAG
00095A 0088 B6 0E     A       LDA    ACR
00096A 008A 2A FC     0088    BPL    *-2    WAIT FOR END OF CONVERSION
00097A 008C B3 0F     A       CPX    ARR
00098A 008E A1 01     A       CMP    #S01   TEST FOR ZERO INPUT READING OF
00099A 0090 25 1A     00AC    BLO    FAIL2   00 OR 01 AND STOP IF ARR-REG > X-REG
00100A 0092 A6 10     A LOOP2  LDA    #S10   INCREMENT DAC BY 16 STEPS
00101A 0094 19 01     A       BCLR   4,PORTB  TO STAY IN CENTER OF 8-BIT STEP
00102A 0096 18 01     A       BSET   4,PORTB
00103A 0098 4A        DECA
00104A 0099 26 F9     0094    BNE    *-5
00105A 009B 3F 0E     A       CLR    ACR    CLEAR CONVERSION FLAG AND
00106A 009D B6 0E     A       LDA    ACR    WAIT FOR END OF CONVERSION
00107A 009F 2A FC     009D    BPL    *-2
00108A 00A1 B3 0F     A       CPX    ARR    TEST FOR CORRECT CONVERSION AND
00109A 00A3 26 07     00AC    BNE    FAIL2   STOP IF ARR-REG. NOT EQUAL TO X-REG.
00110A 00A5 5C        INCX
00111A 00A6 26 EA     0092    BNE    LOOP2   CONTINUE TESTING TILL X-REG.=00
00112A 00A8 15 01     A       BCLR   2,PORTB  TURN LED ON FOR TEST PASS
00113A 00AA 20 FE     00AA    BRA    *       AND STOP
00114A 00AC 17 01     A FAIL2 BCLR   3,PORTB  TURN LED ON FOR TEST FAIL
00115A 00AE 20 FE     00AE    BRA    *       AND STOP
00116          END
TOTAL ERRORS 00000--00000
```

PAGE 001 TSPNS .SA:1 TEMPSN

```
00001          NAM    TEMPSNS
00002          OPT    LLEN=120
00003
00004          *****TEMPERATURE DISPLAY PROGRAM*****
00005
00006          *****REGISTER ADDRESS DEFINITION*****
00007
00008          0001    A PORTB EQU    1
00009          0002    A PORTC EQU    2
00010          0005    A DDRB  EQU    5
00011          0006    A DDRC  EQU    6
00012          0008    A TDR   EQU    8
00013          0009    A TCR   EQU    9
00014          000E    A ACR   EQU   14
00015          000F    A ARR   EQU   15
00016          0040    A BCDHI EQU   $40
00017          0041    A BCDLO EQU   $41
00018          0042    A SEGMNT EQU   $42
00019
00020A 0080          ORG    $80
00021
00022          *****7-SEGMENT DISPLAY LOOKUP TABLE*****
00023
00024          0080    A SEVSEG EQU    *
00025
00026A 0080    40    A      FCB    $01000000 0
00027A 0081    79    A      FCB    $01111001 1
00028A 0082    24    A      FCB    $00100100 2
00029A 0083    30    A      FCB    $00110000 3
00030A 0084    19    A      FCB    $00011001 4
00031A 0085    12    A      FCB    $00010010 5
00032A 0086    02    A      FCB    $00000010 6
00033A 0087    78    A      FCB    $01111000 7
00034A 0088    00    A      FCB    $00000000 8
00035A 0089    18    A      FCB    $00011000 9
00036A 008A    3F    A      FCB    $00111111 -
00037A 008B    7F    A      FCB    $01111111 BLANK
00038
00039
00040A 0090          ORG    $90      PROGRAM START
00041
00042A 0090 9C          START RSP
00043A 0091 A6 F7      A      LDA    #SF7    INITIALIZE PORTC TO SELECT THE
00044A 0093 B7 02      A      STA    PORTC   MOST SIGNIFICANT DISPLAY FIRST
00045A 0095 B7 42      A      STA    SEGMNT
00046A 0097 A6 07      A      LDA    #S07    MAKE PORTC0-2 OUTPUTS
00047A 0099 B7 06      A      STA    DDRC
00048A 009B A6 FF      A      LDA    #SFF    INITIALIZE PORTB TO
00049A 009D B7 01      A      STA    PORTB   TURN OFF ALL DISPLAYS
00050A 009F B7 05      A      STA    DDRB
00051A 00A1 B7 08      A      STA    TDR
00052A 00A3 A6 0F      A      LDA    #S0F    PRESET TIMER FOR PRESCALE OF 128
00053A 00A5 B7 09      A      STA    TCR     AND UNMASK TIMER INTERRUPT
```

PAGE 002 TMSNS .SA:1 TEMPSN

```
00055A 00A7 9B      ADC  SEI
00056A 00A8 3F 0E   A    CLR  ACR  CLEAR CONVERSION COMPLETE FLAG
00057A 00AA B6 0E   A    LDA  ACR  AND SELECT CHANNEL 0
00058A 00AC 2A FC   00AA BPL  *-2  CHECK FOR A CONVERSION COMPLETE
00059A 00AE B6 0F   A    LDA  ARR  ADJUST THE RESULT REGISTER
00060A 00B0 A2 30   A    SBC  #S30 SO ZERO DEGREES F=30 HEX
00061A 00B2 3F 40   A    CLR  BCDHI
00062A 00B4 A1 CF   A    CMP  #SCF
00063A 00B6 25 05   00BD BLO  CONVRT CHECK FOR NEGATIVE NUMBER
00064A 00B8 40      NEGA
00065A 00B9 AE 0A   A    LDX  #S0A STORE A MINUS SIGN IN BCDHI
00066A 00BB BF 40   A    STX  BCDHI
00067A 00BD 97      CONVRT TAX  LOAD X REG WITH VALUE
00068A 00BE AD 05   00C5 BSR  BCD  AND BRANCH TO CONVERSION ROUTINE
00069      *
00070      * NOTE: MANY OTHER TASKS CAN BE PERFORMED BEFORE
00071      * DOING ANOTHER CONVERSION.
00072      *
00073A 00C0 9A      CLI
00074A 00C1 AD 24   00E7 BSR  DELAY  KILL TIME BEFORE DOING ANOTHER CONVERSION
00075A 00C3 20 E2   00A7 BRA  ADC  GO CHECK THE TEMPERATURE AGAIN
00076
00077
00078      *****BINARY TO BCD ROUTINE*****
00079      *
00080      * THE X REGISTER ENTERS WITH THE BINARY VALUE
00081      * AND EXITS WITH 00. THE CONVERTED BINARY TO
00082      * BCD VALUES ARE PLACED IN RAM FOR THE DISPLAY ROUTINE.
00083      *
00084
00085      00C5  A BCD  EQU  *
00086A 00C5 9B      SEI
00087A 00C6 3F 41   A    CLR  BCDLO  BCD LOW BYTE INITIALIZE
00088A 00C8 5D      LOOP  TSTX
00089A 00C9 27 1B   00E6 BEQ  EXIT  LOOP ENDS WHEN X REG=00
00090A 00CB 5A      DECX
00091A 00CC 3C 41   A    INC  BCDLO  MOVE X CONTENTS TO LOW BYTE BCD
00092A 00CE A6 0F   A    LDA  #S0F  MASK OFF HIGH NYBBLE OF LOW BYTE BCD
00093A 00D0 B4 41   A    AND  BCDLO
00094A 00D2 A1 0A   A    CMP  #S0A  CHECK IF LOW BYTE BCD NEEDS ADJUSTING
00095A 00D4 26 F2   00C8 BNE  LOOP
00096A 00D6 B6 41   A    LDA  BCDLO
00097A 00D8 AB 06   A    ADD  #S06  ADJUST FOR A BCD VALUE
00098A 00DA B7 41   A    STA  BCDLO
00099A 00DC A1 A0   A    CMP  #SA0  CHECK IF HIGH BYTE BCD NEEDS ADJUSTING
00100A 00DE 26 E8   00C8 BNE  LOOP  GO DO IT AGAIN
00101A 00E0 3F 41   A    CLR  BCDLO  CLEAR LOW BYTE BCD
00102A 00E2 3C 40   A    INC  BCDHI  AND INCREMENT HIGH BYTE BCD
00103A 00E4 20 E2   00C8 BRA  LOOP  GO DO IT AGAIN
00104A 00E6 81      EXIT  RTS  END OF BCD ADJUST
00105
```

PAGE 003 TMSNS .SA:1 TEMPSN

```
00107      00E7      A DELAY EQU      *      THIS LOOP IS JUST TO KILL TIME
00108A 00E7 AE FF      A      LDX      #SFF
00109A 00E9 A6 FF      A      LDA      #SFF
00110A 00EB 4A                DECA
00111A 00EC 26 FD 00EB      BNE      DELAY+4
00112A 00EE 5A                DECX
00113A 00EF 26 F8 00E9      BNE      DELAY+2
00114A 00F1 81                RTS
00115
00116
00117      *****BCD TO 7-SEGMENT DISPLAY ROUTINE*****
00118      *
00119      * DISPLAYS ARE REFRESHED EVERY 30MSEC WHEN
00120      * A TIMER INTERRUPT OCCURS. THE MOST SIGNIFICANT DIGIT
00121      * IS DISPLAYED FIRST. THE ROUTINE IS FOR DISPLAYS WITH
00122      * COMMON ANODE LEADS.
00123      00F2      A TMRINT EQU      *
00124A 00F2 99                SEC      SET THE CARRY BIT
00125A 00F3 A6 FF      A      LDA      #SFF      TURN 7-SEGMENT DRIVERS OFF
00126A 00F5 B7 01      A      STA      PORTB
00127A 00F7 36 42      A      ROR      SEGMENT SELECT NEXT DISPLAY
00128A 00F9 25 04 00FF      BCS      LOBYTE CHECK CARRY BIT FOR RESTART
00129A 00FB A6 FB      A      LDA      #SFB RESTART WITH MOST SIGNIFICANT DIGIT
00130A 00FD B7 42      A      STA      SEGMENT
00131A 00FF B6 41      A LOBYTE LDA      BCDLO
00132A 0101 02 42 04 0108      BRSET 1,SEGMENT,HIBYTE
00133A 0104 44                LSRA      SHIFT HIGH NYBBLE OF BCDLO
00134A 0105 44                LSRA      TO LOW NYBBLE FOR DISPLAY
00135A 0106 44                LSRA
00136A 0107 44                LSRA
00137A 0108 04 42 06 0111 HIBYTE BRSET 2,SEGMENT,DISPLY
00138A 0108 B6 40      A      LDA      BCDHI
00139A 010D 26 02 0111      BNE      DISPLY
00140A 010F A6 0B      A      LDA      #SOB BLANK IF ZERO
00141A 0111 A4 0F      A DISPLY AND      #SOF
00142A 0113 97                TAX
00143A 0114 EE 80      A      LDX      SEVSEG,X
00144A 0116 BF 01      A      STX      PORTB ENABLE DISPLAY DRIVERS
00145A 0118 B6 42      A      LDA      SEGMENT
00146A 011A B7 02      A      STA      PORTC ENABLE DISPLAY
00147A 011C A6 10      A      LDA      #S10 LOAD TIMER FOR 2048 MACHINE CYCLES
00148A 011E B7 08      A      STA      TDR
00149A 0120 A6 0F      A      LDA      #SOF RESET TIMER INTERRUPT FLAG
00150A 0122 B7 09      A      STA      TCR
00151A 0124 80                RTI
00152
```


PAGE 004 TMSNS .SA:1 TEMPSN

00154A	OF38			ORG	SF38	PROGRAM MASK OPTION REGISTER
00155A	OF38	80	A	FCB	S80	FOR RC CLOCK AND PROGRAMMABLE PRESCALER
00156						
00157A	OFF8			ORG	SFF8	RESET VECTORS USING AN MC68705R3
00158						
00159A	OFF8	00F2	A	TMRVEC	FDB	TMRINT
00160A	OFFA	0090	A	IROVEC	FDB	START
00161A	OFFC	0090	A	SWIVEC	FDB	START
00162A	OFFE	0090	A	RESET	FDB	START
00163					FND	
TOTAL ERRORS 00000--00000						

REFERENCES AND ADDITIONAL READING

Clayton, G.B. *Data Converters*. New York, NY: Halsted Press Books, 1982.

Jung, Walter G. *IC Converter Cookbook*. Indianapolis, IN: Howard W. Sams & Co., 1978.

Motorola, Inc. *MC3412 Laser Trimmed High-Speed 12-Bit D/A Converter Data Sheet*. Phoenix, AZ: Motorola Literature Distribution Center, 1982.

Motorola, Inc. *M6805 HMOS M146805 CMOS Family Microcomputer/Microprocessor User's Manual*. Phoenix, AZ: Motorola Literature Distribution Center, 1983.

Motorola, Inc. *MC68(7)05R/U 8-Bit Microcomputers Advance Information (ADI-977)*. Phoenix, AZ: Motorola Literature Distribution Center, 1984.

Motorola, Inc. *MTS102 Silicon Temperature Sensor Data Sheet*. Phoenix, AZ: Motorola Literature Distribution Center, 1981.

Sheingold, Daniel H., ed. *Transducer Interfacing Handbook—A Guide to Analog Signal Conditioning*. Norwood, MA: Analog Devices, 1980.

Titus, Jonathan A. *Microcomputer-Analog Converter Software & Hardware Interfacing*. Indianapolis, IN: Howard W. Sams & Co., 1979.

TELEPHONE DIALING TECHNIQUES USING THE MC6805

Prepared by
Robert Fischer
Downsview, Ontario, Canada

INTRODUCTION

Telephones and associated ancillary equipment providing intelligent features are fast becoming commonplace. Often, it is necessary for the microprocessor providing the intelligence to also dial a telephone number.

The M6805 Family microcomputers (MCU), with their proven hardware/software versatility, are ideal candidates for such applications. Illustrated here are two cost-effective methods of telephone dialing. Hardware and software is given for both Dual Tone Multi-Frequency (DTMF) and rotary-pulse type dialing.

DEMONSTRATION BOARD DESCRIPTION

Figure 1 shows the schematic of the demonstration board designed around a MC68705P3 single-chip MCU. This board is capable of pulse or DTMF dialing. The type of dialing is selected by switch S1. A 12-contact keyboard is used for input. While this is an extravagant use of I/O, it is acceptable for the purposes of a demonstration board.

Pulse dialing requires a direct connection to the telephone line. Interface to the line is made by a 600-ohm, 1:1 line transformer and a relay that provides on/off hook capability. An indicator light (LED #1) shows the current hook status.

After a power-on reset, the board is in an on-hook state (LED #1 off). The pressing of any key will result in an off-hook state without the digit being dialed. Subsequent key presses will result in the dialing of the corresponding digit. Pressing of the cancel button (S2) returns the board to the on-hook state.

The hardware and software to accomplish either form of dialing is readily applicable to any number of the M6805 Family.

ROTARY PULSE DIALING

From both a hardware and software viewpoint, pulse dialing is by far the simplest form of dialing to implement.

Pulse dialing requires that the telephone line circuit receive a make/break sequence at a 10-pulse-per-second (PPS) rate (see Figure 2). The dialing of the digit 3, for example, requires three make/break sequences. The 10-PPS rate requires the use of either a transistor or high speed relay for line looping. Note that if a low current reed relay is used, port B may be capable of driving the relay directly (eliminating the 2N3904 driver).

Subroutine PDIAL provides the proper timing sequences for pulse dialing. The routine is called with the digit to be dialed resident in the accumulator. Because the timing is not particularly critical, interrupts that can be quickly serviced are permissible.

DTMF DIALING

Dual tone multi-frequency tone dialing is considerably more complex in terms of ROM usage and external hardware. The M6805 MCU is required to generate two simultaneous sine waves of different frequencies. Table 1 shows the key pad digit and the frequencies of the corresponding tone pairs. Note that the tones fall into two groups:

Group	Frequency (Hz)
Low Tones	697, 770, 852, 941
High Tones	1209, 1336, 1477, 1633

TABLE 1 — Keypad Digit and Frequencies for Tone Pairs

Keypad Digit	DTONE Entry	Tone Pair (Hz)	
0	\$0	941	1336
1	\$1	697	1209
2	\$2	697	1336
3	\$3	697	1477
4	\$4	770	1209
5	\$5	770	1336
6	\$6	770	1477
7	\$7	852	1209
8	\$8	852	1336
9	\$9	852	1477
A	\$A	697	1633
B	\$B	770	1633
C	\$C	852	1633
D	\$D	941	1633
*	\$E	941	1209
#	\$F	941	1477

Also note that if the seldom used keys A, B, C, and D are not required, it is not necessary to generate a 1633-Hz tone.

The method used to generate the tones uses a series of 3-bit look-up tables. Consider a sine wave that has been sampled

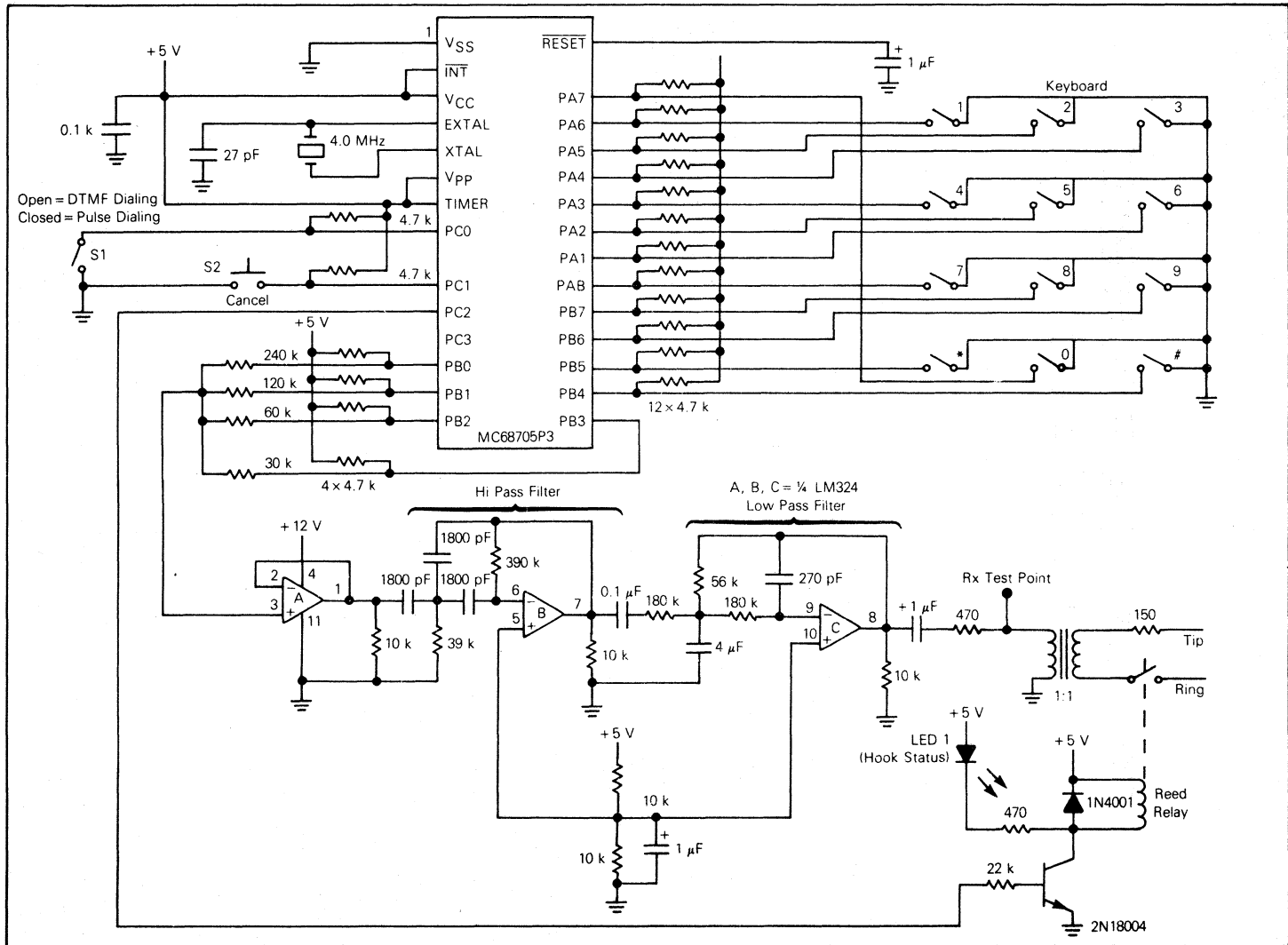


FIGURE 1 — Demonstration Board Schematic

at a constant interval, starting at the positive peak (see Figure 3). Sampling is continued until the next positive peak is encountered. There is, of course, some quantization error associated with this next found peak. If this group of samples were to be continuously cycled, a frequency error would result.

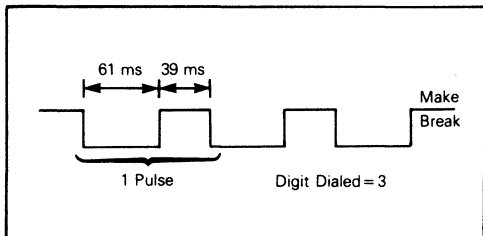


FIGURE 2 — Timing for Rotary Pulse Dialing

To cure this, continue sampling until the next peak is encountered and determine if the resultant frequency error falls within acceptable limits. Figure 3 is actually the output of a program written in BASIC for the EXORciser. This listing is included at the end of this application note. This program is used to design the look-up table for incorporation into the M6805 program according to the error rates acceptable in the end equipment.

This program prompts the user for the sample interval and the frequency of the tone which is to be generated. Sampling of the tone is thus automated and after a peak is encountered, the cumulative frequency error is calculated and displayed along with the sample count. If the user is satisfied with the percentage error, a table of the samples is generated. If the error is still unacceptable, the program continues sampling until the next peak is encountered. Note that the samples have all been "dc shifted".

This program was used to generate the look-up tables for all the tones given in Table 1 with a criteria of 1% maximum frequency error.

The subroutine DTONE actually operates on these tables to generate the DTMF tone pairs. The routine is entered with

the DTMF digit (see Table 1) is resident in the accumulator. Note that interrupts cannot be tolerated by this routine.

The first task of this routine is to convert the digit into the table start addresses for the high and low tones. This routine requires that the tables be resident in page 0 ROM to allow use of indexed addressing with 0 offset. The structure chosen for the tables puts the high group tones in the right nibble and the low group tones in the left nibble. Because the tables are all of different lengths, the table end is marked by an entry of \$F. In defiance of Murphy's Law, the DTMF tables fit exactly into page 0 ROM.

Generation of the tones involves cycling around a loop which plucks a 3-bit low tone sample and adds it to the 3-bit high tone sample. The 4-bit sum is then output to a D/A converter. If the end of the table marker is encountered for either sample, the pointer must be reset to the table start. This loop also keeps track of the duration of the tone burst by counting loops in TIMEH and TIMEL.

Notice that every program path through the loop takes a constant time (122 microseconds). The actual sequence of program development was to first write the loop, determine the execution time, and then, with the sample interval defined, generate the tone tables.

The 4-bit D/A converter is economically implemented with standard 5% resistors (60 kilohms = 30 kilohms + 30 kilohms). Port B was used because of its slightly superior high output voltage drive. It is still necessary to supplement the high drive with pull-up resistors.

One unfortunate by-product of this tone generation technique is the production of subharmonics (and, of course, harmonics). This necessitates the use of an active bandpass filter. This filter consists of separate high pass and low pass sections. The filter response is shown in Figure 4.

The output level to the telephone line is adjusted with a 470-ohm resistor in series with the line transformer primary. This also provides the RX point, where received audio can be obtained for duplex communication.

Using the BASIC software at the end of this application note, the generation of custom tone groups is readily accomplished. Single tone generation is also possible by using the table entry TNOFF at the end of the given DTMF tables. This allows the muting of either the high or low group tones.

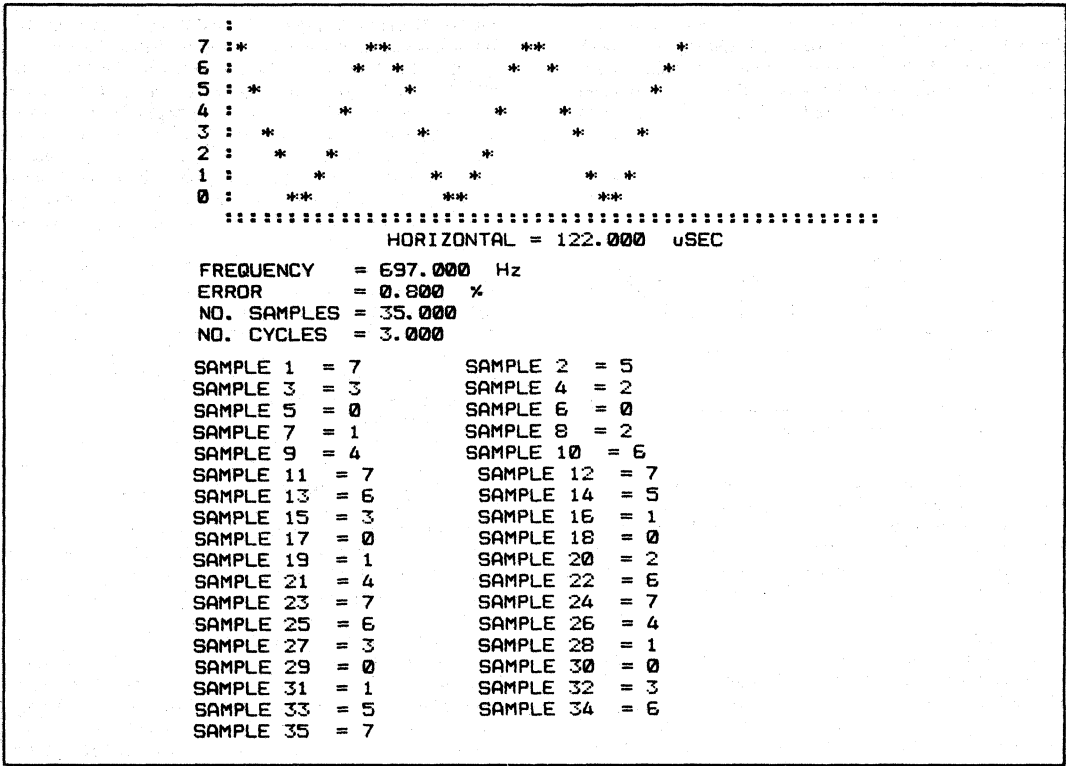


FIGURE 3 — Sine Wave Sampling

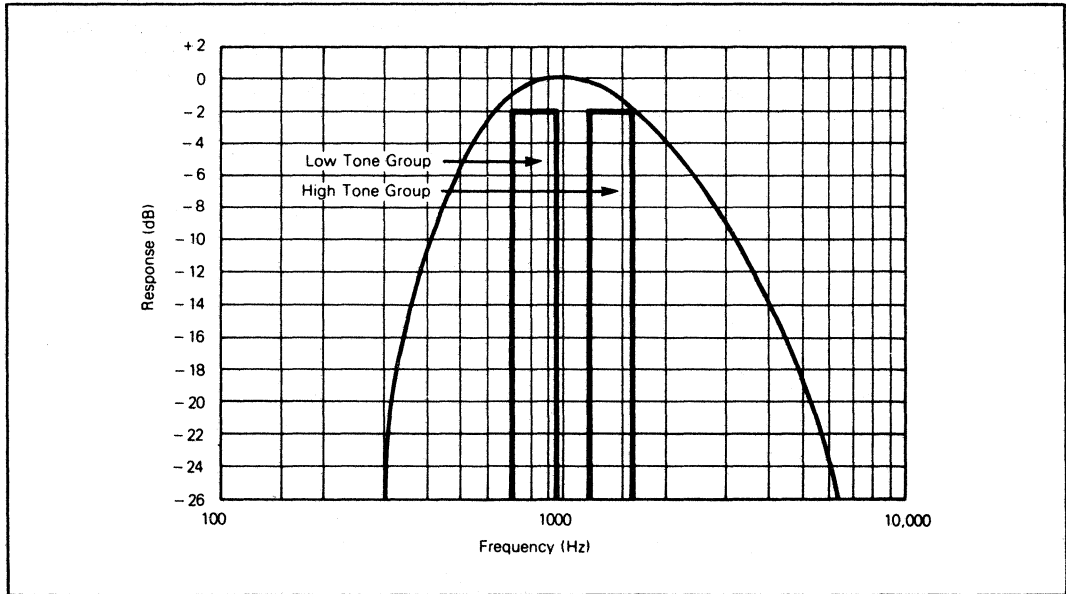


FIGURE 4 — Cumulative High-Pass, Low-Pass, and Line Transformer Response

PAGE 001 DEMO .SA:1 DEMO - MC68705P3 SOURCE FOR DIALER DEMO

00001 NAM DEMO
00002 TTL - MC68705P3 SOURCE FOR DIALER DE
00003 OPT ABS.CRE

00005 *****
00006 * *
00007 * DIALER DEMONSTRATION PROGRAM *
00008 * MC68705P3 VERSION *
00009 * ROBERT FISCHER *
00010 * MOTOROLA SEMICONDUCTOR PRODUCTS INC. *
00011 * TORONTO, CANADA *
00012 * 11/23/83 *
00013 *****

00015 *
00016 * SET MOR
00017 *
00018A 0784 ORG \$784
00019A 0784 5F A FCB %01011111

00021 *
00022 * PORT DEFINITION
00023 *

00025 0000 A PORTA EQU \$0 PORT A I/O
00026 0001 A PORTB EQU \$1 PORT B I/O
00027 0002 A PORTC EQU \$2 PORT C I/O
00028 0004 A PORTAD EQU \$4 PORT A DDR
00029 0005 A PORTBD EQU \$5 PORT B DDR
00030 0006 A PORTCD EQU \$6 PORT C DDR
00031 0008 A TMRDAT EQU \$8 TIMER DATA REGISTER
00032 0009 A TMRCON EQU \$9 TIMER CONTROL REGISTER

00034 *
00035 * RAM ALLOCATION
00036 *
00037A 0040 ORG \$40
00038A 0040 0001 A TMRCON RMB 1 TONE TIME (MS)
00039A 0041 0001 A TMRCON RMB 1 TONE TIME (LS)
00040A 0042 0001 A STARTH RMB 1 HI TONE TABLE START
00041A 0043 0001 A STARTL RMB 1 LO TONE TABLE START
00042A 0044 0001 A TONEH RMB 1 HI TONE CURRENT POINTER
00043A 0045 0001 A TONEL RMB 1 LO TONE CURRENT POINTER
00044A 0046 0001 A SCRATCH RMB 1 TEMPORARY SCRATCH
00045A 0047 0001 A LSTKEY RMB 1 LAST DIALED DIGIT USED

PAGE 002 DEMO .SA:1 DEMO - MC68705P3 SOURCE FOR DIALER DEMO

```
00047 *
00048 *          MAIN ROM
00049 *
00050A 0100          *          ORG    $100
00051 *
00052 *          COLD START.
00053 *          INITIALIZE PORTS.
00054 *
00055A 0100 3F 00    A START CLR   PORTA
00056A 0102 3F 01    A          CLR   PORTB
00057A 0104 3F 02    A          CLR   PORTC
00058A 0106 A6 0F    A          LDA   #%00001111
00059A 0108 B7 05    A          STA   PORTBD SET PORT B DDR
00060A 010A A6 04    A          LDA   #%00000100
00061A 010C B7 06    A          STA   PORTCD SET PORTC DDR
00062 *
00063 *          CHECK KEYBOARD FOR CLOSURE
00064 *
00065A 010E CD 01B5   A SCAN  JSR   KEYIN
00066A 0111 25 07     011A        BCS   DIAL
00067 *          NO CLOSURE- CHECK "ON HOOK" SWITCH
00068A 0113 02 02 F8 010E        BRSET  1,PORTC,SCAN
00069A 0115 15 02     A          BCLR  2,PORTC GO ON HOOK
00070A 0118 20 F4     010E        BRA   SCAN
00071 *
00072 *          DIGIT HAS BEEN PUSHED.
00073 *          IF PREVIOUSLY ON-HOOK; COME OFF-HOOK
00074 *          AND WAIT FOR NEXT KEY.
00075 *
00076A 011A 04 02 04 0121 DIAL  BRSET  2,PORTC,DODIAL
00077A 011D 14 02     A          BSET  2,PORTC OFF-HOOK
00078A 011F 20 ED     010E        BRA   SCAN
00079 *
00080 *          SEE IF PULSE DIALING OR DTMF
00081 *
00082A 0121 01 02 05 0129 DODIAL BRCLR  0,PORTC,PULSE
00083A 0124 CD 013B   A          JSR   DTONE  DTMF DIAL
00084A 0127 20 E5     010E        BRA   SCAN
00085 *
00086 *          PULSE DIALING
00087 *          "*" AND "#" KEYS ARE NOT RECOGNIZED
00088 *
00089A 0129 A1 0A     A PULSE  CMP   #*A
00090A 012B 22 E1     010E        BHI   SCAN
00091A 012D CD 01F5   A          JSR   PDIAL
00092A 0130 20 DC     010E        BRA   SCAN
```

PAGE 003 DEMO .SA:1 DEMO - MC68705P3 SOURCE FOR DIALER DEMO

```
00094 *****
00095 *
00096 *           SUBROUTINES *
00097 *
00098 *****
```

```
00100 *
00101 *           WAITMS
00102 *   WAITS THE NUMBER OF mSECS GIVEN IN ACC.
00103 *   BOTH ACC AND X ARE DESTROYED.
00104 *
00105A 0132 A6 7C      A WAITMS LDX   #124
00106A 0134 5A      WAIT1  DECX
00107A 0135 26 FD    0134   BNE   WAIT1
00108A 0137 4A      DECA
00109A 0138 26 F8    0132   BNE   WAITMS
00110A 013A 81      RTS
```

```
00112 *
00113 *           DTONE
00114 *   GENERATES 2 TONE BURST OF 65 mSEC
00115 *   FOLLOWED BY A FORCED 25 mSEC SILENT
00116 *   PERIOD FOR DTMF DIALING.
00117 *   DTMF DIGIT (*0-*9) IS GIVEN IN ACC.
00118 *
00119 *   INTERRUPTS MUST BE MASKED BEFORE ENTRY.
00120 *
00121A 013B A4 0F      A DTONE  AND   #*F
00122A 013D 97        TAX
00123A 013E 58        LSLX           X 2 FOR DISPLACEMENT
00124A 013F D6 0195  A      LDA   TTAB,X
00125A 0142 B7 42      A      STA  STARTH  HI TONE START POINTER
00126A 0144 B7 44      A      STA  TONEH   HI TONE CURRENT POINTER
00127A 0146 D6 0196  A      LDA   TTAB+1,X
00128A 0149 B7 43      A      STA  STARTL  LO TONE START POINTER
00129A 014B B7 45      A      STA  TONEL   LO TONE CURRENT POINTER
00130 *
00131 *   SET TONE TIME COUNTER
00132 *
00133A 014D A6 FD      A      LDA   #*FD
00134A 014F B7 40      A      STA  TIMEH   MS
00135A 0151 A6 EC      A      LDA   #*EC
00136A 0153 B7 41      A      STA  TIMEL   LS
00137 *
00138 *   TONE GENERATION LOOP.
00139 *   CONSTANT LOOP TIME OF 122 USECS.
00140 *
00141A 0155 3C 41      A TONELP INC   TIMEL
00142A 0157 26 08    0161   BNE   NCARRY
00143A 0159 3C 40      A      INC   TIMEH
00144A 015B 26 07    0154   CONT  BNE   GETLO
00145 *
00146 *   ALL DONE
00147 *
```


PRGE 004 DEMO .SA:1 DEMO - MC68705P3 SOURCE FOR DIALER DEMO

```

00148A 015D A6 19      A      LDA      #25
00149A 015F 20 D1      0132     BRA      WAITMS
00150
*
00151A 0161 3D          NDARRY NOP          NO CARRY- BURN 6 USECS
00152A 0162 20 F7      015E     BRA      CONT
00153
*
00154
* GET LO TONE
00155
*
00156A 0164 BE 45      A GETLO LDX      TONEL      LO TONE CURRENT POINTER
00157A 0166 FE          LDA      ,X              GET BYTE
00158
* LO TONES ARE IN LEFT NIBBLE
00159
* SEE IF TABLE END (MS BIT =1)
00160A 0167 2B 04      016D     BMI      GETNLO
00161A 0169 9D          NOP
00162A 016A 9D          NOP          FILL FOR CONSTANT TIME
00163A 016B 20 03      0170     BRA      SHFLO
00164
*
00165
* TABLE END- RESET TO TABLE START
00166
*
00167A 016D BE 43      A GETNLO LDX      STARTL
00168A 016F FE          LDA      ,X
00169
*
00170
* SWAP NIBBLES
00171
*
00172A 0170 44          SHFLO  LSR#
00173A 0171 44          LSR#
00174A 0172 44          LSR#
00175A 0173 44          LSR#
00176
* LO BYTE VALID- SAVE
00177A 0174 B7 4E      A      STA      SCRTCH
00178
* FIX POINTER AND SAVE
00179A 0176 5C          INCR
00180A 0177 BF 45      A      STX      TONEL
00181
*
00182
* GET HI TONE
00183
*
00184A 0179 BE 44      A      LDX      TONEH      HI TONE CURRENT POINTER
00185A 017B FE          LDA      ,X              GET BYTE
00186
* SEE IF TABLE END
00187A 017C A4 0F          A      AND      #$F
00188A 017E A1 0F          A      CMP      #$F
00189A 0180 27 04      018E     BEQ      GENHI
00190A 0182 9D          NOP
00191A 0183 9D          NOP          FILL FOR CONSTANT TIME
00192A 0184 20 03      0189     BRA      ADDNUM
00193
*
00194
* -I TABLE END- RESET TO START
00195
*
00196A 0186 BE 42      A GENHI LDX      STARTH
00197A 0188 FE          LDA      ,X
00198
* GET HI NIBBLE- FIX X
00199A 0189 5C          ADDNUM INCR
00200A 018A BF 44      A      STX      TONEH
00201
*
00202
* ADD LO AND HI TONES
00203
*
00204A 018C B9 4E      A      ADD      SCRTCH
00205
* OUTPUT THIS SAMPLE

```

PAGE 005 DEMO .SA: DEMO - M058705P3 SOURCE FOR DIALER DEMO

```
00206A 018E B7 01 A STA PORTB
00207 *
00208 * KEEP GOING
00209 *
00210A 0190 9D NOP
00211A 0191 9D NOP
00212A 0192 9D NOP
00213A 0193 20 00 0.55 BRA TONELP FILL FOR 122 USEC LOOP
```

```
00215 *
00216 * TONE PAIR LOOK-UP TABLE
00217 *
00218A 2195 9C A TTAB FCB TN1336 (2)
00219A 0195 E3 A FCB TN941
00220A 0197 80 A FCB TN1209 (1)
00221A 0198 80 A FCB TN697
00222A 0199 9C A FCB TN1336 (2)
00223A 019A 80 A FCB TN697
00224A 019B C2 A FCB TN1477 (3)
00225A 019C 80 A FCB TN697
00226A 019D 80 A FCB TN1209 (4)
00227A 019E A4 A FCB TN770
00228A 019F 9C A FCB TN1336 (5)
00229A 01A0 A4 A FCB TN770
00230A 01A1 C2 A FCB TN1477 (6)
00231A 01A2 A4 A FCB TN770
00232A 01A3 80 A FCB TN1209 (7)
00233A 01A4 C5 A FCB TN852
00234A 01A5 9C A FCB TN1336 (8)
00235A 01A6 C5 A FCB TN852
00236A 01A7 C2 A FCB TN1477 (9)
00237A 01A8 C5 A FCB TN852
00238A 01A9 D9 A FCB TN1633 (A)
00239A 01AA 80 A FCB TN697
00240A 01AB D9 A FCB TN1633 (B)
00241A 01AC A4 A FCB TN770
00242A 01AD D9 A FCB TN1633 (C)
00243A 01AE C5 A FCB TN852
00244A 01AF D9 A FCB TN1633 (D)
00245A 01B0 E3 A FCB TN941
00246A 01B1 80 A FCB TN1209 (*)
00247A 01B2 E3 A FCB TN941
00248A 21B3 C2 A FCB TN1477 (#)
00249A 01B4 E3 A FCB TN941
```

```
00251 *
00252 * KEYIN
00253 * SCANS KEYBOARD FOR PRESSED KEY.
00254 * WHEN A NEW KEY IS PRESSED, RETURN
00255 * WITH CARRY SET AND KEY VALUE IN ACC.
00256 *
00257A 21B5 AD 1F 01D6 KEYIN BSR CHECK CHECK FOR INPUT
00258A 01B7 25 15 01CE BCS KEY11 ANY KEY?
00259 *
```

```

00260          * NO KEY PRESSED.
00261          * IF FIRST TIME, WAIT OUT DEBOUNCE.
00262          *
00263A 0199 B6 47      A      LDA      LSTKEY
00264A 01BB A1 0A      A      CMP      #A
00265A 01BD 27 0D      01CC    BEQ      KEY22
00266          * FIRST TIME
00267A 01BF A6 19      A      LDA      #25
00268A 01C1 CD 0132    A      JSR      WAITMS
00269A 01C4 AD 10      01DE    BSR      CHECK
00270A 01CE 25 25      01CE    BCS      KEY11
00271          * STILL NO KEY
00272A 01C8 A6 0A      A      LDA      #A
00273A 01CA B7 47      A      STA      LSTKEY
00274A 01CC 98          KEY22  CLC
00275A 01CD 81          RTS
00276          *
00277          * GOT A KEY.
00278          * CHECK THAT IT IS FIRST TIME.
00279          *
00280A 01CE B1 47      A KEY11  CMP      LSTKEY
00281A 01D0 27 FA      01CC    BEQ      KEY22
00282A 01D2 B7 47      A      STA      LSTKEY
00283A 01D4 99          SEC
00284A 01D5 81          RTS

00286          *
00287          *      CHECK
00288          * SCANS KEYBOARD FOR CLOSURE.
00289          * IF KEY CLOSED, RETURNS WITH KEY
00290          * VALUE IN ACC AND CARRY SET.
00291          *
00292A 01D6 4F          CHECK  CLRA
00293A 01D7 BE 00      A      LDX      PORTA    GET KEYS
00294A 01D9 58          KEY11  LSLX
00295A 01DA 24 17      01F3    BCC      GOTKEY    KEY CLOSED?
00296A 01DC 4C          INCA      NO
00297A 01DD A1 08      A      CMP      #8
00298A 01DF 26 F8      01D9    BNE      KEY11
00299          * CHECK BALANCE OF KEYS ON PORTB
00300A 01E1 0F 01 0F 01F3  BRCLR   7, PORTB, GOTKEY "8"?
00301A 01E4 4C          INCA
00302A 01E5 00 01 0B 01F3  BRCLR   6, PORTB, GOTKEY "9"?
00303A 01E8 A6 0E      A      LDA      #E
00304A 01EA 0B 01 06 01F3  BRCLR   5, PORTB, GOTKEY "A"?
00305A 01ED 4C          INCA
00306A 01EE 09 01 02 01F3  BRCLR   4, PORTB, GOTKEY "B"?
00307          *
00308          * NO KEY PRESSED.
00309          *
00310A 01F1 98          CLC
00311A 01F2 81          RTS
00312          *
00313          * GOT A KEY.
00314          *
00315A 01F3 99          GOTKEY  SEC

```

PAGE 007 DEVD .SA:1 DEMO - MCGS705P3 SOURCE FOR DIALER DEVD

00316A 01F4 81 RTS

```
00318          *
00319          * PDIAL
00320          * PULSE DIALS THE DIGIT GIVEN IN
00321          * ACC. FOLLOWED BY A 200 MSEC. WAIT.
00322          *
00323A 01F5 4D PDIAL1 STA BNE PDIAL1
00324A 01F6 25 02 01FA BNE PDIAL1
00325          * CHANGE "0" TO *$A
00325A 01F8 A6 0A A LDA *$A
00327A 01FA E7 4E A PDIAL1 STA SCRATCH SAVE
00328          * MAKE A PULSE
00329A 01FC 15 02 A PDIAL2 BCLR 2, PORTC ON HOOK
00330A 01FE A6 0D A LDA #51
00331A 0200 CD 0132 A JSR WAITMS
00332A 0203 14 02 A BSET 2, PORTC OFF HOOK
00333A 0205 A5 27 A LDA #39
00334A 0207 CD 0132 A JSR WAITMS
00335A 020A 3A 46 A DEC SCRATCH
00336A 020C 25 5E 01FC BNE PDIAL2
00337          * WAIT 200 MSEC INTER-DIGIT
00338A 020E A6 08 A LDA #200
00339A 0210 CC 0132 A JMP WAITMS
```

00341 *
 00342 * TOUCH TONE GENERATOR TABLE
 00343 *

00345A	0080			ORG	\$80
00346A	0080	77	A TN697	FCB	\$77
00347A	0081	76	A	FCB	\$76
00348A	0082	53	A	FCB	\$53
00349A	0083	30	A	FCB	\$30
00350A	0084	21	A	FCB	\$21
00351A	0085	03	A	FCB	\$03
00352A	0086	06	A	FCB	\$06
00353A	0087	17	A	FCB	\$17
00354A	0088	25	A	FCB	\$25
00355A	0089	42	A	FCB	\$42
00356A	008A	60	A	FCB	\$60
00357A	008B	71	A	FCB	\$71
00358A	008C	74	A	FCB	\$74
00359A	008D	67	A	FCB	\$67
00360A	008E	57	A	FCB	\$57
00361A	008F	34	A	FCB	\$34
00362A	0090	11	A	FCB	\$11
00363A	0091	00	A	FCB	\$00
00364A	0092	02	A	FCB	\$02
00365A	0093	15	A	FCB	\$15
00366A	0094	27	A	FCB	\$27
00367A	0095	46	A	FCB	\$46
00368A	0096	64	A	FCB	\$64
00369A	0097	71	A	FCB	\$71
00370A	0098	70	A	FCB	\$70
00371A	0099	62	A	FCB	\$62
00372A	009A	45	A	FCB	\$45
00373A	009B	3F	A	FCB	\$3F
00374A	009C	17	A TN1336	FCB	\$17
00375A	009D	05	A	FCB	\$05
00376A	009E	02	A	FCB	\$02
00377A	009F	10	A	FCB	\$10
00378A	00A0	31	A	FCB	\$31
00379A	00A1	55	A	FCB	\$55
00380A	00A2	67	A	FCB	\$67
00381A	00A3	F6	A	FCB	\$F6
00382A	00A4	72	A TN770	FCB	\$72
00383A	00A5	60	A	FCB	\$60
00384A	00A6	51	A	FCB	\$51
00385A	00A7	34	A	FCB	\$34
00386A	00A8	17	A	FCB	\$17
00387A	00A9	06	A	FCB	\$06
00388A	00AA	03	A	FCB	\$03
00389A	00AB	20	A	FCB	\$20
00390A	00AC	41	A	FCB	\$41
00391A	00AD	54	A	FCB	\$54
00392A	00AE	77	A	FCB	\$77
00393A	00AF	76	A	FCB	\$76
00394A	00B0	63	A	FCB	\$63
00395A	00B1	40	A	FCB	\$40
00396A	00B2	20	A	FCB	\$20

00397A	00B3	13	A	FCB	\$13
00398A	00B4	06	A	FCB	\$06
00399A	00B5	17	A	FCB	\$17
00400A	00B6	24	A	FCB	\$24
00401A	00B7	41	A	FCB	\$41
00402A	00B8	60	A	FCB	\$60
00403A	00B9	73	A	FCB	\$73
00404A	00BA	76	A	FCB	\$76
00405A	00BB	57	A	FCB	\$57
00406A	00BC	34	A	FCB	\$34
00407A	00BD	11	A	FCB	\$11
00408A	00BE	00	A	FCB	\$00
00409A	00BF	03	A	FCB	\$03
00410A	00C0	16	A	FCB	\$16
00411A	00C1	3F	A	FCB	\$3F
00412A	00C2	57	A TN1477	FCB	\$57
00413A	00C3	65	A	FCB	\$65
00414A	00C4	F1	A	FCB	\$F1
00415A	00C5	70	A TN852	FCB	\$70
00416A	00C6	63	A	FCB	\$63
00417A	00C7	46	A	FCB	\$46
00418A	00C8	27	A	FCB	\$27
00419A	00C9	03	A	FCB	\$03
00420A	00CA	00	A	FCB	\$00
00421A	00CB	11	A	FCB	\$11
00422A	00CC	35	A	FCB	\$35
00423A	00CD	57	A	FCB	\$57
00424A	00CE	75	A	FCB	\$75
00425A	00CF	72	A	FCB	\$72
00426A	00D0	60	A	FCB	\$60
00427A	00D1	42	A	FCB	\$42
00428A	00D2	16	A	FCB	\$16
00429A	00D3	07	A	FCB	\$07
00430A	00D4	04	A	FCB	\$04
00431A	00D5	20	A	FCB	\$20
00432A	00D6	41	A	FCB	\$41
00433A	00D7	64	A	FCB	\$64
00434A	00D8	7F	A	FCB	\$7F
00435A	00D9	77	A TN1633	FCB	\$77
00436A	00DA	55	A	FCB	\$55
00437A	00DB	31	A	FCB	\$31
00438A	00DC	11	A	FCB	\$11
00439A	00DD	05	A	FCB	\$05
00440A	00DE	17	A	FCB	\$17
00441A	00DF	25	A	FCB	\$25
00442A	00E0	51	A	FCB	\$51
00443A	00E1	61	A	FCB	\$61
00444A	00E2	F4	A	FCB	\$F4
00445A	00E3	77	A TN941	FCB	\$77
00446A	00E4	65	A	FCB	\$65
00447A	00E5	41	A	FCB	\$41
00448A	00E6	21	A	FCB	\$21
00449A	00E7	04	A	FCB	\$04
00450A	00E8	07	A	FCB	\$07
00451A	00E9	25	A	FCB	\$25
00452A	00EA	51	A	FCB	\$51
00453A	00EB	70	A	FCB	\$70
00454A	00EC	74	A	FCB	\$74

PAGE 010 DEMO .SA:1 DEMO - MC68705P3 SOURCE FOR DIALER DEMO

00455A	00ED	57	A	FCB	\$67
00456A	00EE	35	A	FCB	\$35
00457A	00EF	11	A	FCB	\$11
00458A	00F0	00	A	FCB	\$00
00459A	00F1	14	A	FCB	\$14
00460A	00F2	37	A	FCB	\$37
00461A	00F3	55	A	FCB	\$55
00462A	00F4	71	A	FCB	\$71
00463A	00F5	70	A	FCB	\$70
00464A	00F6	54	A	FCB	\$54
00465A	00F7	37	A	FCB	\$37
00466A	00F8	15	A	FCB	\$15
00467A	00F9	01	A	FCB	\$01
00468A	00FA	10	A	FCB	\$10
00469A	00FB	44	A	FCB	\$44
00470A	00FC	6F	A	FCB	\$6F
00471A	00FD	FF	A	FCB	\$FF
00472A	00FE	00	A TNOFF	FCB	\$00
00473A	00FF	FF	A	FCB	\$FF

00475 0080 A TN1209 EQU TN697 (SHARE START ADDRESS)

```

00478
00479
00480
00481A 07F8
00482A 07F8 0100 A ORG $7F8
00483A 07FA 0100 A FDB START TIMER INERRUPT
00484A 07FC 0100 A FDB START HARDWARE INTERRUPT
00485A 07FE 0100 A FDB START SWI
00486 END
TOTAL ERRORS 00000--00000
    
```

```

0189 ADDNUM 00192 00199*
01D6 CHECK 00257 00269 00292*
015B CONT 00144+00152
011A DIAL 00066 00076*
0121 DODIAL 00076 00082*
013B DTCNE 00083 00121*
0186 GENMI 00189 00196*
0164 GETLO 00144 00156*
016D GETVLO 00160 00167*
0143 GOTKEY 00295 00300 00302 00304 00306 00315*
01D9 KEY1 00294+00298
01CE KEY11 00258 00270 00280*
01CC KEY22 00265 00274+00281
01B5 KEYIN 00065 00257*
0047 LSTKEY 00045+00263 00273 00280 00282
01E1 NCARRY 00142 00151*
01F5 PDIAL 00091 00323*
01FA PDIAL1 00324 00327*
01FC PDIAL2 00329+00336
0020 PORTA 00025+00055 00293
0004 PORTAB 00028*
0021 PORTB 00026+00056 00205 00300 00302 00304 00306
0005 PORTBD 00029+00059
0002 PORTC 00027+00057 00068 00069 00076 00077 00082 00329 00332
0006 PORTCD 00030+00061
0129 PULSE 00082 00089*
017E SCAN 00065+00068 00070 00078 00084 00090 00092
0046 SCRTCH 00044+00177 00204 00327 00336
0170 SHALO 00163 00172*
0100 START 00055+00482 00483 00484 00485
0042 STARTH 00040+00125 00196
0043 STARTL 00041+00128 00167
0040 TIMER 00038+00134 00143
0041 TIVEL 00039+00136 00141
0009 TRACON 00032*
0008 TRADAT 00031*
0080 TN1208 00220 00226 00232 00246 00475*
0290 TN1306 00218 00222 00228 00234 00374*
0002 TN1477 00224 00230 00236 00248 00412*
00D9 TN1633 00238 00240 00242 00244 00435*
    
```


PAGE 012 DEMO .SA:1 DEMO - M06870593 SOURCE FOR DIPLER DEMO

0090 TN697 00221 00223 00225 00239 00346+00475
00A4 TN770 00227 00229 00231 00241 00382*
00C5 TN852 00233 00235 00237 00243 00415*
00E3 TN941 00219 00245 00247 00249 00445*
00FE TNOFF 00472*
0044 TONE# 00042*00126 00184 00200
0045 TONEL 00043*00129 00156 00180
0155 TONELP 00141*00213
0195 TTAB 00124 00127 00218*
0134 WAIT1 00106*00107
0032 WAITMS 00105*00109 00149 00268 00331 00334 00339

PAGE 001 TONCAL .SA:1

```
0100 DIM T(100)
0101 H=1
0102 P=122E-6
0110 C=0
0120 W=P*1E6
0121 PRINT "PERIOD = ";W;"USECS (Y/N)";
0122 INPUT A$
0123 IF A$="Y" THEN GOTO 130
0124 IF A$("<n") THEN GOTO 121
0125 INPUT "PERIOD (USECS)=";P
0126 P=P*1E-6
0130 PRINT "TONE FREQ=";
0140 INPUT F
0150 X=2*3.141593*F*P
0160 FOR N=1 TO 100
0170 T(N)=INT(3.5*COS(N*X)+4)
0171 GOSUB 8000
0190 IF T(N)/7 THEN GOTO 370
0200 REM GOT A CREST- CALCULATE CUMULATIVE ERROR
0220 Q=N*P
0230 E=(C/F-Q)*100/Q
0240 GOSUB 7000
0260 PRINT "ERROR          =" ;E;"%"
0270 PRINT "SAMPLE COUNT=";N
0275 PRINT "FREQUENCY      =" ;F;"Hz"
0280 PRINT "CONTINUE (Y/N)";
0290 INPUT A$
0300 IF A$="Y" THEN GOTO 370
0310 IF A$("<n") THEN GOTO 280
0320 REM DONE- PRINT SAMPLE LIST
0330 GOSUB 9000
0332 GOTO 110
0370 NEXT N
0380 PRINT "SAMPLE COUNT > 100"
0390 END
7000 REM PLOT SUBROUTINE
7001 PRINT CHR$(27);"a"
7002 FOR I=1 TO 50
7003 NEXT I
7005 PRINT
7006 PRINT "  I"
7010 FOR I = 1 TO 8
7020 L=8-I
7030 PRINT L;" ";
7040 REM PRINT THIS LINE
7050 FOR J=1 TO N
7060 IF T(J)=L THEN GOTO 7090
7070 PRINT " ";
7080 GOTO 7100
7090 PRINT "*";
7100 NEXT J
7110 PRINT
7120 NEXT I
7130 PRINT "  ";
7140 FOR I=1 TO 70
7150 PRINT " ";
7160 NEXT I
7165 W=P*1E6
```

PAGE 002 TONCAL .SA:1

```
7170 PRINT "                                HORIZONTAL_ =";W;"uSEC"
7180 PRINT
7190 PRINT
7200 RETURN
8000 REM CREST COUNT ROUTINE- LOOK FOR A ZERO CROSSING
8010 IF T(N) > 3 THEN GOTO 8040
8020 H=0
8030 RETURN
8040 IF H=0 THEN C=C+1
8050 H=1
8060 RETURN
9000 REM PRINTER OUTPUT SUBROUTINE
9011 FOR J=1 TO 4
9012 PRINT #2
9013 NEXT J
9020 PRINT #2
9030 PRINT #2, " " ; "
9040 FOR I=1 TO 8
9050 L=8-I
9060 PRINT #2, L; " " ; "
9070 FOR J=1 TO N
9080 IF T(J)=L THEN GOTO 9110
9090 PRINT #2, " " ; "
9100 GOTO 9120
9110 PRINT #2, "*"; "
9120 NEXT J
9130 PRINT #2
9140 NEXT I
9150 PRINT #2, " " ; "
9160 FOR I=1 TO 70
9170 PRINT #2, " " ; "
9180 NEXT I
9190 W=P*1E6
9191 DIGITS= 3
9192 PRINT #2
9200 PRINT #2, "                                HORIZONTAL = ";W;" uSEC"
9210 PRINT #2
9220 PRINT #2
9230 PRINT #2, "FREQUENCY = ";F;" Hz"
9240 PRINT #2, "ERROR = ";E;" %"
9250 PRINT #2, "NO. SAMPLES = ";N
9260 PRINT #2, "NO. CYCLES = ";C
9270 PRINT #2
9271 DIGITS= 0
9280 PRINT #2
9290 FOR J=1 TO N STEP 2
9300 PRINT #2, "SAMPLE ";J;" = ";T(J);
9310 B=J+1
9320 IF B>N THEN GOTO 9340
9330 PRINT #2, " SAMPLE ";B;" = ";T(B);
9340 PRINT #2
9350 NEXT J
9360 PRINT #2, CHR$(12)
9370 RETURN
```

MC68HC11 Floating-Point Package

INTRODUCTION

The MC68HC11 is a very powerful and capable single-chip microcomputer. Its concise instruction set combined with six powerful addressing modes, true bit manipulation, 16-bit arithmetic operations and a second 16-bit index register make it ideal for control applications requiring both high-speed I/O and high-speed calculations.

While most applications can be implemented by using the 16-bit integer precision of the MC68HC11, certain applications or algorithms may be difficult or impossible to implement without floating-point math. The goal in writing the MC68HC11 floating-point package was to provide a fast, flexible way to do floating-point math for just such applications.

The HC11 floating-point package (HC11FP) implements more than just the four basic math functions (add, subtract, multiply, and divide); it also provides routines to convert from ASCII to floating point and from floating point to ASCII. For those applications that require it, the three basic trig functions SINE, COSine, and TANGent are provided along with some trig utility functions for converting to and from both radians and degrees. The square root function is also included.

For those applications that can benefit by using both integer and floating-point operations, there are routines to convert to and from integer and floating-point format.

The entire floating-point package requires just a little over 2k bytes of memory and only requires ten bytes of page-zero RAM in addition to stack RAM. All temporary variables needed by the floating-point routines, reside on the stack. This feature makes the routines completely re-entrant as long as the ten bytes of page zero RAM are saved before using any of the routines. This will allow both interrupt routines and main line programs to use the floating-point package without interfering with one another.

FLOATING-POINT FORMAT

FLOATING-POINT ACCUMULATOR FORMAT

The ten bytes of page-zero RAM are used for two software floating-point accumulators FPACC1 and FPACC2. Each five-byte accumulator consists of a one-byte exponent, a three-byte mantissa, and one byte that is used to indicate the mantissa sign.

The exponent byte is used to indicate the position of the binary point and is biased by decimal 128 (\$80) to make floating-point comparisons easier. This one-byte exponent gives a dynamic range of about $1 \times 10 \pm 38$.

The mantissa consists of three bytes (24 bits) and is used to hold both the integer and fractional portion of the floating-point number. The mantissa is always assumed to be "normalized" (i.e., most-significant bit of the most-significant byte a one). A 24-bit mantissa will provide slightly more than seven decimal digits of precision.

A separate byte is used to indicate the sign of the mantissa rather than keeping it in twos complement form so that unsigned arithmetic operations may be used when manipulating the mantissa. A positive mantissa is indicated by this byte being equal to zero (\$00). A negative mantissa is indicated by this byte being equal to minus one (\$FF).

FPACC1	82 C90FDB 00	+ 3.1415927
FPACC2	82 C90FDB FF	- 3.1415927

MEMORY FORMAT

The way that floating-point numbers are stored in memory or the "memory format" of a floating-point number is slightly different than its floating-point accumulator format. In order to save memory, floating-point numbers are stored in memory in a format called "hidden bit normalized form".

In this format, the number is stored into four consecutive bytes with the exponent residing at the lowest address. The mantissa is stored in the next three consecutive bytes with the most-significant byte stored in the lowest address. Since the most-significant bit of the mantissa in a normalized floating-point number is always a one, this bit can be used to store the sign of the mantissa. This results in positive numbers having the most-significant bit of the mantissa cleared (zero) and negative numbers having their most-significant bit set (one). An example follows:

82 490FDB	+ 3.1415927
82 C90FDB	- 3.1415927

There are four routines that can be used to save and load the floating-point accumulators and at the same time convert between the floating-point accumulator and memory format. These routines are discussed in detail in **FLOATING-POINT ROUTINES**.

ERRORS

There are seven error conditions that may be returned by the HC11 floating-point package. When an error occurs, the condition is indicated to the calling program by setting the carry bit in the condition code register and returning an error code in the A-accumulator. The error codes and their meanings are explained below.

Error #	Meaning
1	Format Error in ASCII to Floating-Point Conversion
2	Floating-Point Overflow
3	Floating-Point Underflow
4	Division by Zero (0)
5	Floating-Point Number too Large or Small to Convert to Integer
6	Square Root of a Negative Number
7	TAN of $\pi/2$ (90)

NOTE

None of the routines check for valid floating-point numbers in either FPACC1 or FPACC2. Having illegal floating-point values in the floating-point accumulators will produce unpredictable results.

FLOATING-POINT ROUTINES

The following paragraphs provide a description of each routine in the floating-point package. The information provided includes the subroutine name, operation performed, subroutine size, stack space required, other subroutines that are called, input, output, and possible error conditions.

The Stack Space required by the subroutine includes not only that required for the particular routines local variables, but also stack space that is used by any other subroutines that are called including return addresses. Note that the trig functions require a good deal of stack space.

Since some applications may not require all the routines provided in the floating-point package, the description of each routine includes the names of other subroutines that it calls. This makes it easy to determine exactly which subroutines are required for a particular function.

ASCII-TO-FLOATING-POINT CONVERSION

Subroutine Name: ASCFLT
 Operation: ASCII (X) \rightarrow FPACC1
 Size: 352 Bytes (includes NUMERIC subroutine)
 Stack Space: 14 Bytes
 Calls: NUMERIC, FPNORM, FLTMUL, PSHFPAC2, PULFPAC2
 Input: X register points to ASCII string to convert.
 Output: FPACC1 contains the floating-point number.
 Error Conditions: Floating-point format error may be returned.
 Notes: This routine converts an ASCII floating-point number to the format required by all of the floating-point routines. Conversion stops either when a non-decimal character is encountered before the exponent or after one or two exponent digits have been converted. The input format is very flexible. Some examples are shown below.

20.095
 0.125
 7.2984E + 10
 167.824E5
 005.9357E - 7
 500

FLOATING-POINT MULTIPLY

Subroutine Name: FLTMUL
 Operation: FPACC1 \times FPACC2 \rightarrow FPACC1
 Size: 169 Bytes
 Stack Space: 10 Bytes
 Calls: PSHFPAC2, PULFPAC2, CHCK0
 Input: FPACC1 and FPACC2 contain the numbers to be multiplied.
 Output: FPACC1 contains the product of the two floating-point accumulators. FPACC2 remains unchanged.
 Error Conditions: Overflow, Underflow.

FLOATING-POINT ADD

Subroutine Name: FLTADD
Operation: $FPACC1 + FPACC2 \rightarrow FPACC1$
Size: 194 Bytes
Stack Space: 6 Bytes
Calls: PSHFPAC2, PULFPAC2, CHCK0
Input: FPACC1 and FPACC2 contain the numbers to be added.
Output: FPACC1 contains the sum of the two numbers. FPACC2 remains unchanged.
Error Conditions: Overflow, Underflow.
Notes: The floating-point add routine performs full signed addition. Both floating-point accumulators may have mantissas with the same or different sign.

FLOATING-POINT SUBTRACT

Subroutine Name: FLTSUB
Operation: $FPACC1 - FPACC2 \rightarrow FPACC1$
Size: 12 Bytes
Stack Space: 8 Bytes
Calls: FLTADD
Input: FPACC1 and FPACC2 contain the numbers to be subtracted.
Output: FPACC1 contains the difference of the two numbers ($FPACC1 - FPACC2$). FPACC2 remains unchanged.
Error Conditions: Overflow, Underflow.
Notes: Since FLTADD performs full signed addition, the floating-point subtract routine inverts the sign byte of FPACC2, calls FLTADD, and then changes the sign of FPACC2 back to what it was originally.

FLOATING-POINT DIVIDE

Subroutine Name: FLTDIV
Operation: $FPACC1 \div FPACC2 \rightarrow FPACC1$
Size: 209 Bytes
Stack Space: 11 Bytes
Calls: PSHFPAC2, PULFPAC2
Input: FPACC1 and FPACC2 contain the divisor and dividend respectively.
Output: FPACC1 contains the quotient. FPACC2 remains unchanged.
Error Conditions: Divide by zero, Overflow, Underflow

FLOATING-POINT-TO-ASCII CONVERSION

Subroutine Name: FLTASC
Operation: $FPACC1 \rightarrow (X)$
Size: 370 Bytes
Stack Space: 28 Bytes
Calls: FLTMUL, FLTCMP, PSHFPAC2, PULFPAC2
Input: FPACC1 contains the number to be converted to an ASCII string. The index register X points to a 14 byte string buffer.
Output: The buffer pointed to by the X index register contains an ASCII string that represents the number in FPACC1. The string is terminated with a zero (0) byte and the X register points to the start of the string.
Error Conditions: None

FLOATING POINT COMPARE

Subroutine Name: FLTCMP
Operation: FPACC1 - FPACC2
Size: 42 Bytes
Stack Space: None
Calls: None
Input: FPACC1 and FPACC2 contain the numbers to be compared.
Output: Condition codes are properly set so that all branch instructions may be used to alter program flow. FPACC1 and FPACC2 remain unchanged.
Error Conditions: None

UNSIGNED INTEGER TO FLOATING POINT

Subroutine Name: UINT2FLT
Operation: (16-bit unsigned integer) \blacklozenge FPACC1
Size: 18 Bytes
Stack Space: 6 Bytes
Calls: FPNORM, CHCK0
Input: The lower 16-bits of the FPACC1 mantissa contain an unsigned 16-bit integer.
Output: FPACC1 contains the floating-point representation of the 16-bit unsigned integer.
Error Conditions: None

SIGNED INTEGER TO FLOATING POINT

Subroutine Name: SINT2FLT
Operation: (16-bit signed integer) \blacklozenge FPACC1
Size: 24 Bytes
Stack Space: 7 Bytes
Calls: UINT2FLT
Input: The lower 16-bits of the FPACC1 mantissa contain a signed 16-bit integer.
Output: FPACC1 contains the floating-point representation of the 16-bit signed integer.
Error Conditions: None

FLOATING POINT TO INTEGER

Subroutine Name: FLT2INT
Operation: FPACC1 \blacklozenge (16-bit signed or unsigned integer)
Size: 74 Bytes
Stack Space: 2 Bytes
Calls: CHCK0
Input: FPACC1 may contain a floating-point number in the range $65535 \leq \text{FPACC1} \leq -32767$.
Output: The lower 16-bits of the FPACC1 mantissa will contain a 16-bit signed or unsigned number.
Error Conditions: None
Notes: If the floating-point number in FPACC1 is positive, it will be converted to an unsigned integer. If the number is negative it will be converted to a signed twos complement integer. This type of conversion will allow 16-bit addresses to be represented as positive numbers in floating-point format. Any fractional part of the floating-point number is discarded.

TRANSFER FPACC1 TO FPACC2

Subroutine Name: TFR1TO2
Operation: FPACC1 \blacklozenge FPACC2
Size: 13 Bytes
Stack Space: 0 Bytes
Calls: None
Input: FPACC1 contains a floating-point number.
Output: FPACC2 contains the same number as FPACC1.
Error Conditions: None

FLOATING-POINT FUNCTIONS

The following paragraphs describe the supplied floating-point functions, returned results, and possible error conditions. Note that even though the Taylor series which is used to calculate the trig functions requires that the input angle be expressed in radians; less precision is lost through angle reduction if the angle being reduced is expressed in degrees. Once the angle is reduced, the DEG2RAD subroutine is called to convert the angle to radians.

To reduce the number of factors in the Taylor expansion series all angles are reduced to fall between 0° and 45° by the ANGRED subroutine. This subroutine returns the reduced angle in FPACC1 along with the quad number that the original angle was in, and a flag that tells the calling routine whether it actually needs to calculate the sine or the cosine of the reduced angle to obtain the proper answer.

SQUARE ROOT

Subroutine Name: FLTSQR
Operation: $\sqrt{\text{FPACC1}}$ \rightarrow FPACC1
Size: 104 Bytes
Stack Space: 21 Bytes
Calls: TFR1TO2, FLTDIV, FLTADD, PSHFPAC2, PULFPAC2
Input: FPACC1 contains a valid floating-point number.
Output: FPACC1 contains the square root of the original number. FPACC2 is unchanged.
Error Conditions: NSQRTERR is returned if the number in FPACC1 is negative and FPACC1 remains unchanged.

SINE

Subroutine Name: FLTSIN
Operation: $\text{SIN}(\text{FPACC1}) \rightarrow \text{FPACC1}$
Size: 380 Bytes (Includes SINCOS subroutine)
Stack Space: 50 Bytes
Calls: ANGRED, SINCOS, DEG2RAD, PSHFPAC2, PULFPAC2
Input: FPACC1 contains an angle in radians in the range $-2\pi \leq \text{FPACC1} \leq +2\pi$.
Output: FPACC1 contains the sine of FPACC1, and FPACC2 remains unchanged.
Error Conditions: None
Notes: The Taylor Expansion Series is used to calculate the sine of the angle between 0° and 45° ($\pi \div 4$). The subroutine ANGRED is called to reduce the input angle to within this range. Spot checks show a maximum error of $+1.5 \times 10^{-7}$ throughout the input range.

COSINE

Subroutine Name: FLTCOS
Operation: $\text{COS}(\text{FPACC1}) \rightarrow \text{FPACC1}$
Size: 384 Bytes (Includes SINCOS subroutine)
Stack Space: 50 Bytes
Calls: ANGRED, FLTSIN, DEG2RAD, PSHFPAC2
Input: FPACC1 contains an angle in radians in the range $-2\pi \leq \text{FPACC1} \leq +2\pi$.
Output: FPACC1 contains the cosine of FPACC1, and FPACC2 remains unchanged.
Error Conditions: None
Notes: The Taylor Expansion Series is used to calculate the cosine of the angle between 0° and 45° ($\pi \div 4$). The subroutine ANGRED is called to reduce the input angle to within this range. Spot checks show a maximum error of $+1.5 \times 10^{-7}$ throughout the input range.

TANGENT

Subroutine Name: FLTTAN
Operation: TAN (FPACC1) \blacklozenge FPACC1
Size: 35 Bytes (Also requires FLTSIN and FLCOS)
Stack Space: 56 Bytes
Calls: TFR1TO2, EXG1AND2, FLTSIN, FLCOS, FLTDIV, PSHFPAC2, PULFPAC2
Input: FPACC1 contains an angle in radians in the range $-2\pi \leq \text{FPACC1} \leq +2\pi$.
Output: FPACC1 contains the tangent of the input angle, and FPACC2 remains unchanged.
Error Conditions: Returns largest legal number if tangent of $\pm\pi/2$ is attempted.
Notes: The tangent of the input angle is calculated by first obtaining the sine and cosine of the input angle and then using the following formula: $\text{TAN} = \text{SIN} / \text{COS}$. At 89.9° the tangent function is only accurate to 5 decimal digits. For angles greater than 89.9° accuracy decreases rapidly.

DEGREES TO RADIANS CONVERSION

Subroutine Name: DEG2RAD
Operation: $\text{FPACC1} \times \pi / 180 \blacklozenge \text{FPACC1}$
Size: 15 Bytes
Stack Space: 16 Bytes
Calls: GETFPAC2, FLTMUL
Input: Any valid floating-point number representing an angle in degrees.
Output: Input angles equivalent in radians.
Error Conditions: None

RADIANS TO DEGREES CONVERSION

Subroutine Name: RAD2DEG
Operation: $\text{FPACC1} \times 180 / \pi \blacklozenge \text{FPACC1}$
Size: 8 Bytes (Also requires DEG2RAD subroutine)
Stack Space: 16 Bytes
Calls: DEG2RAD
Input: Any valid floating-point number representing an angle in radians.
Output: Input angles equivalent in degrees.
Error Conditions: Overflow, Underflow.

PI

Subroutine Name: GETPI
Operation: $\pi \blacklozenge \text{FPACC1}$
Size: 6 Bytes
Stack Space: None
Input: None
Output: The value of π is returned in FPACC1.
Error Conditions: None
Notes: This routine should be used to obtain the value of π if it is required in calculations since it is accurate to the full 24 bits of the mantissa.

FORMAT CONVERSION ROUTINES

As discussed in **FLOATING-POINT ACCUMULATOR FORMAT** and **MEMORY FORMAT**, the format for floating-point numbers as they appear in the floating-point accumulators is different than the way numbers are stored in memory. This was done primarily to save memory when a large number of floating-point variables are used in a program. Four routines are provided to convert to and from the different formats while at the same time moving a number into or out of the floating-point accumulators. By always using these routines to move num-

bers into and out of the floating-point accumulators, it would be extremely easy to adapt this floating-point package to work with any other floating-point format.

One example might be to interface this package with code produced by Motorola's 68HC11 'C' compiler. The Motorola 'C' compiler generates code for single-precision floating-point numbers whose internal format is that defined by the *IEEE Standard for Binary Floating-Point Arithmetic*. By rewriting the four routines described below the IEEE format could be easily converted to the format required by this floating-point package.

Get FPACC(x)

Subroutine Name: GETFPAC1 and GETFPAC2
Operation: (X) ♦ FPACC1; (X) ♦ FPACC2
Size: 22 Bytes each
Stack Space: None
Input: The X index register points to the 'memory formatted' number to be moved into the floating-point accumulator.
Output: The number pointed to by X is in the specified floating-point accumulator.
Error Conditions: None

Put FPACC(x)

Subroutine Name: PUTFPAC1 and PUTFPAC2
Operation: FPACC1 ♦ (X); FPACC2 ♦ (X)
Size: 22 Bytes each
Stack Space: None
Input: The X index register points to four consecutive memory locations where the number will be stored.
Output: The floating point accumulator is moved into consecutive memory locations pointed to by the X index register.
Error Conditions: None


```

0047          TTL      ASCFLT
0048          *
0049          *
0050          *          ASCII TO FLOATING POINT ROUTINE          *
0051          *
0052          *          This routine will accept most any ASCII floating point format
0053          *          and return a 32-bit floating point number. The following are
0054          *          some examples of legal ASCII floating point numbers.
0055          *
0056          *          20.095
0057          *          0.125
0058          *          7.2984E10
0059          *          167.824E5
0060          *          5.9357E-7
0061          *          500
0062          *
0063          *          The floating point number returned is in "FPACC1".
0064          *
0065          *
0066          *          The exponent is biased by 128 to facilitate floating point
0067          *          comparisons. A pointer to the ASCII string is passed to the
0068          *          routine in the D-register.
0069          *
0070          *
0071          *
0072          *
0073          *
0074          *          ORG      $0000
0075          *
0076          *          FPACC1EX RMB 1          FLOATING POINT ACCUMULATOR #1..
0077          *          FPACC1MN RMB 3
0078          *          MANTSGN1 RMB 1          MANTISSA SIGN FOR FPACC1 (0=+, FF=-).
0079          *          FPACC2EX RMB 1          FLOATING POINT ACCUMULATOR #2.
0080          *          FPACC2MN RMB 3
0081          *          MANTSGN2 RMB 1          MANTISSA SIGN FOR FPACC2 (0=+, FF=-).
0082          *
0083          *
0084          *          FLTFMTER EQU 1
0085          *
0086          *
0087          *          LOCAL VARIABLES (ON STACK POINTED TO BY Y)
0088          *
0089          0000          EXPSIGN EQU 0          EXPONENT SIGN (0=+, FF=-).
0090          0001          PWR10EXP EQU 1          POWER 10 EXPONENT.
0091          *
0092          *
0093          0000          ORG      $C000          (TEST FOR EVB)
0094          *
0095          0000          ASCFLT EQU *
0096          0000 3C          PSHX          SAVE POINTER TO ASCII STRING.
0097          0001 8D C8 39          JSR PSHFPAC2          SAVE FPACC2.
0098          0004 CE 00 00          LDX #0          PUSH ZEROS ON STACK TO INITIALIZE LOCALS.
0099          0007 3C          PSHX          ALLOCATE 2 BYTES FOR LOCALS.
0100          0008 DF 00          STX FPACC1EX          CLEAR FPACC1.
0101          000A DF 02          STX FPACC1EX+2
0102          000C 7F 00 04          CLR MANTSGN1          MAKE THE MANTISSA SIGN POSITIVE INITIALLY.
0103          000F 18 30          TSY          POINT TO LOCALS.
0104          0011 CD EE 06          LDX 6,Y          GET POINTER TO ASCII STRING.
0105          0014 A6 00          ASCFLT1 LDAA 0,X          GET 1ST CHARACTER IN STRING.
0106          0016 8D C1 55          JSR NUMERIC          IS IT A NUMBER.
0107          0019 25 28          BCS ASCFLT4          YES. GO PROCESS IT.
0108          *
0109          *          LEADING MINUS SIGN ENCOUNTERED?
0110          *
0111          001B 81 2D          ASCFLT2 CMPA #-          NO. IS IT A MINUS SIGN?

```

0112 C01D 26 08	BNE	ASCFLT3	NO. GO CHECK FOR DECIMAL POINT.
0113 C01F 73 00 04	COM	MANTSGN1	YES. SET MANTISSA SIGN. LEADING MINUS BEFORE?
0114 C022 08	INX		POINT TO NEXT CHARACTER.
0115 C023 A6 00	LDAA	0,X	GET IT.
0116 C025 8D C1 55	JSR	NUMERIC	IS IT A NUMBER?
0117 C028 25 19	BCS	ASCFLT4	YES. GO PROCESS IT.
0118	*		
0119	*	LEADING DECIMAL POINT?	
0120	*		
0121			
0122 C02A 81 2E	ASCFLT3	CMPA #'.	IS IT A DECIMAL POINT?
0123 C02C 26 08	BNE	ASCFLT5	NO. FORMAT ERROR.
0124 C02E 08	INX		YES. POINT TO NEXT CHARACTER.
0125 C02F A6 00	LDAA	0,X	GET IT.
0126 C031 8D C1 55	JSR	NUMERIC	MUST HAVE AT LEAST ONE DIGIT AFTER D.P.
0127 C034 24 03	BCC	ASCFLT5	GO REPORT ERROR.
0128 C036 7E C0 C1	JMP	ASCFLT11	GO BUILD FRACTION.
0129	*		
0130	*	FLOATING POINT FORMAT ERROR	
0131	*		
0132 C039 31	ASCFLT5	INS	DE-ALLOCATE LOCALS.
0133 C03A 31	INS		
0134 C03B 8D C8 43	JSR	PULFPAC2	RESTORE FPACC2.
0135 C03E 38	PULX		GET POINTER TO TERMINATING CHARACTER IN STRING.
0136 C03F 86 01	LDAA	#FLTFMTER	FORMAT ERROR.
0137 C041 0D	SEC		SET ERROR FLAG.
0138 C042 39	RTS		RETURN.
0139	*		
0140	*	PRE DECIMAL POINT MANTISSA BUILD	
0141	*		
0142 C043 A6 00	ASCFLT4	LDAA 0,X	
0143 C045 8D C1 55	JSR	NUMERIC	
0144 C048 24 72	BCC	ASCFLT10	
0145 C04A 8D C0 D2	JSR	ADDNXTD	
0146 C04D 08	INX		
0147 C04E 24 F3	BCC	ASCFLT4	
0148	*		
0149	*	PRE DECIMAL POINT MANTISSA OVERFLOW	
0150	*		
0151 C050 7C 00 00	ASCFLT6	INC FPACC1EX	INC FOR EACH DIGIT ENCOUNTERED PRIOR TO D.P.
0152 C053 A6 00	LDAA	0,X	GET NEXT CHARACTER.
0153 C055 08	INX		POINT TO NEXT.
0154 C056 8D C1 55	JSR	NUMERIC	IS IT S DIGIT?
0155 C059 25 F5	BCS	ASCFLT6	YES. KEEP BUILDING POWER 10 MANTISSA.
0156 C05B 81 2E	CMPA	#'.	NO. IS IT A DECIMAL POINT?
0157 C05D 26 0A	BNE	ASCFLT7	NO. GO CHECK FOR THE EXPONENT.
0158	*		
0159	*	ANY FRACTIONAL DIGITS ARE NOT SIGNIFIGANT	
0160	*		
0161 C05F A6 00	ASCFLT8	LDAA 0,X	GET THE NEXT CHARACTER.
0162 C061 8D C1 55	JSR	NUMERIC	IS IT A DIGIT?
0163 C064 24 03	BCC	ASCFLT7	NO. GO CHECK FOR AN EXPONENT.
0164 C066 08	INX		POINT TO THE NEXT CHARACTER.
0165 C067 20 F6	BRA	ASCFLT8	FLUSH REMAINING DIGITS.
0166 C069 81 45	ASCFLT7	CMPA #'E	NO. IS IT THE EXPONENT?
0167 C06B 27 03	BEQ	ASCFLT13	YES. GO PROCESS IT.
0168 C06D 7E C1 17	JMP	FINISH	NO. GO FINISH THE CONVERSION.
0169	*		
0170	*	PROCESS THE EXPONENT	
0171	*		
0172 C070 08	ASCFLT13	INX	POINT TO NEXT CHARACTER.
0173 C071 A6 00	LDAA	0,X	GET THE NEXT CHARACTER.
0174 C073 8D C1 55	JSR	NUMERIC	SEE IF IT'S A DIGIT.
0175 C076 25 15	BCS	ASCFLT9	YES. GET THE EXPONENT.
0176 C078 81 2D	CMPA	#'-	NO. IS IT A MINUS SIGN?
0177 C07A 27 06	BEQ	ASCFLT15	YES. GO FLAG A NEGATIVE EXPONENT.

0178 C07C 81 2B	CMPA	#1+	NO. IS IT A PLUS SIGN?
0179 C07E 27 05	BEQ	ASCFLT6	YES. JUST IGNORE IT.
0180 C080 20 87	BRA	ASCFLT5	NO. FORMAT ERROR.
0181 C082 18 63 00	ASCFLT15 COM	EXPSIGN,Y	FLAG A NEGATIVE EXPONENT. IS IT 1ST?
0182 C085 08	ASCFLT16 INX		POINT TO NEXT CHARACTER.
0183 C086 A6 00	LDAA	0,X	GET NEXT CHARACTER.
0184 C088 8D C1 55	JSR	NUMERIC	IS IT A NUMBER?
0185 C088 24 AC	BCC	ASCFLT5	NO. FORMAT ERROR.
0186 C08D 80 30	ASCFLT9 SUBA	#S30	MAKE IT BINARY.
0187 C08F 18 A7 01	STAA	PWR10EXP,Y	BUILD THE POWER 10 EXPONENT.
0188 C092 08	INX		POINT TO NEXT CHARACTER.
0189 C093 A6 00	LDAA	0,X	GET IT.
0190 C095 8D C1 55	JSR	NUMERIC	IS IT NUMERIC?
0191 C098 24 13	BCC	ASCFLT14	NO. GO FINISH UP THE CONVERSION.
0192 C09A 18 E6 01	LDAB	PWR10EXP,Y	YES. GET PREVIOUS DIGIT.
0193 C09D 58	LSLB		MULT. BY 2.
0194 C09E 58	LSLB		NOW BY 4.
0195 C09F 18 EB 01	ADDB	PWR10EXP,Y	BY 5.
0196 C0A2 58	LSLB		BY 10.
0197 C0A3 80 30	SUBA	#S30	MAKE SECOND DIGIT BINARY.
0198 C0A5 18	ABA		ADD IT TO FIRST DIGIT.
0199 C0A6 18 A7 01	STAA	PWR10EXP,Y	
0200 C0A9 81 26	CMPA	#38	IS THE EXPONENT OUT OF RANGE?
0201 C0AB 22 8C	BHI	ASCFLT5	YES. REPORT ERROR.
0202 C0AD 18 A6 01	ASCFLT14 LDAA	PWR10EXP,Y	GET POWER 10 EXPONENT.
0203 C0B0 18 6D 00	TST	EXPSIGN,Y	WAS IT NEGATIVE?
0204 C0B3 2A 01	BPL	ASCFLT12	NO. GO ADD IT TO BUILT 10 PWR EXPONENT.
0205 C0B5 40	NEGA		
0206 C0B6 98 00	ASCFLT12 ADDA	FPACC1EX	FINAL TOTAL PWR 10 EXPONENT.
0207 C0B8 97 00	STAA	FPACC1EX	SAVE RESULT.
0208 C0BA 20 58	BRA	FINISH	GO FINISH UP CONVERSION.
0209	*		
0210	*	PRE-DECIMAL POINT NON-DIGIT FOUND, IS IT A DECIMAL POINT?	
0211	*		
0212 C0BC 81 2E	ASCFLT10 CMPA	#1.	IS IT A DECIMAL POINT?
0213 C0BE 26 A9	BNE	ASCFLT7	NO. GO CHECK FOR THE EXPONENT.
0214 C0C0 08	INX		YES. POINT TO NEXT CHARACTER.
0215	*		
0216	*	POST DECIMAL POINT PROCESSING	
0217	*		
0218 C0C1 A6 00	ASCFLT11 LDAA	0,X	GET NEXT CHARACTER.
0219 C0C3 8D C1 55	JSR	NUMERIC	IS IT NUMERIC?
0220 C0C6 24 A1	BCC	ASCFLT7	NO. GO CHECK FOR EXPONENT.
0221 C0C8 8D 08	BSR	ADDNXTD	YES. ADD IN THE DIGIT.
0222 C0CA 08	INX		POINT TO THE NEXT CHARACTER.
0223 C0CB 25 92	BCS	ASCFLT8	IF OVER FLOW, FLUSH REMAINING DIGITS.
0224 C0CD 7A 00 00	DEC	FPACC1EX	ADJUST THE 10 POWER EXPONENT.
0225 C0D0 20 EF	BRA	ASCFLT11	PROCESS ALL FRACTIONAL DIGITS.
0226	*		
0227	*		
0228	*		
0229 C0D2 96 01	ADDNXTD LDAA	FPACC1MN	GET UPPER 8 BITS.
0230 C0D4 97 06	STAA	FPACC2MN	COPY INTO FPAC2.
0231 C0D6 DC 02	LDD	FPACC1MN+1	GET LOWER 16 BITS OF MANTISSA.
0232 C0D8 DD 07	STD	FPACC2MN+1	COPY INTO FPACC2.
0233 C0DA 05	LSLD		MULT. BY 2.
0234 C0DB 79 00 01	ROL	FPACC1MN	OVERFLOW?
0235 C0DE 25 2E	BCS	ADDNXTD1	YES. DON'T ADD THE DIGIT IN.
0236 C0E0 05	LSLD		MULT BY 4.
0237 C0E1 79 00 01	ROL	FPACC1MN	OVERFLOW?
0238 C0E4 25 28	BCS	ADDNXTD1	YES. DON'T ADD THE DIGIT IN.
0239 C0E6 D3 07	ADDD	FPACC2MN+1	BY 5.
0240 C0E8 36	PSHA		SAVE A.
0241 C0E9 96 01	LDAA	FPACC1MN	GET UPPER 8 BITS.
0242 C0EB 89 00	ADCA	#0	ADDIN POSSABLE CARRY FROM LOWER 16 BITS.
0243 C0ED 98 06	ADDA	FPACC2MN	ADD IN UPPER 8 BITS.

0244	COEF 97 01	STAA	FPACC1MN	SAVE IT.	
0245	COF1 32	PULA		RESTORE A.	
0246	COF2 25 1A	BCS	ADDNXTD1	OVERFLOW? IF SO DON'T ADD IT IN.	
0247	COF4 05	LSDL		BY 10.	
0248	COF5 79 00 01	ROL	FPACC1MN		
0249	COF8 00 02	STD	FPACC1MN+1	SAVE THE LOWER 16 BITS.	
0250	COFA 25 12	BCS	ADDNXTD1	OVERFLOW? IF SO DON'T ADD IT IN.	
0251	COFC E6 00	LDAB	0,X	GET CURRENT DIGIT.	
0252	COFE C0 30	SUBB	#S30	MAKE IT BINARY.	
0253	C100 4F	CLRA		16-BIT.	
0254	C101 03 02	ADDD	FPACC1MN+1	ADD IT IN TO TOTAL.	
0255	C103 00 02	STD	FPACC1MN+1	SAVE THE RESULT.	
0256	C105 96 01	LDAA	FPACC1MN	GET UPPER 8 BITS.	
0257	C107 89 00	ADCA	#0	ADD IN POSSIBLE CARRY. OVERFLOW?	
0258	C109 25 03	BCS	ADDNXTD1	YES. COPY OLD MANTISSA FROM FPACC2.	
0259	C108 97 01	STAA	FPACC1MN	NO. EVERYTHING OK.	
0260	C100 39	RTS		RETURN.	
0261	C10E DC 07	ADDNXTD1	LDD	FPACC2MN+1	RESTORE THE ORIGINAL MANTISSA BECAUSE
0262	C110 00 02	STD	FPACC1MN+1		OF OVERFLOW.
0263	C112 96 06	LDAA	FPACC2MN		
0264	C114 97 01	STAA	FPACC1MN		
0265	C116 39	RTS		RETURN.	
0266		*			
0267		*			
0268		*			
0269		*			
0270		*			
0271		*			
0272		*			
0273		*			
0274	C117	FINISH	EQU	*	
0275	C117 CD EF 06	STX	6,Y	SAVE POINTER TO TERMINATING CHARACTER IN STRING.	
0276	C11A CE 00 00	LDX	#FPACC1EX	POINT TO FPACC1.	
0277	C110 8D C1 80	JSR	CHCK0	SEE IF THE NUMBER IS ZERO.	
0278	C120 27 2C	BEQ	FINISH3	QUIT IF IT IS.	
0279	C122 96 00	LDAA	FPACC1EX	GET THE POWER 10 EXPONENT.	
0280	C124 18 A7 01	STAA	PWR10EXP,Y	SAVE IT.	
0281	C127 86 98	LDAA	#S80+24	SET UP INITIAL EXPONENT (# OF BITS + BIAS).	
0282	C129 97 00	STAA	FPACC1EX		
0283	C128 8D C1 61	JSR	FPNORM	GO NORMALIZE THE MANTISSA.	
0284	C12E 18 6D 01	TST	PWR10EXP,Y	IS THE POWER 10 EXPONENT POSITIVE OR ZERO?	
0285	C131 27 18	BEQ	FINISH3	IT'S ZERO, WE'RE DONE.	
0286	C133 2A 08	BPL	FINISH1	IT'S POSITIVE MULTIPLY BY 10.	
0287	C135 CE C1 88	LDX	#CONSTP1	NO. GET CONSTANT .1 (DIVIDE BY 10).	
0288	C138 8D C8 66	JSR	GETFPAC2	GET CONSTANT INTO FPACC2.	
0289	C13B 18 6D 01	NEG	PWR10EXP,Y	MAKE THE POWER 10 EXPONENT POSITIVE.	
0290	C13E 20 06	BRA	FINISH2	GO DO THE MULTIPLIES.	
0291	C140 CE C1 8F	FINISH1	LDX	#CONST10	GET CONSTANT '10' TO MULTIPLY BY.
0292	C143 8D C8 66	JSR	GETFPAC2	GET CONSTANT INTO FPACC2.	
0293	C146 8D C1 93	FINISH2	JSR	FLTMUL	GO MULTIPLY FPACC1 BY FPACC2, RESULT IN FPACC1.
0294	C149 18 6A 01	DEC	PWR10EXP,Y	DECREMENT THE POWER 10 EXPONENT.	
0295	C14C 26 F8	BNE	FINISH2	GO CHECK TO SEE IF WE'RE DONE.	
0296	C14E 31	FINISH3	INS	DE-ALLOCATE LOCALS.	
0297	C14F 31	INS			
0298	C150 8D C8 43	JSR	PULFPAC2	RESTORE FPACC2.	
0299	C153 38	PULX		GET POINTER TO TERMINATING CHARACTER IN STRING.	
0300	C154 39	RTS		RETURN WITH NUMBER IN FPACC1.	
0301		*			
0302		*			
0303	C155	NUMERIC	EQU	*	
0304	C155 81 30	CMPA	#10	IS IT LESS THAN AN ASCII 0?	
0305	C157 25 06	BLO	NUMERIC1	YES. NOT NUMERIC.	
0306	C159 81 39	CMPA	#19	IS IT GREATER THAN AN ASCII 9?	
0307	C15B 22 02	BHI	NUMERIC1	YES. NOT NUMERIC.	
0308	C15D 00	SEC		IT WAS NUMERIC. SET THE CARRY.	
0309	C15E 39	RTS		RETURN.	

0310 C15F 0C	NUMERIC1 CLC		NON-NUMERIC CHARACTER. CLEAR THE CARRY.
0311 C160 39	RTS		RETURN.
0312	*		
0313 C161	FPNORM EQU *		
0314 C161 CE 00 00	LDX #FPACC1EX		POINT TO FPACC1.
0315 C166 8D 1A	BSR CHCK0		CHECK TO SEE IF IT'S 0.
0316 C166 27 14	BEQ FPNORM3		YES. JUST RETURN.
0317 C168 7D 00 01	TST FPACC1MN		IS THE NUMBER ALREADY NORMALIZED?
0318 C168 2B 0F	BMI FPNORM3		YES. JUST RETURN..
0319 C160 DC 02	FPNORM1 LDD FPACC1MN+1		GET THE LOWER 16 BITS OF THE MANTISSA.
0320 C16F 7A 00 00	FPNORM2 DEC FPACC1EX		DECREMENT THE EXPONENT FOR EACH SHIFT.
0321 C172 27 0A	BEQ FPNORM4		EXPONENT WENT TO 0. UNDERFLOW.
0322 C174 05	LSLD		SHIFT THE LOWER 16 BITS.
0323 C175 79 00 01	ROL FPACC1MN		ROTATE THE UPPER 8 BITS. NUMBER NORMALIZED?
0324 C178 2A F5	BPL FPNORM2		NO. KEEP SHIFTING TO THE LEFT.
0325 C17A DD 02	STD FPACC1MN+1		PUT THE LOWER 16 BITS BACK INTO FPACC1.
0326 C17C 0C	FPNORM3 CLC		SHOW NO ERRORS.
0327 C17D 39	RTS		YES. RETURN.
0328 C17E 0D	FPNORM4 SEC		FLAG ERROR.
0329 C17F 39	RTS		RETURN.
0330	*		
0331 C180	CHCK0 EQU *		CHECKS FOR ZERO IN FPACC POINTED TO BY X.
0332 C180 37	PSHB		SAVE D.
0333 C181 36	PSHA		
0334 C182 EC 00	LDD 0,X		GET FPACC EXPONENT & HIGH 8 BITS.
0335 C184 26 02	BNE CHCK01		NOT ZERO. RETURN.
0336 C186 EC 02	LDD 2,X		CHECK LOWER 16 BITS.
0337 C188 32	CHCK01 PULA		RESTORE D.
0338 C189 33	PULB		
0339 C18A 39	RTS		RETURN WITH CC SET.
0340	*		
0341 C188 7D 4C CC CD	CONSTP1 FCB \$7D,\$4C,\$CC,\$CD		0.1 DECIMAL
0342 C18F 84 20 00 00	CONST10 FCB \$84,\$20,\$00,\$00		10.0 DECIMAL
0343	*		
0344	*		


```

0345                                TTL   FLTMUL
0346                                *****
0347                                *
0348                                *                               FPMULT: FLOATING POINT MULTIPLY
0349                                *
0350                                *   THIS FLOATING POINT MULTIPLY ROUTINE MULTIPLIES "FPACC1" BY
0351                                *   "FPACC2" AND PLACES THE RESULT IN TO FPACC1. FPACC2 REMAINS
0352                                *   UNCHANGED.
0353                                *                               WORSE CASE = 2319 CYCLES = 1159 US @ 2MHZ
0354                                *
0355                                *****
0356                                *
0357                                *
0358 C193                          FLTMUL EQU *
0359 C193 BD C8 39                  JSR   PSHFPACC2   SAVE FPACC2.
0360 C196 CE 00 00                  LDX   #FPACC1EX  POINT TO FPACC1
0361 C199 BD C1 80                  JSR   CHCKO      CHECK TO SEE IF FPACC1 IS ZERO.
0362 C19C 27 31                    BEQ   FPMULT3    IT IS. ANSWER IS 0.
0363 C19E CE 00 05                  LDX   #FPACC2EX  POINT TO FPACC2.
0364 C1A1 8D C1 80                  JSR   CHCKO      IS IT 0?
0365 C1A4 26 08                    BNE   FPMULT4    NO. CONTINUE.
0366 C1A6 4F                        CLRA
0367 C1A7 5F                        CLRB
0368 C1A8 DD 00                    STD   FPACC1EX   MAKE FPACC1 0.
0369 C1AA DD 02                    STD   FPACC1MN+1
0370 C1AC 20 21                    BRA   FPMULT3    RETURN.
0371 C1AE 96 04                    FPMULT4 LDAA  MANTSGN1  GET FPACC1 EXPONENT.
0372 C1B0 98 09                    EORA  MANTSGN2   SET THE SIGN OF THE RESULT.
0373 C1B2 97 04                    STAA  MANTSGN1   SAVE THE SIGN OF THE RESULT.
0374 C1B4 96 00                    LDAA  FPACC1EX   GET FPACC1 EXPONENT.
0375 C1B6 98 05                    ADDA  FPACC2EX   ADD IT TO FPACC2 EXPONENT.
0376 C1B8 2A 07                    BPL   FPMULT1    IF RESULT IS MINUS AND
0377 C1BA 24 0C                    BCC   FPMULT2    THE CARRY IS SET THEN:
0378 C1BC 86 02                    FPMULT5 LDAA  #OVFERR  OVERFLOW ERROR.
0379 C1BE 0D                        SEC
0380 C1BF 20 14                    BRA   FPMULT6    RETURN.
0381 C1C1 25 05                    FPMULT1 BCS   FPMULT2  IF RESULT IS PLUS & THE CARRY IS SET THEN ALL OK.
0382 C1C3 86 03                    LDAA  #UNFERR    ELSE UNDERFLOW ERROR OCCURED.
0383 C1C5 0D                        SEC
0384 C1C6 20 0D                    BRA   FPMULT6    RETURN.
0385 C1C8 88 80                    FPMULT2 ADDA  #S80    ADD 128 BIAS BACK IN THAT WE LOST.
0386 C1CA 97 0D                    STAA  FPACC1EX   SAVE THE NEW EXPONENT.
0387 C1CC 8D C1 D9                  JSR   UMULT      GO MULTIPLY THE "INTEGER" MANTISSAS.
0388 C1CF 7D 00 00                  FPMULT3 TST   FPACC1EX  WAS THERE AN OVERFLOW ERROR FROM ROUNDING?
0389 C1D2 27 E8                    BEQ   FPMULT5    YES. RETURN ERROR.
0390 C1D4 0C                        CLC
0391 C1D5 8D C8 43                  FPMULT6 JSR   PULFPACC2  RESTORE FPACC2.
0392 C1D8 39                        RTS
0393                                *
0394                                *
0395 C1D9                          UMULT EQU *
0396 C1D9 CE 00 00                  LDX   #0
0397 C1DC 3C                        PSHX
0398 C1DD 3C                        PSHX
0399 C1DE 30                        TSX
0400 C1DF 86 18                    LDAA  #24        SET COUNT TO THE NUMBER OF BITS.
0401 C1E1 A7 00                    STAA  0,X
0402 C1E3 96 08                    UMULT1 LDAA  FPACC2MN+2  GET THE L.S. BYTE OF THE MULTIPLIER.
0403 C1E5 44                        LSR
0404 C1E6 24 0C                    BCC   UMULT2    IF CARRY CLEAR, DON'T ADD MULTIPLICAND TO P.P.
0405 C1E8 DC 02                    LDD  FPACC1MN+1  GET MULTIPLICAND L.S. 16 BITS.
0406 C1EA E3 02                    ADDD  2,X        ADD TO PARTIAL PRODUCT.
0407 C1EC ED 02                    STD  2,X        SAVE IN P.P.
0408 C1EE 96 01                    LDAA  FPACC1MN   GET UPPER 8 BITS OF MULTIPLICAND.
0409 C1F0 A9 01                    ADCA  1,X        ADD IT W/ CARRY TO P.P.

```

0410 C1F2 A7 01	STAA	1,X	SAVE TO PARTIAL PRODUCT.
0411 C1F4 66 01	UMULT2 ROR	1,X	ROTATE PARTIAL PRODUCT TO THE RIGHT.
0412 C1F6 66 02	ROR	2,X	
0413 C1F8 66 03	ROR	3,X	
0414 C1FA 76 00 06	ROR	FPACC2MN	SHIFT THE MULTIPLIER TO THE RIGHT 1 BIT.
0415 C1FD 76 00 07	ROR	FPACC2MN+1	
0416 C200 76 00 08	ROR	FPACC2MN+2	
0417 C203 6A 00	DEC	0,X	DONE YET?
0418 C205 26 DC	BNE	UMULT1	NO. KEEP GOING.
0419 C207 60 01	TST	1,X	DOES PARTIAL PRODUCT NEED TO BE NORMALIZED?
0420 C209 2B 0C	BMI	UMULT3	NO. GET ANSWER & RETURN.
0421 C20B 78 00 06	LSL	FPACC2MN	GET BIT THAT WAS SHIFTED OUT OF P.P REGISTER.
0422 C20E 69 03	ROL	3,X	PUT IT BACK INTO THE PARTIAL PRODUCT.
0423 C210 69 02	ROL	2,X	
0424 C212 69 01	ROL	1,X	
0425 C214 7A 00 00	DEC	FPACC1EX	FIX EXPONENT.
0426 C217 7D 00 06	UMULT3 TST	FPACC2MN	DO WE NEED TO ROUND THE PARTIAL PRODUCT?
0427 C21A 2A 18	BPL	UMULT4	NO. JUST RETURN.
0428 C21C EC 02	LDD	2,X	YES. GET THE LEAST SIGNIFIGANT 16 BITS.
0429 C21E C3 00 01	ADDD	#1	ADD 1.
0430 C221 ED 02	STD	2,X	SAVE RESULT.
0431 C223 A6 01	LDAA	1,X	PROPIGATE THROUGH.
0432 C225 89 00	ADCA	#0	
0433 C227 A7 01	STAA	1,X	
0434 C229 24 09	BCC	UMULT4	IF CARRY CLEAR ALL IS OK.
0435 C22B 66 01	ROR	1,X	IF NOT OVERFLOW. ROTATE CARRY INTO P.P.
0436 C22D 66 02	ROR	2,X	
0437 C22F 66 03	ROR	3,X	
0438 C231 7C 00 00	INC	FPACC1EX	UP THE EXPONENT.
0439 C234 31	UMULT4 INS		TAKE COUNTER OFF STACK.
0440 C235 38	PULX		GET M.S. 16 BITS OF PARTIAL PRODUCT.
0441 C236 DF 01	STX	FPACC1MN	PUT IT IN FPACC1.
0442 C238 32	PULA		GET L.S. 8 BITS OF PARTIAL PRODUCT.
0443 C239 97 03	STAA	FPACC1MN+2	PUT IT IN FPACC1.
0444 C23B 39	RTS		RETURN.
0445	*		
0446	*		
0447	*		

```

0448                               TTL   FLTADD
0449                               *****
0450                               *
0451                               *           FLOATING POINT ADDITION           *
0452                               *
0453                               *   This subroutine performs floating point addition of the two numbers *
0454                               *   in FPACC1 and FPACC2. The result of the addition is placed in   *
0455                               *   FPACC1 while FPACC2 remains unchanged. This subroutine performs *
0456                               *   full signed addition so either number may be of the same or opposite *
0457                               *   sign.
0458                               *
0459                               *           WORSE CASE = 1030 CYCLES = 515 uS @ 2MHz           *
0460                               *
0461                               *
0462                               *
0463 C23C                               FLTADD EQU *
0464 C23C BD C8 39                               JSR PSHFPAC2   SAVE FPACC2.
0465 C23F CE 00 05                               LDX #FPACC2EX POINT TO FPACC2
0466 C242 BD C1 80                               JSR CHK0      IS IT ZERO?
0467 C245 26 05                               BNE FLTADD1   NO. GO CHECK FOR 0 IN FPACC1.
0468 C247 0C                               FLTADD6 CLC   NO ERRORS.
0469 C248 BD C8 43                               FLTADD10 JSR PULFPAC2 RESTORE FPACC2.
0470 C248 39                               RTS          ANSWER IN FPACC1. RETURN.
0471 C24C CE 00 00                               FLTADD1 LDX #FPACC1EX POINT TO FPACC1.
0472 C24F BD C1 80                               JSR CHK0      IS IT ZERO?
0473 C252 26 0E                               BNE FLTADD2   NO. GO ADD THE NUMBER.
0474 C254 DC 05                               FLTADD4 LDD FPACC2EX ANSWER IS IN FPACC2. MOVE IT INTO FPACC1.
0475 C256 DD 00                               STD FPACC1EX
0476 C258 DC 07                               LDD FPACC2MN+1 MOVE LOWER 16 BITS OF MANTISSA.
0477 C25A DD 02                               STD FPACC1MN+1
0478 C25C 96 09                               LDAA MANTSGN2 MOVE FPACC2 MANTISSA SIGN INTO FPACC1.
0479 C25E 97 04                               STAA MANTSGN1
0480 C260 20 E5                               BRA FLTADD6   RETURN.
0481 C262 96 00                               FLTADD2 LDAA FPACC1EX GET FPACC1 EXPONENT.
0482 C264 91 05                               CMPA FPACC2EX ARE THE EXPONENTS THE SAME?
0483 C266 27 23                               BEQ FLTADD7   YES. GO ADD THE MANTISSA'S.
0484 C268 90 05                               SUBA FPACC2EX NO. FPACC1EX-FPACC2EX. IS FPACC1 > FPACC2?
0485 C26A 2A 0F                               BPL FLTADD3   YES. GO CHECK RANGE.
0486 C26C 40                               NEGA         NO. FPACC1 < FPACC2. MAKE DIFFERENCE POSITIVE.
0487 C26D 81 17                               CMPA #23     ARE THE NUMBERS WITHIN RANGE?
0488 C26F 22 E3                               BHI FLTADD4   NO. FPACC2 IS LARGER. GO MOVE IT INTO FPACC1.
0489 C271 16                               TAB          PUT DIFFERENCE IN B.
0490 C272 08 00                               ADOB FPACC1EX CORRECT FPACC1 EXPONENT.
0491 C274 07 00                               STAB FPACC1EX SAVE THE RESULT.
0492 C276 CE 00 01                               LDX #FPACC1MN POINT TO FPACC1 MANTISSA.
0493 C279 20 07                               BRA FLTADD5   GO DENORMALIZE FPACC1 FOR THE ADD.
0494 C27B 81 17                               FLTADD3 CMPA #23     FPACC1 > FPACC2. ARE THE NUMBERS WITHIN RANGE?
0495 C27D 22 C8                               BHI FLTADD6   NO. ANSWER ALREADY IN FPACC1. JUST RETURN.
0496 C27F CE 00 06                               LDX #FPACC2MN POINT TO THE MANTISSA TO DENORMALIZE.
0497 C282 64 00                               FLTADD5 LSR 0,X    SHIFT THE FIRST BYTE OF THE MANTISSA.
0498 C284 66 01                               ROR 1,X      THE SECOND.
0499 C286 66 02                               ROR 2,X      AND THE THIRD.
0500 C288 4A                               DECA        DONE YET?
0501 C289 26 F7                               BNE FLTADD5   NO. KEEP SHIFTING.
0502 C28B 96 04                               FLTADD7 LDAA MANTSGN1 GET FPACC1 MANTISSA SIGN.
0503 C28D 91 09                               CMPA MANTSGN2 ARE THE SIGNS THE SAME?
0504 C28F 27 48                               BEQ FLTADD11 YES. JUST GO ADD THE TWO MANTISSAS.
0505 C291 7D 00 04                               TST MANTSGN1 NO. IS FPACC1 THE NEGATIVE NUMBER?
0506 C294 2A 14                               BPL FLTADD8   NO. GO DO FPACC1-FPACC2.
0507 C296 DE 06                               LDX FPACC2MN YES. EXCHANGE FPACC1 & FPACC2 BEFORE THE SUB.
0508 C298 3C                               PSHX        SAVE IT.
0509 C299 DE 01                               LDX FPACC1MN GET PART OF FPACC1.
0510 C29B DF 06                               STX FPACC2MN PUT IT IN FPACC2.
0511 C29D 38                               PULX        GET SAVED PORTION OF FPACC2
0512 C29E DF 01                               STX FPACC1MN PUT IT IN FPACC1.

```

0513	C2A0	DE	08	LDX	FPACC2MN+2	GET LOWER 8 BITS & SIGN OF FPACC2.		
0514	C2A2	3C		PSHX		SAVE IT.		
0515	C2A3	DE	03	LDX	FPACC1MN+2	GET LOWER 8 BITS & SIGN OF FPACC1.		
0516	C2A5	DF	08	STX	FPACC2MN+2	PUT IT IN FPACC2.		
0517	C2A7	38		PULX		GET SAVED PART OF FPACC2.		
0518	C2A8	DF	03	STX	FPACC1MN+2	PUT IT IN FPACC1.		
0519	C2AA	DC	02	FLTADD8	LDD	FPACC1MN+1	GET LOWER 16 BITS OF FPACC1.	
0520	C2AC	93	07	SUBD	FPACC2MN+1	SUBTRACT LOWER 16 BITS OF FPACC2.		
0521	C2AE	DD	02	STD	FPACC1MN+1	SAVE RESULT.		
0522	C2B0	96	01	LDA	FPACC1MN	GET HIGH 8 BITS OF FPACC1 MANTISSA.		
0523	C2B2	92	06	SBCA	FPACC2MN	SUBTRACT HIGH 8 BITS OF FPACC2.		
0524	C2B4	97	01	STAA	FPACC1MN	SAVE THE RESULT. IS THE RESULT NEGATIVE?		
0525	C2B6	24	16	BCC	FLTADD9	NO. GO NORMALIZE THE RESULT.		
0526	C2B8	96	01	LDA	FPACC1MN	YES. NEGATE THE MANTISSA.		
0527	C2BA	43		COMA				
0528	C2BB	36		PSHA		SAVE THE RESULT.		
0529	C2BC	DC	02	LDD	FPACC1MN+1	GET LOWER 16 BITS.		
0530	C2BE	53		COMB		FORM THE ONE'S COMPLEMENT.		
0531	C2BF	43		COMA				
0532	C2C0	C3	00	ADD	#1	FORM THE TWO'S COMPLEMENT.		
0533	C2C3	DD	02	STD	FPACC1MN+1	SAVE THE RESULT.		
0534	C2C5	32		PULA		GET UPPER 8 BITS BACK.		
0535	C2C6	89	00	ADCA	#0	ADD IN POSSIBLE CARRY.		
0536	C2C8	97	01	STAA	FPACC1MN	SAVE RESULT.		
0537	C2CA	86	FF	LDA	#\$FF	SHOW THAT FPACC1 IS NEGATIVE.		
0538	C2CC	97	04	STAA	MANTSGN1			
0539	C2CE	BD	C1	61	FLTADD9	JSR	FPNORM	GO NORMALIZE THE RESULT.
0540	C2D1	24	06		BCC	FLTADD12	EVERYTHING'S OK SO RETURN.	
0541	C2D3	86	03		LDA	#UNFERR	UNDERFLOW OCCURED DURING NORMALIZATION.	
0542	C2D5	0D			SEC		FLAG ERROR.	
0543	C2D6	7E	C2	48		JMP	FLTADD10	RETURN.
0544	C2D9	7E	C2	47	FLTADD12	JMP	FLTADD6	CAN'T BRANCH THAT FAR FROM HERE.
0545				*				
0546	C2DC	DC	02	FLTADD11	LDD	FPACC1MN+1	GET LOWER 16 BITS OF FPACC1.	
0547	C2DE	D3	07	ADD	FPACC2MN+1	ADD IT TO THE LOWER 16 BITS OF FPACC2.		
0548	C2E0	DD	02	STD	FPACC1MN+1	SAVE RESULT IN FPACC1.		
0549	C2E2	96	01	LDA	FPACC1MN	GET UPPER 8 BITS OF FPACC1.		
0550	C2E4	99	06	ADCA	FPACC2MN	ADD IT (WITH CARRY) TO UPPER 8 BITS OF FPACC2.		
0551	C2E6	97	01	STAA	FPACC1MN	SAVE THE RESULT.		
0552	C2E8	24	EF	BCC	FLTADD12	NO OVERFLOW SO JUST RETURN.		
0553	C2EA	76	00	ROR	FPACC1MN	PUT THE CARRY INTO THE MANTISSA.		
0554	C2ED	76	00	ROR	FPACC1MN+1	PROPAGATE THROUGH MANTISSA.		
0555	C2F0	76	00	ROR	FPACC1MN+2			
0556	C2F3	7C	00	INC	FPACC1EX	UP THE MANTISSA BY 1.		
0557	C2F6	26	E1	BNE	FLTADD12	EVERYTHING'S OK JUST RETURN.		
0558	C2F8	86	02	LDA	#OVFERR	RESULT WAS TOO LARGE. OVERFLOW.		
0559	C2FA	0D		SEC		FLAG ERROR.		
0560	C2FB	7E	C2	48	JMP	FLTADD10	RETURN.	
0561				*				
0562				*				
0563				*				

```

0564                                TTL   FLTSUB
0565                                *****
0566                                *
0567                                *           FLOATING POINT SUBTRACT SUBROUTINE
0568                                *
0569                                *           This subroutine performs floating point subtraction ( FPACC1-FPACC2) *
0570                                *           by inverting the sign of FPACC2 and then calling FLTADD since
0571                                *           FLTADD performs complete signed addition. Upon returning from
0572                                *           FLTADD the sign of FPACC2 is again inverted to leave it unchanged
0573                                *           from its original value.
0574                                *
0575                                *           WORSE CASE = 1062 CYCLES = 531 uS @ 2MHz
0576                                *
0577                                *****
0578                                *
0579                                *
0580 C2FE                                FLTSUB EQU *
0581 C2FE 8D 03                                BSR FLTSUB1 INVERT SIGN.
0582 C300 8D C2 3C                            JSR FLTADD GO DO FLOATING POINT ADD.
0583 C303 96 09                                FLTSUB1 LDAA MANTSGN2 GET FPACC2 MANTISSA SIGN.
0584 C305 88 FF                                EORA #$FF INVERT THE SIGN.
0585 C307 97 09                                STAA MANTSGN2 PUT BACK.
0586 C309 39                                RTS RETURN.
0587                                *
0588                                *
0589                                *

```


0655 C35E 74 00 06	FLTDIV14 LSR	FPACC2MN	SHIFT THE DIVISOR TO THE RIGHT 1 BIT.
0656 C361 76 00 07	ROR	FPACC2MN+1	
0657 C364 76 00 08	ROR	FPACC2MN+2	
0658 C367 96 00	LDAA	FPACC1EX	GET FPACC1 EXPONENT.
0659 C369 D6 05	LDAB	FPACC2EX	GET FPACC2 EXPONENT.
0660 C36B 50	NEGB		ADD THE TWO'S COMPLEMENT TO SET FLAGS PROPERLY.
0661 C36C 18	ABA		
0662 C36D 28 06	BMI	FLTDIV5	IF RESULT MINUS CHECK CARRY FOR POSS. OVERFLOW.
0663 C36F 25 06	BCS	FLTDIV7	IF PLUS & CARRY SET ALL IS OK.
0664 C371 86 03	LDAA	#UNFERR	IF NOT, UNDERFLOW ERROR.
0665 C373 20 D3	BRA	FLTDIV6	RETURN WITH ERROR.
0666 C375 25 CE	FLTDIV5 BCS	FLTDIV8	IF MINUS & CARRY SET OVERFLOW ERROR.
0667 C377 88 B1	FLTDIV7 ADDA	#S81	ADD BACK BIAS+1 (THE '1' COMPENSATES FOR ALGOR.)
0668 C379 97 00	STAA	FPACC1EX	SAVE RESULT.
0669 C37B DC 01	FLTDIV9 LDD	FPACC1MN	SAVE DIVIDEND IN CASE SUBTRACTION DOESN'T GO.
0670 C37D ED 04	STD	4,X	
0671 C37F 96 03	LDAA	FPACC1MN+2	
0672 C381 A7 06	STAA	6,X	
0673 C383 DC 02	LDD	FPACC1MN+1	GET LOWER 16 BITS FOR SUBTRACTION.
0674 C385 93 07	SUBD	FPACC2MN+1	
0675 C387 DD 02	STD	FPACC1MN+1	SAVE RESULT.
0676 C389 96 01	LDAA	FPACC1MN	GET HIGH 8 BITS.
0677 C38B 92 06	SBCA	FPACC2MN	
0678 C38D 97 01	STAA	FPACC1MN	
0679 C38F 2A 08	BPL	FLTDIV10	SUBTRACTION WENT OK. GO DO SHIFTS.
0680 C391 EC 04	LDD	4,X	RESTORE OLD DIVIDEND.
0681 C393 DD 01	STD	FPACC1MN	
0682 C395 A6 06	LDAA	6,X	
0683 C397 97 03	STAA	FPACC1MN+2	
0684 C399 69 03	FLTDIV10 ROL	3,X	ROTATE CARRY INTO QUOTIENT.
0685 C39B 69 02	ROL	2,X	
0686 C39D 69 01	ROL	1,X	
0687 C39F 78 00 C3	LSL	FPACC1MN+2	SHIFT DIVIDEND TO LEFT FOR NEXT SUBTRACT.
0688 C3A2 79 00 C2	ROL	FPACC1MN+1	
0689 C3A5 79 00 C1	ROL	FPACC1MN	
0690 C3A8 6A 00	DEC	0,X	DONE YET?
0691 C3AA 26 CF	BNE	FLTDIV9	NO. KEEP GOING.
0692 C3AC 63 01	COM	1,X	RESULT MUST BE COMPLEMENTED.
0693 C3AE 63 02	COM	2,X	
0694 C3B0 63 03	COM	3,X	
0695 C3B2 DC 02	LDD	FPACC1MN+1	DO 1 MORE SUBTRACT FOR ROUNDING.
0696 C3B4 93 07	SUBD	FPACC2MN+1	(DON'T NEED TO SAVE THE RESULT.)
0697 C3B6 96 01	LDAA	FPACC1MN	
0698 C3B8 92 06	SBCA	FPACC2MN	(NO NEED TO SAVE THE RESULT.)
0699 C3BA EC 02	LDD	2,X	GET LOW 16 BITS.
0700 C3BC 24 03	BCC	FLTDIV11	IF IT DIDNT GO RESULT OK AS IS.
0701 C3BE 0C	CLC		CLEAR THE CARRY.
0702 C3BF 20 03	BRA	FLTDIV13	GO SAVE THE NUMBER.
0703 C3C1 C3 00 C1	FLTDIV11 ADDD	#1	ROUND UP BY 1.
0704 C3C4 DD C2	FLTDIV13 STD	FPACC1MN+1	PUT IT IN FPACC1.
0705 C3C6 A6 01	LDAA	1,X	GET HIGH 8 BITS.
0706 C3C8 89 0C	ADCA	#0	
0707 C3CA 97 01	STAA	FPACC1MN	SAVE RESULT.
0708 C3CC 24 09	BCC	FLTDIV12	IF CARRY CLEAR ANSWER OK.
0709 C3CE 76 00 C1	ROR	FPACC1MN	IF NOT OVERFLOW. ROTATE CARRY IN.
0710 C3D1 76 00 02	ROR	FPACC1MN+1	
0711 C3D4 76 00 03	ROR	FPACC1MN+2	
0712 C3D7 0C	FLTDIV12 CLC		NO ERRORS.
0713 C3D8 7E C3 4B	JMP	FLTDIV6	RETURN.
0714	*		
0715	*		
0716	*		

```

0717
0718
0719
0720
0721
0722
0723
0724
0725
0726
0727
0728
0729
0730
0731
0732
0733 C3DB
0734 C3DB 3C
0735 C3DC CE 00 00
0736 C3DF BD C1 80
0737 C3E2 26 07
0738 C3E4 38
0739 C3E5 CC 30 00
0740 C3E8 ED 00
0741 C3EA 39
0742 C3EB DE 00
0743 C3ED 3C
0744 C3EE DE 02
0745 C3F0 3C
0746 C3F1 96 04
0747 C3F3 36
0748 C3F4 BD C8 39
0749 C3F7 CE 00 00
0750 C3FA 3C
0751 C3FB 3C
0752 C3FC 3C
0753 C3FD 18 30
0754 C3FF CD EE 0F
0755 C402 86 20
0756 C404 7D 00 04
0757 C407 27 05
0758 C409 7F 00 04
0759 C40C 86 20
0760 C40E A7 00
0761 C410 08
0762 C411 CD EF 00
0763 C414 CE C5 45
0764 C417 BD C8 66
0765 C41A BD C5 4D
0766 C41D 22 19
0767 C41F CE C5 41
0768 C422 BD C8 66
0769 C425 BD C5 4D
0770 C428 22 16
0771 C42A 18 6A 02
0772 C42D CE C1 8F
0773 C430 BD C8 66
0774 C433 BD C1 93
0775 C436 20 DC
0776 C438 18 6C 02
0777 C43B CE C1 88
0778 C43E 20 F0
0779 C440 CE C5 49
0780 C443 BD C8 66
0781 C446 BD C2 3C

TTL FLTASC
*****
*
* FLOATING POINT TO ASCII CONVERSION SUBROUTINE
*
* This subroutine performs floating point to ASCII conversion of
* the number in FPACC1. The ascii string is placed in a buffer
* pointed to by the X index register. The buffer must be at least
* 14 bytes long to contain the ASCII conversion. The resulting
* ASCII string is terminated by a zero (0) byte. Upon exit the
* X Index register will be pointing to the first character of the
* string. FPACC1 and FPACC2 will remain unchanged.
*****
*
*
FLTASC EQU *
0734 C3DB 3C PSHX SAVE THE POINTER TO THE STRING BUFFER.
0735 C3DC CE 00 00 LDX #FPACC1EX POINT TO FPACC1.
0736 C3DF BD C1 80 JSR CHCK0 IS FPACC1 0?
0737 C3E2 26 07 BNE FLTASC1 NO. GO CONVERT THE NUMBER.
0738 C3E4 38 PULX RESTORE POINTER.
0739 C3E5 CC 30 00 LDD #3000 GET ASCII CHARACTER + TERMINATING BYTE.
0740 C3E8 ED 00 STD 0,X PUT IT IN THE BUFFER.
0741 C3EA 39 RTS RETURN.
0742 C3EB DE 00 FLTASC1 LDX FPACC1EX SAVE FPACC1.
0743 C3ED 3C PSHX
0744 C3EE DE 02 LDX FPACC1MN+1
0745 C3F0 3C PSHX
0746 C3F1 96 04 LDAA MANTSGN1
0747 C3F3 36 PSHA
0748 C3F4 BD C8 39 JSR PSHFPAC2 SAVE FPACC2.
0749 C3F7 CE 00 00 LDX #0
0750 C3FA 3C PSHX ALLOCATE LOCALS.
0751 C3FB 3C PSHX
0752 C3FC 3C PSHX SAVE SPACE FOR STRING BUFFER POINTER.
0753 C3FD 18 30 TSY POINT TO LOCALS.
0754 C3FF CD EE 0F LDX 15,Y GET POINTER FROM STACK.
0755 C402 86 20 LDAA #520 PUT A SPACE IN THE BUFFER IF NUMBER NOT NEGATIVE.
0756 C404 7D 00 04 TST MANTSGN1 IS IT NEGATIVE?
0757 C407 27 05 BEQ FLTASC2 NO. GO PUT SPACE.
0758 C409 7F 00 04 CLR MANTSGN1 MAKE NUMBER POSITIVE FOR REST OF CONVERSION.
0759 C40C 86 20 LDAA #' YES. PUT MINUS SIGN IN BUFFER.
0760 C40E A7 00 FLTASC2 STAA 0,X
0761 C410 08 INX POINT TO NEXT LOCATION.
0762 C411 CD EF 00 STX 0,Y SAVE POINTER.
0763 C414 CE C5 45 FLTASC5 LDX #99999999 POINT TO CONSTANT 99999999.
0764 C417 BD C8 66 JSR GETFPAC2 GET INTO FPACC2.
0765 C41A BD C5 4D JSR FLTAMP COMPARE THE NUMBERS. IS FPACC1 > 99999999?
0766 C41D 22 19 BHI FLTASC3 YES. GO DIVIDE FPACC1 BY 10.
0767 C41F CE C5 41 LDX #99999999 POINT TO CONSTANT 9999999.9
0768 C422 BD C8 66 JSR GETFPAC2 MOVE IT INTO FPACC2.
0769 C425 BD C5 4D JSR FLTAMP COMPARE NUMBERS. IS FPACC1 > 9999999.9?
0770 C428 22 16 BHI FLTASC4 YES. GO CONTINUE THE CONVERSION.
0771 C42A 18 6A 02 DEC 2,Y DECREMENT THE MULT./DIV. COUNT.
0772 C42D CE C1 8F LDX #CONST10 NO. MULTIPLY BY 10. POINT TO CONSTANT.
0773 C430 BD C8 66 FLTASC6 JSR GETFPAC2 MOVE IT INTO FPACC2.
0774 C433 BD C1 93 JSR FLTAMP
0775 C436 20 DC BRA FLTASC5 GO DO COMPARE AGAIN.
0776 C438 18 6C 02 FLTASC3 INC 2,Y INCREMENT THE MULT./DIV. COUNT.
0777 C43B CE C1 88 LDX #CONSTP1 POINT TO CONSTANT ".1".
0778 C43E 20 F0 BRA FLTASC6 GO DIVIDE FPACC1 BY 10.
0779 C440 CE C5 49 FLTASC4 LDX #CONSTP5 POINT TO CONSTANT OF ".5".
0780 C443 BD C8 66 JSR GETFPAC2 MOVE IT INTO FPACC2.
0781 C446 BD C2 3C JSR FLTADD ADD .5 TO NUMBER IN FPACC1 TO ROUND IT.

```


0782 C449 D6 00	LDAB	FPACC1EX	GET FPACC1 EXPONENT.
0783 C448 C0 B1	SUBB	#8B1	TAKE OUT BIAS +1.
0784 C44D 50	NEGB		MAKE IT NEGATIVE.
0785 C44E CB 17	ADDB	#23	ADD IN THE NUMBER OF MANTISSA BITS -1.
0786 C450 20 0A	BRA	FLTASC17	GO CHECK TO SEE IF WE NEED TO SHIFT AT ALL.
0787 C452 74 00 01	FLTASC7	LSR	FPACC1MN
0788 C455 76 00 02	ROR	FPACC1MN+1	SHIFT MANTISSA TO THE RIGHT BY THE RESULT (MAKE
0789 C458 76 00 03	ROR	FPACC1MN+2	THE NUMBER AN INTEGER).
0790 C45B 5A	DECB		DONE SHIFTING?
0791 C45C 26 F4	FLTASC17	BNE	FLTASC7
0792 C45E 86 01	LDA	#1	NO. KEEP GOING.
0793 C460 18 A7 03	STAA	3,Y	GET INITIAL VALUE OF "DIGITS AFTER D.P." COUNT.
0794 C463 18 A6 02	LDA	2,Y	INITIALIZE IT.
0795 C466 8B 08	ADDA	#8	GET DECIMAL EXPONENT.
0796	*		ADD THE NUMBER OF DECIMAL +1 TO THE EXPONENT.
0797 C468 2B 0A	BMI	FLTASC8	WAS THE ORIGINAL NUMBER > 9999999?
0798 C46A 81 08	CMPA	#8	YES. MUST BE REPRESENTED IN SCIENTIFIC NOTATION.
0799 C46C 24 06	BHS	FLTASC8	WAS THE ORIGINAL NUMBER < 1?
0800 C46E 4A	DECA		YES. MUST BE REPRESENTED IN SCIENTIFIC NOTATION.
0801 C46F 18 A7 03	STAA	3,Y	NO. NUMBER CAN BE REPRESENTED IN 7 DIGITS.
0802	*		MAKE THE DECIMAL EXPONENT THE DIGIT COUNT BEFORE
0803 C472 86 02	LDA	#2	THE DECIMAL POINT.
0804 C474 80 02	FLTASC8	SUBA	#2
0805 C476 18 A7 02	STAA	2,Y	SETUP TO ZERO THE DECIMAL EXPONENT.
0806 C479 18 6D 03	TST	3,Y	SUBTRACT 2 FROM THE DECIMAL EXPONENT.
0807 C47C 2E 15	BGT	FLTASC9	SAVE THE DECIMAL EXPONENT.
0808 C47E 86 2E	LDA	#1.	DOES THE NUMBER HAVE AN INTEGER PART? (EXP. >0)
0809 C480 CD EE 00	LDX	0,Y	YES. GO PUT IT OUT.9
0810 C483 A7 00	STAA	0,X	NO. GET DECIMAL POINT.
0811 C485 08	INX		GET POINTER TO BUFFER.
0812 C486 18 6D 03	TST	3,Y	PUT THE DECIMAL POINT IN THE BUFFER.
0813 C489 27 05	BEQ	FLTASC18	POINT TO NEXT BUFFER LOCATION.
0814 C48B 86 30	LDA	#10	IS THE DIGIT COUNT TILL EXPONENT =0?
0815 C48D A7 00	STAA	0,X	NO. NUMBER IS <.1
0816 C48F 08	INX		YES. FORMAT NUMBER AS .0XXXXXX
0817 C490 CD EF 00	FLTASC18	STX	0,Y
0818 C493 CE C5 2C	FLTASC9	LDX	#DEC DIG
0819 C496 86 07	LDA	#7	INITIALIZE THE THE NUMBER OF DIGITS COUNT.
0820 C498 18 A7 05	STAA	5,Y	
0821 C49B 18 6F 04	FLTASC10	CLR	4,Y
0822 C49E DC 02	FLTASC11	LDD	FPACC1MN+1
0823 C4A0 A3 01	SUBD	1,X	CLEAR THE DECIMAL DIGIT ACCUMULATOR.
0824 C4A2 DD 02	STD	FPACC1MN+1	GET LOWER 16 BITS OF MANTISSA.
0825 C4A4 96 01	LDA	FPACC1MN	SUBTRACT LOWER 16 BITS OF CONSTANT.
0826 C4A6 A2 00	SBCA	0,X	SAVE RESULT.
0827 C4A8 97 01	STAA	FPACC1MN	GET UPPER 8 BITS.
0828 C4AA 25 05	BCS	FLTASC12	SUBTRACT UPPER 8 BITS.
0829 C4AC 18 6C 04	INC	4,Y	SAVE RESULT. UNDERFLOW?
0830 C4AF 20 ED	BRA	FLTASC11	YES. GO ADD DECIMAL NUMBER BACK IN.
0831 C4B1 DC 02	FLTASC12	LDD	FPACC1MN+1
0832 C4B3 E3 01	ADD	1,X	ADD 1 TO DECIMAL NUMBER.
0833 C4B5 DD 02	STD	FPACC1MN+1	TRY ANOTHER SUBTRACTION.
0834 C4B7 96 01	LDA	FPACC1MN	GET FPACC1 MANTISSA LOW 16 BITS.
0835 C4B9 A9 00	ADCA	0,X	ADD LOW 16 BITS BACK IN.
0836 C4BB 97 01	STAA	FPACC1MN	SAVE THE RESULT.
0837 C4BD 18 A6 04	LDA	4,Y	GET HIGH 8 BITS.
0838 C4C0 8B 30	ADDA	#830	ADD IN HIGH 8 BITS OF CONSTANT.
0839 C4C2 3C	PSHX		SAVE RESULT.
0840 C4C3 CD EE 00	LDX	0,Y	GET DIGIT.
0841 C4C6 A7 00	STAA	0,X	MAKE IT ASCII.
0842 C4C8 08	INX		SAVE POINTER TO CONSTANTS.
0843 C4C9 18 6A 03	DEC	3,Y	GET POINTER TO BUFFER.
0844 C4CC 26 05	BNE	FLTASC16	PUT DIGIT IN BUFFER.
0845 C4CE 86 2E	LDA	#1.	POINT TO NEXT BUFFER LOCATION.
0846 C4D0 A7 00	STAA	0,X	SHOULD WE PUT A DECIMAL POINT IN THE BUFFER YET?
0847 C4D2 08	INX		NO. CONTINUE THE CONVERSION.
			YES. GET DECIMAL POINT.
			PUT IT IN THE BUFFER.
			POINT TO THE NEXT BUFFER LOCATION.

0848 C4D3 CD EF 00	FLTASC16 STX	0,Y	SAVE UPDATED POINTER.
0849 C4D6 38	PULX		RESTORE POINTER TO CONSTANTS.
0850 C4D7 08	INX		POINT TO NEXT CONSTANT.
0851 C4D8 08	INX		
0852 C4D9 08	INX		
0853 C4DA 18 6A 05	DEC	5,Y	DONE YET?
0854 C4DD 26 BC	BNE	FLTASC10	NO. CONTINUE CONVERSION OF "MANTISSA".
0855 C4DF CD EE 00	LDX	0,Y	YES. POINT TO BUFFER STRING BUFFER.
0856 C4E2 09	FLTASC13 DEX		POINT TO LAST CHARACTER PUT IN THE BUFFER.
0857 C4E3 A6 00	LDAA	0,X	GET IT.
0858 C4E5 81 30	CMPA	#530	WAS IT AN ASCII 0?
0859 C4E7 27 F9	BEQ	FLTASC13	YES. REMOVE TRAILING ZEROS.
0860 C4E9 08	INX		POINT TO NEXT AVAILABLE LOCATION IN BUFFER.
0861 C4EA 18 E6 02	LDAB	2,Y	DO WE NEED TO PUT OUT AN EXPONENT?
0862 C4ED 27 2A	BEQ	FLTASC15	NO. WE'RE DONE.
0863 C4EF 86 45	LDAA	#'E	YES. PUT AN 'E' IN THE BUFFER.
0864 C4F1 A7 00	STAA	0,X	
0865 C4F3 08	INX		POINT TO NEXT BUFFER LOCATION.
0866 C4F4 86 28	LDAA	#'+	ASSUME EXPONENT IS POSITIVE.
0867 C4F6 A7 00	STAA	0,X	PUT PLUS SIGN IN THE BUFFER.
0868 C4F8 50	TSTB		IS IT REALLY MINUS?
0869 C4F9 2A 05	BPL	FLTASC14	NO. IS'S OK AS IS.
0870 C4FB 50	NEGB		YES. MAKE IT POSITIVE.
0871 C4FC 86 2D	LDAA	#'-	PUT THE MINUS SIGN IN THE BUFFER.
0872 C4FE A7 00	STAA	0,X	
0873 C500 08	FLTASC14 INX		POINT TO NEXT BUFFER LOCATION.
0874 C501 CD EF 00	STX	0,Y	SAVE POINTER TO STRING BUFFER.
0875 C504 4F	CLRA		SET UP FOR DIVIDE.
0876 C505 CE 00 0A	LDX	#10	DIVIDE DECIMAL EXPONENT BY 10.
0877 C508 02	IDIV		
0878 C509 37	PSHB		SAVE REMAINDER.
0879 C50A 8F	XGDX		PUT QUOTIENT IN D.
0880 C50B CB 30	ADDB	#530	MAKE IT ASCII.
0881 C50D CD EE 00	LDX	0,Y	GET POINTER.
0882 C510 E7 00	STAB	0,X	PUT NUMBER IN BUFFER.
0883 C512 08	INX		POINT TO NEXT LOCATION.
0884 C513 33	PULB		GET SECOND DIGIT.
0885 C514 CB 30	ADDB	#530	MAKE IT ASCII.
0886 C516 E7 00	STAB	0,X	PUT IT IN THE BUFFER.
0887 C518 08	INX		POINT TO NEXT LOCATION.
0888 C519 6F 00	FLTASC15 CLR	0,X	TERMINATE STRING WITH A ZERO BYTE.
0889 C51B 38	PULX		CLEAR LOCALS FROM STACK.
0890 C51C 38	PULX		
0891 C51D 38	PULX		
0892 C51E BD CB 43	JSR	PULFPAC2	RESTORE FPACC2.
0893 C521 32	PULA		
0894 C522 97 04	STAA	MANTSGN1	
0895 C524 38	PULX		RESTORE FPACC1.
0896 C525 DF 02	STX	FPACC1MN+1	
0897 C527 38	PULX		
0898 C528 DF 00	STX	FPACC1EX	
0899 C52A 38	PULX		POINT TO THE START OF THE ASCII STRING.
0900 C52B 39	RTS		RETURN.
0901	*		
0902	*		
0903 C52C	DECDIG EQU	*	
0904 C52C 0F 42 40	FCB	\$0F,\$42,\$40	DECIMAL 1,000,000
0905 C52F 01 86 A0	FCB	\$01,\$86,\$A0	DECIMAL 100,000
0906 C532 00 27 10	FCB	\$00,\$27,\$10	DECIMAL 10,000
0907 C535 00 03 E8	FCB	\$00,\$03,\$E8	DECIMAL 1,000
0908 C538 00 00 64	FCB	\$00,\$00,\$64	DECIMAL 100
0909 C53B 00 00 0A	FCB	\$00,\$00,\$0A	DECIMAL 10
0910 C53E 00 00 01	FCB	\$00,\$00,\$01	DECIMAL 1
0911	*		
0912	*		
0913 C541	P9999999 EQU	*	CONSTANT 999999.9

0914 C541 94 74 23 FE	FCB	\$94,\$74,\$23,\$FE	
0915	*		
0916 C545	N9999999 EQU	* CONSTANT 9999999.	
0917 C545 98 18 96 7F	FCB	\$98,\$18,\$96,\$7F	
0918	*		
0919 C549	CONSTP5 EQU	* CONSTANT .5	
0920 C549 80 00 00 00	FCB	\$80,\$00,\$00,\$00	
0921	*		
0922	*		
0923 C54D	FLTCMP EQU	*	
0924 C54D 7D 00 04	TST MANTSGN1	IS FPACC1 NEGATIVE?	
0925 C550 2A 12	BPL FLTCMP2	NO. CONTINUE WITH COMPARE.	
0926 C552 7D 00 09	TST MANTSGN2	IS FPACC2 NEGATIVE?	
0927 C555 2A 0D	BPL FLTCMP2	NO. CONTINUE WITH COMPARE.	
0928 C557 DC 05	LDD FPACC2EX	YES. BOTH ARE NEGATIVE SO COMPARE MUST BE DONE	
0929 C559 1A 93 00	CPD FPACC1EX	BACKWARDS. ARE THEY EQUAL SO FAR?	
0930 C55C 26 05	BNE FLTCMP1	NO. RETURN WITH CONDITION CODES SET.	
0931 C55E DC 07	LDD FPACC2MN+1	YES. COMPARE LOWER 16 BITS OF MANTISSAS.	
0932 C560 1A 93 02	CPD FPACC1MN+1		
0933 C563 39	FLTCMP1 RTS	RETURN WITH CONDITION CODES SET.	
0934 C564 96 04	FLTCMP2 LDAA	MANTSGN1 GET FPACC1 MANTISSA SIGN.	
0935 C566 91 09	CMPA MANTSGN2	BOTH POSITIVE?	
0936 C568 26 F9	BNE FLTCMP1	NO. RETURN WITH CONDITION CODES SET.	
0937 C56A DC 00	LDD FPACC1EX	GET FPACC1 EXPONENT & UPPER 8 BITS OF MANTISSA.	
0938 C56C 1A 93 05	CPD FPACC2EX	SAME AS FPACC2?	
0939 C56F 26 F2	BNE FLTCMP1	NO. RETURN WITH CONDITION CODES SET.	
0940 C571 DC 02	LDD FPACC1MN+1	GET FPACC1 LOWER 16 BITS OF MANTISSA.	
0941 C573 1A 93 07	CPD FPACC2MN+1	COMPARE WITH FPACC2 LOWER 16 BITS OF MANTISSA.	
0942 C576 39	RTS	RETURN WITH CONDITION CODES SET.	
0943	*		
0944	*		
0945	*		

```

0946                                TTL      INT2FLT
0947                                *
0948                                *
0949                                *          UNSIGNED INTEGER TO FLOATING POINT          *
0950                                *
0951                                *          This subroutine performs "unsigned" integer to floating point
0952                                *          conversion of a 16 bit word.  The 16 bit integer must be in the
0953                                *          lower 16 bits of FPACC1 mantissa.  The resulting floating point
0954                                *          number is returned in FPACC1.
0955                                *
0956                                *
0957                                *
0958                                *
0959 C577                                UINT2FLT EQU      *
0960 C577 CE 00 00                                LDX      #FPACC1EX    POINT TO FPACC1.
0961 C57A BD C1 80                                JSR      CHCKO      IS IT ALREADY 0?
0962 C57D 26 01                                BNE     UINTFLT1   NO. GO CONVERT.
0963 C57F 39                                RTS                                YES. JUST RETURN.
0964 C580 86 98                                UINTFLT1 LDA    #59B      GET BIAS + NUMBER OF BITS IN MANTISSA.
0965 C582 97 00                                STAA   FPACC1EX    INITIALIZE THE EXPONENT.
0966 C584 BD C1 61                                JSR    FPNORM     GO MAKE IT A NORMALIZED FLOATING POINT VALUE.
0967 C587 0C                                CLC                                NO ERRORS.
0968 C588 39                                RTS                                RETURN.
0969                                *
0970                                *
0971                                *
0972                                *
0973                                *          SIGNED INTEGER TO FLOATING POINT          *
0974                                *
0975                                *
0976                                *          This routine works just like the unsigned integer to floating
0977                                *          point routine except the the 16 bit integer in the FPACC1
0978                                *          mantissa is considered to be in two's complement format.  This
0979                                *          will return a floating point number in the range -32768 to +32767.
0980                                *
0981                                *
0982                                *
0983                                *
0984 C589                                SINT2FLT EQU      *
0985 C589 DC 02                                LDD     FPACC1MN+1  GET THE LOWER 16 BITS OF FPACC1 MANTISSA.
0986 C58B 36                                PSHA                                SAVE SIGN OF NUMBER.
0987 C58C 2A 07                                BPL     SINTFLT1   IF POSITIVE JUST GO CONVERT.
0988 C58E 43                                COMA                                MAKE POSITIVE.
0989 C58F 53                                COMB                                #1
0990 C590 C3 00 01                                ADDD   #1          TWO'S COMPLEMENT.
0991 C593 DD 02                                STD     FPACC1MN+1  PUT IT BACK IN FPACC1 MANTISSA.
0992 C595 8D E0                                SINTFLT1 BSR    UINT2FLT  GO CONVERT.
0993 C597 32                                PULA                                GET SIGN OF ORIGINAL INTEGER.
0994 C598 C6 FF                                LDAB   #$FF       GET "MINUS SIGN".
0995 C59A 4D                                TSTA                                WAS THE NUMBER NEGATIVE?
0996 C59B 2A 02                                BPL     SINTFLT2   NO. RETURN.
0997 C59D D7 04                                STAB   MANTSGN1   YES. SET FPACC1 SIGN BYTE.
0998 C59F 0C                                SINTFLT2 CLC      NO ERRORS.
0999 C5A0 39                                RTS      RETURN.
1000                                *
1001                                *
1002                                *

```

```

1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019 C5A1
1020 C5A1 CE 00 00
1021 C5A4 8D C1 80
1022 C5A7 27 41
1023 C5A9 06 00
1024 C5AB C1 81
1025 C5AD 25 34
1026 C5AF 7D 00 04
1027 C5B2 2B 16
1028 C5B4 C1 90
1029 C5B6 22 27
1030 C5B8 C0 98
1031 C5BA 74 00 01
1032 C5BD 76 00 02
1033 C5C0 76 00 03
1034 C5C3 5C
1035 C5C4 26 F4
1036 C5C6 7F 00 00
1037 C5C9 39
1038 C5CA C1 8F
1039 C5CC 22 11
1040 C5CE C0 98
1041 C5D0 8D E8
1042 C5D2 DC 02
1043 C5D4 43
1044 C5D5 53
1045 C5D6 C3 00 01
1046 C5D9 DD 02
1047 C5DB 7F 00 04
1048 C5DE 39
1049 C5DF 86 05
1050 C5E1 0D
1051 C5E2 39
1052 C5E3 CC 00 00
1053 C5E6 DD 00
1054 C5E8 DD 02
1055 C5EA 39
1056
1057
1058

```

```

TTL   FLT2INT
*****
*
*
*           FLOATING POINT TO INTEGER CONVERSION
*
*   This subroutine will perform "unsigned" floating point to integer
*   conversion.  The floating point number if positive, will be
*   converted to an unsigned 16 bit integer ( 0 <= X <= 65535 ).  If
*   the number is negative it will be converted to a twos complement
*   16 bit integer.  This type of conversion will allow 16 bit
*   addresses to be represented as positive numbers when in floating
*   point format.  Any fractional number part is disregarded
*
*****
*
*
FLT2INT EQU *
LDX #FPACC1EX POINT TO FPACC1.
JSR CHCKO IS IT 0?
BEQ FLT2INT3 YES. JUST RETURN.
LDAB FPACC1EX GET FPACC1 EXPONENT.
CMPB #81 IS THERE AN INTEGER PART?
BLO FLT2INT2 NO. GO PUT A 0 IN FPACC1.
TST MANTSGN1 IS THE NUMBER NEGATIVE?
BMI FLT2INT1 YES. GO CONVERT NEGATIVE NUMBER.
CMPB #90 IS THE NUMBER TOO LARGE TO BE MADE AN INTEGER?
BHI FLT2INT4 YES. RETURN WITH AN ERROR.
SUBB #98 SUBTRACT THE BIAS PLUS THE NUMBER OF BITS.
FLT2INT5 LSR FPACC1MN MAKE THE NUMBER AN INTEGER.
ROR FPACC1MN+1
ROR FPACC1MN+2
INCB DONE SHIFTING?
BNE FLT2INT5 NO. KEEP GOING.
CLR FPACC1EX ZERO THE EXPONENT (ALSO CLEARS THE CARRY).
RTS
FLT2INT1 CMPB #8F IS THE NUMBER TOO SMALL TO BE MADE AN INTEGER?
BHI FLT2INT4 YES. RETURN ERROR.
SUBB #98 SUBTRACT BIAS PLUS NUMBER OF BITS.
BSR FLT2INT5 GO DO SHIFT.
LDD FPACC1MN+1 GET RESULTING INTEGER.
COMA MAKE IT NEGATIVE.
COMB
ADDD #1 TWO'S COMPLEMENT.
STD FPACC1MN+1 SAVE RESULT.
CLR MANTSGN1 CLEAR MANTISSA SIGN. (ALSO CLEARS THE CARRY)
RTS RETURN.
FLT2INT4 LDAA #TOLGSMER NUMBER TOO LARGE OR TOO SMALL TO CONVERT TO INT.
SEC FLAG ERROR.
RTS RETURN.
FLT2INT2 LDD #0
STD FPACC1EX ZERO FPACC1.
STD FPACC1MN+1 (ALSO CLEARS THE CARRY)
FLT2INT3 RTS RETURN.
*
*
*

```

```

1059          TTL      FLTSOR
1060          *****
1061          *
1062          *          SQUARE ROOT SUBROUTINE          *
1063          *
1064          *          This routine is used to calculate the square root of the floating *
1065          *          point number in FPACC1. If the number in FPACC1 is negative an *
1066          *          error is returned.
1067          *
1068          *          WORSE CASE = 16354 CYCLES = 8177 uS @ 2MHz
1069          *
1070          *****
1071          *
1072          *
1073 C5EB      FLTSOR EQU *
1074 C5EB CE 00 00          LDX #FPACC1EX POINT TO FPACC1.
1075 C5EE BD C1 80          JSR CHCKO IS IT ZERO?
1076 C5F1 26 01          BNE FLTSOR1 NO. CHECK FOR NEGATIVE.
1077 C5F3 39          RTS YES. RETURN.
1078 C5F4 7D 00 04          FLTSOR1 TST MANTSGN1 IS THE NUMBER NEGATIVE?
1079 C5F7 2A 04          BPL FLTSOR2 NO. GO TAKE ITS SQUARE ROOT.
1080 C5F9 86 06          LDAA #NSORTERR YES. ERROR.
1081 C5FB 0D          SEC FLAG ERROR.
1082 C5FC 39          RTS RETURN.
1083 C5FD BD C8 39          FLTSOR2 JSR PSHFPAC2 SAVE FPACC2.
1084 C600 86 04          LDAA #4 GET ITERATION LOOP COUNT.
1085 C602 36          PSHA SAVE IT ON THE STACK.
1086 C603 DE 02          LDX FPACC1MN+1 SAVE INITIAL NUMBER.
1087 C605 3C          PSHX
1088 C606 DE 00          LDX FPACC1EX
1089 C608 3C          PSHX
1090 C609 18 30          TSY POINT TO IT.
1091 C60B 8D 39          BSR TFR1TO2 TRANSFER FPACC1 TO FPACC2.
1092 C60D 96 05          LDAA FPACC2EX GET FPACC1 EXPONENT.
1093 C60F 80 80          SUBA #S80 REMOVE BIAS FROM EXPONENT.
1094 C611 4C          INCA COMPENSATE FOR ODD EXPONENTS (GIVES CLOSER GUESS)
1095 C612 2A 03          BPL FLTSOR3 IF NUMBER >1 DIVIDE EXPONENT BY 2 & ADD BIAS.
1096 C614 44          LSRA IF <1 JUST DIVIDE IT BY 2.
1097 C615 20 03          BRA FLTSOR4 GO CALCULATE THE SQUARE ROOT.
1098 C617 44          FLTSOR3 LSRA DIVIDE EXPONENT BY 2.
1099 C618 88 80          ADDA #S80 ADD BIAS BACK IN.
1100 C61A 97 05          FLTSOR4 STAA FPACC2EX SAVE EXPONENT/2.
1101 C61C BD C3 0A          FLTSOR5 JSR FLTDIV DIVIDE THE ORIGINAL NUMBER BY THE GUESS.
1102 C61F BD C2 3C          JSR FLTADD ADD THE "GUESS" TO THE QUOTIENT.
1103 C622 7A 00 00          DEC FPACC1EX DIVIDE THE RESULT BY 2 TO PRODUCE A NEW GUESS.
1104 C625 8D 1F          BSR TFR1TO2 PUT THE NEW GUESS INTO FPACC2.
1105 C627 18 EC 00          LDD 0,Y GET THE ORIGINAL NUMBER.
1106 C62A DD 00          STD FPACC1EX PUT IT BACK IN FPACC1.
1107 C62C 18 EC 02          LDD 2,Y GET MANTISSA LOWER 16 BITS.
1108 C62F DD 02          STD FPACC1MN+1
1109 C631 18 6A 04          DEC 4,Y BEEN THROUGH THE LOOP 4 TIMES?
1110 C634 26 E6          BNE FLTSOR5 NO. KEEP GOING.
1111 C636 DC 05          LDD FPACC2EX THE FINAL GUESS IS THE ANSWER.
1112 C638 DD 00          STD FPACC1EX PUT IT IN FPACC1.
1113 C63A DC 07          LDD FPACC2MN+1
1114 C63C DD 02          STD FPACC1MN+1
1115 C63E 38          PULX GET RID OF ORIGINAL NUMBER.
1116 C63F 38          PULX
1117 C640 31          INS GET RID OF LOOP COUNT VARIABLE.
1118 C641 BD C8 43          JSR PULFPAC2 RESTORE FPACC2.
1119 C644 0C          CLC NO ERRORS.
1120 C645 39          RTS
1121          *
1122          *
1123 C646          TFR1TO2 EQU *

```

1124 C646 DC 00	LDD	FPACC1EX	GET FPACC1 EXPONENT & HIGH 8 BIT OF MANTISSA.
1125 C648 DD 05	STD	FPACC2EX	PUT IT IN FPACC2.
1126 C64A DC 02	LDD	FPACC1MN+1	GET FPACC1 LOW 16 BITS OF MANTISSA.
1127 C64C DD 07	STD	FPACC2MN+1	PUT IT IN FPACC2.
1128 C64E 96 04	LDA	MANTSGN1	TRANSFER THE SIGN.
1129 C650 97 09	STAA	MANTSGN2	
1130 C652 39	RTS		RETURN.
1131	*		
1132	*		
1133	*		

```

1134                                TTL   FLTSIN
1135                                *
1136                                *
1137                                *           FLOATING POINT SINE
1138                                *
1139                                *
1140                                *
1141                                *
1142 C653          FLTSIN EQU      *
1143 C653 BD C8 39      JSR      PSHFPAC2   SAVE FPACC2 ON THE STACK.
1144 C656 BD C7 59      JSR      ANGRED     GO REDUCE THE ANGLE TO BETWEEN +/-PI.
1145 C659 37           PSHB             SAVE THE QUAD COUNT.
1146 C65A 36           PSHA             SAVE THE SINE/COSINE FLAG.
1147 C65B BD C8 13      JSR      DEG2RAD   CONVERT DEGREES TO RADIANs.
1148 C65E 32           PULA             RESTORE THE SINE/COSINE FLAG.
1149 C65F BD C6 8F      FLTSIN1 JSR      SIN COS   GO GET THE SINE OF THE ANGLE.
1150 C662 32           PULA             RESTORE THE QUAD COUNT.
1151 C663 81 02        CMPA      #2           WAS THE ANGLE IN QUADS 1 OR 2?
1152 C665 23 03        BLS      FLTSIN2   YES. SIGN OF THE ANSWER IS OK.
1153 C667 73 00 04      COM      MANTSGN1  NO. SINE IN QUADS 3 & 4 IS NEGATIVE.
1154 C66A 0C           FLTSIN2 CLC             SHOW NO ERRORS.
1155 C66B BD C8 43      JSR      PULFPAC2  RESTORE FPACC2
1156 C66E 39           RTS             RETURN.
1157                                *
1158                                *
1159                                *

```



```

1160
1161
1162
1163
1164
1165
1166
1167
1168 C66F
1169 C66F BD C8 39
1170 C672 BD C7 59
1171 C675 37
1172 C676 36
1173 C677 BD C8 13
1174 C67A 32
1175 C678 88 01
1176 C67D BD C6 8F
1177 C680 32
1178 C681 81 01
1179 C683 27 07
1180 C685 81 04
1181 C687 27 03
1182 C689 73 00 04
1183 C68C 7E C6 6A
1184
1185
1186

```

```

TTL   FLT COS
*****
*
*
*           FLOATING POINT COSINE
*
*****
*
*
FLT COS EQU *
JSR   PSHFPAC2   SAVE FPACC2 ON THE STACK.
JSR   ANGREDE   GO REDUCE THE ANGLE TO BETWEEN +/-PI.
PSHB  PSNB      SAVE THE QUAD COUNT.
PSNA  PSNA      SAVE THE SINE/COSINE FLAG.
JSR   DEG2RAD   CONVERT TO RADIAN.
PULA  PULA      RESTORE THE SINE/COSINE FLAG.
EDRA  #S01     COMPLIMENT 90'S COPMLIMENT FLAG FOR COSINE.
JSR   SINCOS    GO GET THE COSINE OF THE ANGLE.
PULA  PULA      RESTORE THE QUAD COUNT.
CMPA  #1       WAS THE ORIGINAL ANGLE IN QUAD 1?
BEQ   FLT COS 1 YES. SIGN IS OK.
CMPA  #4       WAS IT IN QUAD 4?
BEQ   FLT COS 1 YES. SIGN IS OK.
COM   MANTSGN1 NO. COSINE IS NEGATIVE IN QUADS 2 & 3.
FLT COS JMP FLT SIN2 FLAG NO ERRORS, RESTORE FPACC2, & RETURN.
*
*
*

```

```

1187          TTL      SINCOS
1188          *****
1189          *
1190          *          FLOATING POINT SINE AND COSINE SUBROUTINE          *
1191          *
1192          *****
1193          *
1194          *
1195 C68F      SINCOS EQU *
1196 C68F 36      PSHA          SAVE SINE/COSINE FLAG ON STACK.
1197 C690 DE 02      LDX          FPACC1MN+1  SAVE THE VALUE OF THE ANGLE.
1198 C692 3C      PSHX
1199 C693 DE 00      LDX          FPACC1EX
1200 C695 3C      PSHX
1201 C696 96 04      LDAA        MANTSGN1
1202 C698 36      PSHA
1203 C699 CE C7 C3  LDX          #SINFACT  POINT TO THE FACTORIAL TABLE.
1204 C69C 3C      PSHX          SAVE POINTER TO THE SINE FACTORIAL TABLE.
1205 C69D 3C      PSHX          JUST ALLOCATE ANOTHER LOCAL (VALUE NOT IMPORTANT)
1206 C69E 86 04      LDAA        #%4          GET INITIAL LOOP COUNT.
1207 C6A0 36      PSHA          SAVE AS LOCAL ON STACK
1208 C6A1 18 30      TSY          POINT TO LOCALS.
1209 C6A3 BD C6 46      JSR          TFR1TO2  TRANSFER FPACC1 TO FPACC2.
1210 C6A6 BD C1 93      JSR          FLT MUL      GET X^2 IN FPACC1.
1211 C6A9 18 60 0A      TST          10,Y      ARE WE DOING THE SINE?
1212 C6AC 27 08      BEQ          SINCOS7  YES. GO DO IT.
1213 C6AE CE C7 D3      LDX          #COSFACT  NO. GET POINTER TO COSINE FACTORIAL TABLE.
1214 C6B1 CD EF 01      STX          1,Y      SAVE IT.
1215 C6B4 BD C6 46      JSR          TFR1TO2  COPY X^2 INTO FPACC2.
1216 C6B7 20 06      BRA          SINCOS4  GENERATE EVEN POWERS OF "X" FOR COSINE.
1217 C6B9 BD C7 AA      SINCOS7 JSR          EXG1AND2  PUT X^2 IN FPACC2 & X IN FPACC1.
1218 C6BC BD C1 93      SINCOS1 JSR          FLT MUL      CREATE X^3,5,7,9 OR X^2,4,6,8.
1219 C6BF DE 02      SINCOS4 LDX          FPACC1MN+1  SAVE EACH ONE ON THE STACK.
1220 C6C1 3C      PSHX
1221 C6C2 DE 00      LDX          FPACC1EX
1222 C6C4 3C      PSHX
1223 C6C5 96 04      LDAA        MANTSGN1
1224 C6C7 36      PSHA          SAVE THE MANTISSA SIGN.
1225 C6C8 18 6A 00      DEC          0,Y      HAVE WE GENERATED ALL THE POWERS YET?
1226 C6CB 26 EF      BNE         SINCOS1  NO. GO DO SOME MORE.
1227 C6CD 86 04      LDAA        #%4          SET UP LOOP COUNT.
1228 C6CF 18 A7 00      STAA       0,Y
1229 C6D2 30      TSX
1230 C6D3 CD EF 03      SINCOS2 STX          3,Y      POINT TO POWERS ON THE STACK.
1231 C6D6 CD EE 01      LDX          1,Y      SAVE THE POINTER.
1232 C6D9 BD C8 66      JSR          GETFPACC2  GET THE POINTER TO THE FACTORIAL CONSTANTS.
1233 C6DC 08      INX          PUT THE NUMBER IN FPACC2.
1234 C6DD 08      INX          POINT TO THE NEXT CONSTANT.
1235 C6DE 08      INX
1236 C6DF 08      INX
1237 C6E0 CD EF 01      STX          1,Y      SAVE THE POINTER.
1238 C6E3 CD EE 03      LDX          3,Y      GET POINTER TO POWERS.
1239 C6E6 A6 00      LDAA       0,X      GET NUMBER SIGN.
1240 C6E8 97 04      STAA       MANTSGN1  PUT IN FPACC1 MANTISSA SIGN.
1241 C6EA EC 01      LDD          1,X      GET LOWER 16-BITS OF THE MANTISSA.
1242 C6EC DD 00      STD         FPACC1EX  PUT IN FPACC1 MANTISSA.
1243 C6EE EC 03      LDD          3,X      GET HIGH 8 BITS OF THE MANTISSA & EXPONENT.
1244 C6F0 DD 02      STD         FPACC1MN+1  PUT IT IN FPACC1 EXPONENT & MANTISSA.
1245 C6F2 BD C1 93      JSR          FLT MUL      MULTIPLY THE TWO.
1246 C6F5 CD EE 03      LDX          3,Y      GET POINTER TO POWERS BACK.
1247 C6F8 DC 02      LDD         FPACC1MN+1  SAVE RESULT WHERE THE POWER OF X WAS.
1248 C6FA ED 03      STD          3,X
1249 C6FC DC 00      LDD         FPACC1EX
1250 C6FE ED 01      STD          1,X
1251 C700 96 04      LDAA        MANTSGN1  SAVE SIGN.

```

1252 C702 A7 00	STAA	0,X	
1253 C704 08	INX		POINT TO THE NEXT POWER.
1254 C705 08	INX		
1255 C706 08	INX		
1256 C707 08	INX		
1257 C708 08	INX		
1258 C709 18 6A 00	DEC	0,Y	DONE?
1259 C70C 26 C5	BNE	SINCOS2	NO. GO DO ANOTHER MULTIPLICATION.
1260 C70E 86 03	LDA	#83	GET LOOP COUNT.
1261 C710 18 A7 00	STAA	0,Y	SAVE IT.
1262 C713 CD EE 03	SINCOS3 LDX	3,Y	POINT TO RESULTS ON THE STACK.
1263 C716 09	DEX		POINT TO PREVIOUS RESULT.
1264 C717 09	DEX		
1265 C718 09	DEX		
1266 C719 09	DEX		
1267 C71A 09	DEX		
1268 C71B CD EF 03	STX	3,Y	SAVE THE NEW POINTER.
1269 C71E A6 00	LDA	0,X	GET NUMBERS SIGN.
1270 C720 97 09	STAA	MANTSGN2	PUT IT IN FPACC2.
1271 C722 EC 01	LDD	1,X	GET LOW 16 BITS OF THE MANTISSA.
1272 C724 DD 05	STD	FPACC2EX	PUT IN FPACC2.
1273 C726 EC 03	LDD	3,X	GET HIGH 8 BIT & EXPONENT.
1274 C728 DD 07	STD	FPACC2MN+1	PUT IN FPACC2.
1275 C72A BD C2 3C	JSR	FLTADD	GO ADD THE TWO NUMBERS.
1276 C72D 18 6A 00	DEC	0,Y	DONE?
1277 C730 26 E1	BNE	SINCOS3	NO. GO ADD THE NEXT TERM IN.
1278 C732 18 60 0A	TST	10,Y	ARE WE DOING THE SINE?
1279 C735 27 08	BEQ	SINCOS5	YES. GO PUT THE ORIGINAL ANGLE INTO FPACC2.
1280 C737 CE C7 E3	LDX	#ONE	NO. FOR COSINE PUT THE CONSTANT 1 INTO FPACC2.
1281 C73A BD C8 66	JSR	GETFPAC2	
1282 C73D 20 0F	BRA	SINCOS6	GO ADD IT TO THE SUM OF THE TERMS.
1283 C73F 18 A6 05	SINCOS5 LDA	5,Y	GET THE VALUE OF THE ORIGINAL ANGLE.
1284 C742 97 09	STAA	MANTSGN2	PUT IT IN FPACC2.
1285 C744 18 EC 06	LDD	6,Y	
1286 C747 DD 05	STD	FPACC2EX	
1287 C749 18 EC 08	LDD	8,Y	
1288 C74C DD 07	STD	FPACC2MN+1	
1289 C74E BD C2 3C	SINCOS6 JSR	FLTADD	GO ADD IT TO THE SUM OF THE TERMS.
1290 C751 30	TSX		NOW CLEAN UP THE STACK.
1291 C752 8F	XGDX		PUT STACK IN D.
1292 C753 C3 00 1F	ADD	#31	CLEAR ALL THE TERMS & TEMPS OFF THE STACK.
1293 C756 8F	XGDX		
1294 C757 35	TXS		UPDATE THE STACK POINTER.
1295 C758 39	RTS		RETURN.
1296	*		
1297	*		
1298 C759	ANGRED EQU	*	
1299 C759 4F	CLRA		INITIALIZE THE 45'S COMPLIMENT FLAG.
1300 C75A 36	PSHA		PUT IT ON THE STACK.
1301 C75B 4C	INCA		INITIALIZE THE QUAD COUNT TO 1.
1302 C75C 36	PSHA		PUT IT ON THE STACK.
1303 C75D 18 30	TSY		POINT TO IT.
1304 C75F CE C7 EB	LDX	#THREE60	POINT TO THE CONSTANT 360.
1305 C762 BD C8 66	JSR	GETFPAC2	GET IT INTO FPACC.
1306 C765 7D 00 04	TST	MANTSGN1	IS THE INPUT ANGLE NEGATIVE?
1307 C768 2A 03	BPL	ANGRED1	NO. SKIP THE ADD.
1308 C76A BD C2 3C	JSR	FLTADD	YES. MAKE THE ANGLE POSITIVE BY ADDING 360 DEG.
1309 C76D 7A 00 05	ANGRED1 DEC	FPACC2EX	MAKE THE CONSTANT IN FPACC2 90 DEGREES.
1310 C770 7A 00 05	DEC	FPACC2EX	
1311 C773 BD C5 4D	ANGRED2 JSR	FLTCMP	IS THE ANGLE LESS THAN 90 DEGREES ALREADY?
1312 C776 23 08	BLS	ANGRED3	YES. RETURN WITH QUAD COUNT.
1313 C778 BD C2 FE	JSR	FLTSUB	NO. REDUCE ANGLE BY 90 DEGREES.
1314 C77B 18 6C 00	INC	0,Y	INCREMENT THE QUAD COUNT.
1315 C77E 20 F3	BRA	ANGRED2	GO SEE IF IT'S LESS THAN 90 NOW.
1316 C780 18 A6 00	ANGRED3 LDA	0,Y	GET THE QUAD COUNT.
1317 C783 81 01	CMPA	#1	WAS THE ORIGINAL ANGLE IN QUAD 1?

1318 C785 27 08	BEG	ANGRED4	YES. COMPUTE TRIG FUNCTION AS IS.
1319 C787 81 03	CHPA	#3	NO. WAS THE ORIGINAL ANGLE IN QUAD 3?
1320 C789 27 07	BEG	ANGRED4	YES. COMPUTE THE TRIG FUNCTION AS IF IN QUAD 1.
1321 C788 86 FF	LDA	#\$FF	NO. MUST COMPUTE THE TRIG FUNCTION OF THE 90'S
1322 C78D 97 04	STAA	MANTSGN1	COMPLIMENT ANGLE.
1323 C78F 8D C2 3C	JSR	FLTADD	ADD 90 DEGREES TO THE NEGATED ANGLE.
1324 C792 7A 00 05	ANGRED4 DEC	FPACC2EX	MAKE THE ANGLE IN FPACC2 45 DEGREES.
1325 C795 8D C5 4D	JSR	FLTCMP	IS THE ANGLE < 45 DEGREES?
1326 C798 23 0D	BLS	ANGRED5	YES. IT'S OK AS IT IS.
1327 C79A 7C 00 05	INC	FPACC2EX	NO. MUST GET THE 90'S COMPLIMENT.
1328 C79D 86 FF	LDA	#\$FF	MAKE FPACC1 NEGATIVE.
1329 C79F 97 04	STAA	MANTSGN1	
1330 C7A1 8D C2 3C	JSR	FLTADD	GET THE 90'S COMPLIMENT.
1331 C7A4 18 6C 01	INC	1,Y	SET THE FLAG.
1332 C7A7 33	ANGRED5 PULB		GET THE QUAD COUNT.
1333 C7AB 32	PULA		GET THE COMPLIMENT FLAG.
1334 C7A9 39	RTS		RETURN WITH THE QUAD COUNT & COMPLIMENT FLAG.
1335	*		
1336	*		
1337 C7AA	EXGIAND2 EQU	*	
1338 C7AA DC 00	LDD	FPACC1EX	
1339 C7AC DE 05	LDX	FPACC2EX	
1340 C7AE DD 05	STD	FPACC2EX	
1341 C7B0 DF 00	STX	FPACC1EX	
1342 C7B2 DC 02	LDD	FPACC1MN+1	
1343 C7B4 DE 07	LDX	FPACC2MN+1	
1344 C7B6 DD 07	STD	FPACC2MN+1	
1345 C7B8 DF 02	STX	FPACC1MN+1	
1346 C7BA 96 04	LDA	MANTSGN1	
1347 C7BC D6 09	LDAB	MANTSGN2	
1348 C7BE 97 09	STAA	MANTSGN2	
1349 C7C0 D7 04	STAB	MANTSGN1	
1350 C7C2 39	RTS		RETURN.
1351	*		
1352	*		
1353 C7C3	SINFACT EQU	*	
1354 C7C3 6E 38 EF 1D	FCB	\$6E,\$38,\$EF,\$1D	+(1/91)
1355 C7C7 74 D0 0D 01	FCB	\$74,\$D0,\$0D,\$01	-(1/71)
1356 C7CB 7A D8 88 89	FCB	\$7A,\$D8,\$88,\$89	+(1/51)
1357 C7CF 7E AA AA AB	FCB	\$7E,\$AA,\$AA,\$AB	-(1/31)
1358	*		
1359	*		
1360 C7D3	COSFACT EQU	*	
1361 C7D3 71 50 0D 01	FCB	\$71,\$50,\$0D,\$01	+(1/81)
1362 C7D7 77 B6 0B 61	FCB	\$77,\$B6,\$0B,\$61	-(1/61)
1363 C7DB 7C 2A AA AB	FCB	\$7C,\$2A,\$AA,\$AB	+(1/41)
1364 C7DF 80 80 00 00	FCB	\$80,\$80,\$00,\$00	-(1/21)
1365	*		
1366	*		
1367 C7E3 81 00 00 00	ONE	FCB	\$81,\$00,\$00,\$00 1.0
1368 C7E7 82 49 0F DB	P1	FCB	\$82,\$49,\$0F,\$DB 3.1415927
1369 C7EB 89 34 00 00	THREE60	FCB	\$89,\$34,\$00,\$00 360.0
1370	*		
1371	*		
1372	*		

1373
 1374
 1375
 1376
 1377
 1378
 1379
 1380
 1381 C7EF
 1382 C7EF BD C8 39
 1383 C7F2 BD C6 46
 1384 C7F5 BD C6 6F
 1385 C7F8 BD C7 AA
 1386 C7FB BD C6 53
 1387 C7FE BD C3 0A
 1388 C801 24 08
 1389 C803 CE C8 0F
 1390 C806 BD C8 50
 1391 C809 86 07
 1392 C80B BD C8 43
 1393 C80E 39
 1394
 1395
 1396 C80F
 1397 C80F FE 7F FF FF
 1398
 1399
 1400

```

TTL      FLTTAN
*****
*
*           FLOATING POINT TANGENT
*
*****
*
*
FLTTAN   EQU      *
          JSR      PSHFPAC2     SAVE FPACC2 ON THE STACK.
          JSR      TFR1TO2     PUT A COPY OF THE ANGLE IN FPACC2.
          JSR      FLTCOS      GET COSINE OF THE ANGLE.
          JSR      EXGIAND2    PUT RESULT IN FPACC2 & PUT ANGLE IN FPACC1.
          JSR      FLTSIN      GET SIN OF THE ANGLE.
          JSR      FLTDIV      GET TANGENT OF ANGLE BY DOING SIN/COS.
          BCC      FLTTAN1     IF CARRY CLEAR, ANSWER OK.
          LDX      #MAXNUM     TANGENT OF 90 WAS ATTEMPTED. PUT LARGEST
          JSR      GETFPAC1    NUMBER IN FPACC1.
          LDAA     #TAN90ERR   GET ERROR CODE IN A.
FLTTAN1  JSR      PULFPAC2     RESTORE FPACC2.
          RTS                RETURN.
*
*
MAXNUM   EQU      *
          FCB      $FE,$7F,$FF,$FF     LARGEST POSITIVE NUMBER WE CAN HAVE.
*
*
*
  
```

```

1401                                TTL   TRIGUTIL
1402                                *****
1403                                *
1404                                *
1405                                *
1406                                *
1407                                *
1408                                *
1409                                *
1410                                *
1411                                *
1412                                *****
1413                                *
1414                                *
1415 C813    DEG2RAD EQU *
1416 C813 8D C8 39    JSR   PSHFPAC2    SAVE FPACC2.
1417 C816 CE C8 31    LDX   #PIOV180    POINT TO CONVERSION CONSTANT PI/180.
1418 C819 8D C8 66    DEG2RAD1 JSR   GETFPAC2    PUT IT INTO FPACC2.
1419 C81C 8D C1 93    JSR   FLT MUL    CONVERT DEGREES TO RADIANs.
1420 C81F 8D C8 43    JSR   PULFPAC2    RESTORE FPACC2.
1421 C822 39          RTS           RETURN. (NOTE! DON'T REPLACE THE "JSR/RTS" WITH
1422                                *
1423                                *
1424                                *
1425 C823    RAD2DEG EQU *
1426 C823 8D C8 39    JSR   PSHFPAC2    SAVE FPACC2.
1427 C826 CE C8 35    LDX   #C180OVPI    POINT TO CONVERSION CONSTANT 180/PI.
1428 C829 20 EE      BRA   DEG2RAD1    GO DO CONVERSION & RETURN.
1429                                *
1430                                *
1431 C82B    GETPI   EQU *
1432 C82B CE C7 E7    LDX   #PI           POINT TO CONSTANT "PI".
1433 C82E 7E C8 50    JMP   GETFPAC1    PUT IT IN FPACC1 AND RETURN.
1434                                *
1435                                *
1436 C831    PIOV180 EQU *
1437 C831 7B 0E FA 35 FCB   $7B,$0E,$FA,$35
1438                                *
1439 C835    C180OVPI EQU *
1440 C835 86 65 2E E1 FCB   $86,$65,$2E,$E1
1441                                *
1442                                *
1443                                *

```

```

1444                                TTL   PSHPULFPAC2
1445                                *****
1446                                *
1447                                *   The following two subroutines, PSHFPAC2 & PULFPAC2, push FPACC2
1448                                *   onto and pull FPACC2 off of the hardware stack respectively.
1449                                *   The number is stored in the "memory format".
1450                                *
1451                                *****
1452                                *
1453                                *
1454 C839                            PSHFPAC2 EQU *
1455 C839 38                          PULX          GET THE RETURN ADDRESS OFF OF THE STACK.
1456 C83A 3C                          PSHX          ALLOCATE FOUR BYTES OF STACK SPACE.
1457 C83B 3C                          PSHX
1458 C83C 8F                          XGDX          PUT THE RETURN ADDRESS IN D.
1459 C83D 30                          TSX          POINT TO THE STORAGE AREA.
1460 C83E 37                          PSHB          PUT THE RETURN ADDRESS BACK ON THE STACK.
1461 C83F 36                          PSHA
1462 C840 7E C8 8C                    JMP   PUTFPAC2  GO PUT FPACC2 ON THE STACK & RETURN.
1463                                *
1464                                *
1465 C843                            PULFPAC2 EQU *
1466 C843 30                          TSX          POINT TO THE RETURN ADDRESS.
1467 C844 08                          INX          POINT TO THE SAVED NUMBER.
1468 C845 08                          INX
1469 C846 BD C8 66                    JSR   GETFPAC2  RESTORE FPACC2.
1470 C849 38                          PULX          GET THE RETURN ADDRESS OFF THE STACK.
1471 C84A 31                          INS          REMOVE THE NUMBER FROM THE STACK.
1472 C84B 31                          INS
1473 C84C 31                          INS
1474 C84D 31                          INS
1475 C84E 6E 00                      JMP   0,X      RETURN.
1476                                *
1477                                *
1478                                *

```

```

1479                               TTL   GETFPAC
1480                               *****
1481                               *
1482                               *                               GETFPACX SUBROUTINE
1483                               *
1484                               *   The GETFPAC1 and GETFPAC2 subroutines get a floating point number
1485                               *   stored in memory and put it into either FPACC1 or FPACC2 in a format
1486                               *   that is expected by all the floating point math routines. These
1487                               *   routines may easily be replaced to convert any binary floating point
1488                               *   format (i.e. IEEE format) to the format required by the math
1489                               *   routines. The "memory" format converted by these routines is shown
1490                               *   below:
1491                               *
1492                               *   31 _____ 24 23 22 _____ 0
1493                               *   exponent   s      mantissa
1494                               *
1495                               *   The exponent is biased by 128 to facilitate floating point
1496                               *   comparisons. The sign bit is 0 for positive numbers and 1
1497                               *   for negative numbers. The mantissa is stored in hidden bit
1498                               *   normalized format so that 24 bits of precision can be obtained.
1499                               *   Since a normalized floating point number always has its most
1500                               *   significant bit set, we can use the 24th bit to hold the mantissa
1501                               *   sign. This allows us to get 24 bits of precision in the mantissa
1502                               *   and store the entire number in just 4 bytes. The format required by
1503                               *   the math routines uses a separate byte for the sign, therefore each
1504                               *   floating point accumulator requires five bytes.
1505                               *
1506                               *   *****
1507                               *
1508                               *
1509                               *   GETFPAC1 EQU *
1510                               *   LDD 0,X           GET THE EXPONENT & HIGH BYTE OF THE MANTISSA,
1511                               *   BEQ GETFP12        IF NUMBER IS ZERO, SKIP SETTING THE MS BIT.
1512                               *   CLR MANTSGN1       SET UP FOR POSITIVE NUMBER.
1513                               *   TSTB              IS NUMBER NEGATIVE?
1514                               *   BPL GETFP11        NO. LEAVE SIGN ALONE.
1515                               *   COM MANTSGN1       YES. SET SIGN TO NEGATIVE.
1516                               *   GETFP11 ORAB #80    RESTORE MOST SIGNIFICANT BIT IN MANTISSA.
1517                               *   GETFP12 STD FPACC1EX  PUT IN FPACC1.
1518                               *   LDD 2,X           GET LOW 16-BITS OF THE MANTISSA.
1519                               *   STD FPACC1MN+1     PUT IN FPACC1.
1520                               *   RTS                RETURN.
1521                               *
1522                               *
1523                               *   GETFPAC2 EQU *
1524                               *   LDD 0,X           GET THE EXPONENT & HIGH BYTE OF THE MANTISSA,
1525                               *   BEQ GETFP22        IF NUMBER IS 0, SKIP SETTING THE MS BIT.
1526                               *   CLR MANTSGN2       SET UP FOR POSITIVE NUMBER.
1527                               *   TSTB              IS NUMBER NEGATIVE?
1528                               *   BPL GETFP21        NO. LEAVE SIGN ALONE.
1529                               *   COM MANTSGN2       YES. SET SIGN TO NEGATIVE.
1530                               *   GETFP21 ORAB #80    RESTORE MOST SIGNIFICANT BIT IN MANTISSA.
1531                               *   GETFP22 STD FPACC2EX  PUT IN FPACC1.
1532                               *   LDD 2,X           GET LOW 16-BITS OF THE MANTISSA.
1533                               *   STD FPACC2MN+1     PUT IN FPACC1.
1534                               *   RTS                RETURN.
1535                               *
1536                               *
1537                               *

```



```

1538                                TTL    PUTFPAC
1539                                *****
1540                                *                                           *
1541                                *           PUTFPACx SUBROUTINE           *
1542                                *                                           *
1543                                *   These two subroutines perform to opposite function of GETFPAC1 and *
1544                                *   GETFPAC2. Again, these routines are used to convert from the *
1545                                *   internal format used by the floating point package to a "memory" *
1546                                *   format. See the GETFPAC1 and GETFPAC2, documentation for a *
1547                                *   description of the "memory" format. *
1548                                *                                           *
1549                                *****
1550                                *
1551                                *
1552 C87C                                PUTFPAC1 EQU *
1553 C87C DC 00                                LDD  FPACC1EX  GET FPACC1 EXPONENT & UPPER 8 BITS OF MANT.
1554 C87E 7D 00 04                            TST  MANTSGN1  IS THE NUMBER NEGATIVE?
1555 C881 2B 02                                BMI  PUTFP11   YES. LEAVE THE M.S. BIT SET.
1556 C883 C4 7F                                ANDB #57F     NO. CLEAR THE M.S. BIT.
1557 C885 ED 00                                PUTFP11 STD  0,X  SAVE IT IN MEMORY
1558 C887 DC 02                                LDD  FPACC1MN+1 GET L.S. 16 BITS OF THE MANTISSA.
1559 C889 ED 02                                STD  2,X
1560 C88B 39                                RTS
1561                                *
1562                                *
1563 C88C                                PUTFPAC2 EQU *
1564 C88C DC 05                                LDD  FPACC2EX  GET FPACC1 EXPONENT & UPPER 8 BITS OF MANT.
1565 C88E 7D 00 09                            TST  MANTSGN2  IS THE NUMBER NEGATIVE?
1566 C891 2B 02                                BMI  PUTFP21   YES. LEAVE THE M.S. BIT SET.
1567 C893 C4 7F                                ANDB #57F     NO. CLEAR THE M.S. BIT.
1568 C895 ED 00                                PUTFP21 STD  0,X  SAVE IT IN MEMORY
1569 C897 DC 07                                LDD  FPACC2MN+1 GET L.S. 16 BITS OF THE MANTISSA.
1570 C899 ED 02                                STD  2,X
1571 C89B 39                                RTS
1572                                *
1573
1574

```

ADDNXTD COD2 *0229 0145 0221
 ADDNXTD1 C10E *0261 0235 0238 0246 0250 0258
 ANGRES C759 *1298 1144 1170
 ANGRES1 C760 *1309 1307
 ANGRES2 C773 *1311 1315
 ANGRES3 C780 *1316 1312
 ANGRES4 C792 *1324 1318 1320
 ANGRES5 C7A7 *1332 1326
 ASCFLT C000 *0095
 ASCFLT1 C014 *0105
 ASCFLT10 C0BC *0212 0144
 ASCFLT11 C0C1 *0218 0128 0225
 ASCFLT12 C0B6 *0206 0204
 ASCFLT13 C070 *0172 0167
 ASCFLT14 C0AD *0202 0191
 ASCFLT15 C082 *0181 0177
 ASCFLT16 C085 *0182 0179
 ASCFLT2 C01B *0111
 ASCFLT3 C02A *0122 0112
 ASCFLT4 C043 *0142 0107 0117 0147
 ASCFLT5 C039 *0132 0123 0127 0180 0185 0201
 ASCFLT6 C050 *0151 0155
 ASCFLT7 C069 *0166 0157 0163 0213 0220
 ASCFLT8 C05F *0161 0165 0223
 ASCFLT9 C08D *0186 0175
 C180OVP1 C835 *1439 1427
 CHCKO C180 *0331 0277 0315 0361 0364 0466 0472 0610 0616 0736
 0961 1021 1075
 CHCKO1 C188 *0337 0335
 CONST10 C18F *0342 0291 0772
 CONSTP1 C188 *0341 0287 0777
 CONSTP5 C549 *0919 0779
 COSFACT C7D3 *1360 1213
 DECDIG C52C *0903 0818
 DEG2RAD C813 *1415 1147 1173
 DEG2RAD1 C819 *1418 1428
 DIVOERR 0004 *0041 0612
 EXG1AND2 C7AA *1337 1217 1385
 EXPSIGN 0000 *0089 0181 0203
 FINISH C117 *0274 0168 0208
 FINISH1 C140 *0291 0286
 FINISH2 C146 *0293 0290 0295
 FINISH3 C14E *0296 0278 0285
 FLT2INT C5A1 *1019
 FLT2INT1 C5CA *1038 1027
 FLT2INT2 C5E3 *1052 1025
 FLT2INT3 C5EA *1055 1022
 FLT2INT4 C5DF *1049 1029 1039
 FLT2INT5 C5BA *1031 1035 1041
 FLTADD C23C *0463 0582 0781 1102 1275 1289 1308 1323 1330
 FLTADD1 C24C *0471 0467
 FLTADD10 C248 *0469 0543 0560
 FLTADD11 C2DC *0546 0504
 FLTADD12 C2D9 *0544 0540 0552 0557
 FLTADD2 C262 *0481 0473
 FLTADD3 C27B *0494 0485
 FLTADD4 C254 *0474 0488
 FLTADD5 C282 *0497 0493 0501
 FLTADD6 C247 *0468 0480 0495 0544
 FLTADD7 C28B *0502 0483
 FLTADD8 C2AA *0519 0506
 FLTADD9 C2CE *0539 0525
 FLTASC C3DB *0733
 FLTASC1 C3EB *0742 0737
 FLTASC10 C49B *0821 0854
 FLTASC11 C49E *0822 0830

			0928	0938	1092	1100	1111	1125	1272	1286	1309	1310	1324
			1327	1339	1340	1531	1564						
FPACC2MN	0006	*0034	0230	0232	0239	0243	0261	0263	0402	0414	0415		
			0416	0421	0426	0476	0496	0507	0510	0513	0516	0520	0523
			0547	0550	0632	0635	0649	0652	0655	0656	0657	0674	0677
			0696	0698	0931	0941	1113	1127	1274	1288	1343	1344	1533
			1569										
FPMULT1	C1C1	*0381	0376										
FPMULT2	C1C8	*0385	0377	0381									
FPMULT3	C1CF	*0388	0362	0370									
FPMULT4	C1AE	*0371	0365										
FPMULT5	C18C	*0378	0389										
FPMULT6	C1D5	*0391	0380	0384									
FPNORM	C161	*0313	0283	0539	0966								
FPNORM1	C160	*0319											
FPNORM2	C16F	*0320	0324										
FPNORM3	C17C	*0326	0316	0318									
FPNORM4	C17E	*0328	0321										
GETFP11	C85D	*1516	1514										
GETFP12	C85F	*1517	1511										
GETFP21	C873	*1530	1528										
GETFP22	C875	*1531	1525										
GETFPAC1	C850	*1509	1390	1433									
GETFPAC2	C866	*1523	0288	0292	0764	0768	0773	0780	1232	1281	1305		
			1418	1469									
GETPI	C82B	*1431											
MANTSGN1	0004	*0032	0102	0113	0371	0373	0479	0502	0505	0538	0622		
			0623	0746	0756	0758	0894	0924	0934	0997	1026	1047	1078
			1128	1153	1182	1201	1223	1240	1251	1306	1322	1329	1346
			1349	1512	1515	1554							
MANTSGN2	0009	*0035	0372	0478	0503	0583	0585	0621	0926	0935	1129		
			1270	1284	1347	1348	1526	1529	1565				
MAXNUM	C80F	*1396	1389										
N9999999	C545	*0916	0763										
NSORTERR	0006	*0043	1080										
NUMERIC	C155	*0303	0106	0116	0126	0143	0154	0162	0174	0184	0190		
			0219										
NUMERIC1	C15F	*0310	0305	0307									
ONE	C7E3	*1367	1280										
OVFERR	0002	*0039	0378	0558	0640								
P9999999	C541	*0913	0767										
PI	C7E7	*1368	1432										
PIOV180	C831	*1436	1417										
PSHFPAC2	C839	*1454	0097	0359	0464	0620	0748	1083	1143	1169	1382		
			1416	1426									
PULFPAC2	C843	*1465	0134	0298	0391	0469	0646	0892	1118	1155	1392		
			1420										
PUTFP11	C885	*1557	1555										
PUTFP21	C895	*1568	1566										
PUTFPAC1	C87C	*1552											
PUTFPAC2	C88C	*1563	1462										
PWR10EXP	0001	*0090	0187	0192	0195	0199	0202	0280	0284	0289	0294		
RAD2DEG	C823	*1425											
SINCOS	C68F	*1195	1149	1176									
SINCOS1	C68C	*1218	1226										
SINCOS2	C6D3	*1230	1259										
SINCOS3	C713	*1262	1277										
SINCOS4	C68F	*1219	1216										
SINCOS5	C73F	*1283	1279										
SINCOS6	C74E	*1289	1282										
SINCOS7	C689	*1217	1212										
SINFACT	C7C3	*1353	1203										
SINT2FLT	C589	*0984											
SINTFLT1	C595	*0992	0987										
SINTFLT2	C59F	*0998	0996										
TAN90ERR	0007	*0044	1391										

TFR1T02	C646	*1123	1091	1104	1209	1215	1383
THREE60	C7EB	*1369	1304				
TOLGSMER	0005	*0042	1049				
U1NT2FLT	C577	*0959	0992				
U1NTFLT1	C580	*0964	0962				
UMULT	C1D9	*0395	0387				
UMULT1	C1E3	*0402	0418				
UMULT2	C1F4	*0411	0404				
UMULT3	C217	*0426	0420				
UMULT4	C234	*0439	0427	0434			
UNFERR	0003	*0040	0382	0541	0664		

Using the Serial Peripheral Interface to Communicate Between Multiple Microcomputers

As the complexity of user applications increases, many designers find themselves needing multiple microprocessors to provide necessary functionality in a circuit. Communication between multiple processors can often be difficult, especially when differing processors are used. A possible solution to this problem is usage of the serial peripheral interface (SPI), an interface intended for communication between integrated circuits on the same printed wire board. The MC68HC05C4 is one of the first single-chip microcomputers to incorporate SPI into hardware. One advantage of the SPI is that it can be provided in software, allowing communication between two microcomputers where one has SPI hardware and one does not. Special interfacing is necessary when using the hardware SPI to communicate with a microcomputer that does not include SPI hardware. This interface can be illustrated with a circuit used to display either temperature or time, that incorporates both a MC68HC05C4 and a MC68705R3. The MC68HC05C4 monitors inputs from a keypad and controls the SPI data exchange, while the MC68705R3 determines temperature by performing an analog-to-digital conversion on inputs from a temperature sensor and controls the LED display. Communication between the microcomputers is handled via SPI, with the MC68HC05C4 handling exchanges in hardware, and the MC68705R3 handling them in software.

Usage of software SPI can be expanded to include circuits where the single-chip implementing the SPI in software controls the data exchange, and those in which neither single-chip has hardware SPI capability. Minor modifications to the SPI code are necessary when data exchanges are controlled by the software.

Debugging designs including multiple processors can often be confusing. Some of the confusion can be alleviated by careful planning of both the physical debugging environment and the order in which software is checked.

SERIAL PERIPHERAL INTERFACE

Communication between the two processors is handled via the serial peripheral interface (SPI). Every SPI system consists of one master and one or more slaves, where a master is defined as the microcomputer that provides the SPI clock, and a slave is any integrated circuit that receives the SPI clock from the master. It is possible to have a system where more than one IC can be master, but there can only be one master at any given time. In this design, the MC68HC05C4 is the master and the MC68705R3 is the slave. Four basic signals, master-out/slave-in (MOSI), master-in/slave-out (MISO), serial clock (SCK), and slave select (SS), are needed for an SPI. These four signals are provided on the MC68HC05C4 on port D, pins 2-5.

SIGNALS

The MOSI pin is configured as a data output on the master and a data input on the slave. This pin is used to transfer data serially from the master to a slave, in this case the MC68HC05C4 to the MC68705R3. Data is transferred most significant bit first.

Data transfer from slave to master is carried out across the MISO, master-in/slave-out, line. The MISO pin is configured as an input on the master device and an output on the slave device. As with data transfers across the MOSI line, data is transmitted most significant bit first.

All data transfers are synchronized by the serial clock. One bit of data is transferred every clock pulse, and one byte can be exchanged in eight clock cycles. Since the serial clock is generated by the master, it is an input on the slave. The serial clock is derived from the master's internal processor clock, and clock rate is selected by setting bits 0 and 1 of the serial peripheral control register to choose one of four divide-by values. Values for the MCUs crystal oscillators and the SPI divide-by must be chosen so that the SPI clock is no faster than the internal processor clock on the slave.

The last of the four SPI signals is the slave select (SS). Slave select is an active low signal, and the SS pin is a fixed input which is used to enable a slave to receive data. A master will become a slave when it detects a low level on its SS line. In this design, the MC68HC05C4 is always the master, so its SS line is tied to V_{DD} through a pull-up resistor.

REGISTERS

Three registers unique to the serial peripheral interface provide control, status, and data storage.

The Serial Peripheral Control Register (SPCR), shown below, provides control for the SPI.

	7	6	5	4	3	2	1	0	
\$0A	SPIE	SPE	—	MSTR	CPOL	CPHA	SPR1	SPRO	SPCR
RESET	0	0	0	0	0	1	U	U	

SPIE—Serial Peripheral Interrupt Enable

- 0 = SPIF interrupts disabled
- 1 = SPI interrupt if SPIF = 1

SPE—Serial Peripheral System Enable

- 0 = SPI system off
- 1 = SPI system on

MSTR—Master Mode Select

- 0 = Slave mode
- 1 = Master mode

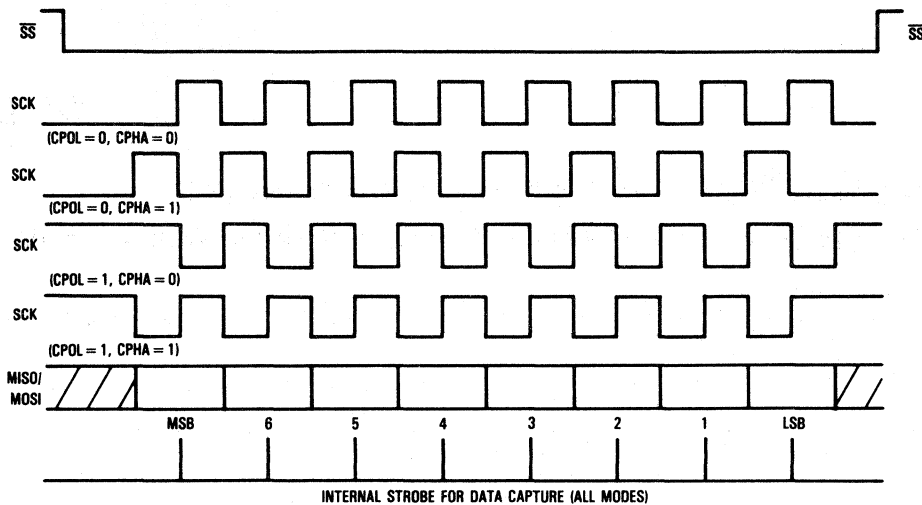


Figure 1. Data Clock Timing Diagram

CPOL—Clock Polarity

When the clock polarity bit is cleared and data is not being transferred, a steady state low value is produced at the SCK pin of the master device. Conversely, if this bit is set, the SCK pin will idle high. This bit is also used in conjunction with the clock phase control bit to produce the desired clock-data relationship between master and slave. See Figure 1.

CPHA—Clock Phase

The clock phase bit, in conjunction with the CPOL bit, controls the clock-data relationship between master and slave. The CPOL bit can be thought of as simply inserting an inverter in series with the SCK line. The CPHA bit selects one of two fundamentally different clocking protocols. When CPHA = 0, the shift clock is the OR of SCK with \overline{SS} . As soon as \overline{SS} goes low the transaction begins and the first edge on SCK involves the first data sample. When CPHA = 1, the \overline{SS} pin may be thought of as a simple output enable control. Refer to Figure 1.

SPR1 and SPR0—SPI Clock Rate Selects

These two serial peripheral rate bits select one of four baud rates (Table 1) to be used as SCK if the device is a master; however, they have no effect in the slave mode.

Table 1. Serial Peripheral Rate Selection

SPR1	SPR0	Internal Processor Clock Divide By
0	0	2
0	1	4
1	0	16
1	1	32

Data for the SPI is transmitted and received via the Serial Peripheral Data Register (SPDR). A data transfer is initiated by the master writing to its SPDR. If the master is sending data to a slave, it first loads the data into the SPDR and then transfers it to the slave. When reading data, the data bits are gathered in the SPDR and then the complete byte can be accessed by reading the SPDR.

DEMONSTRATION BOARD DESCRIPTION

A keypad input from the user is used to choose the output display function. The MC68HC05C4 monitors the keypad, decodes any valid inputs, and sends the data to the MC68705R3. If the user has requested a temperature display, the MC68705R3 sends a binary value of temperature in degrees fahrenheit to the MC68HC05C4, where the value is converted to a celcius binary coded decimal value and returned to the MC68705R3 to be displayed. The LEDs are common anode displays and are driven directly off of port B on the MC68705R3. If the user desires the circuit to function as a real-time clock, a starting time must be entered and transmitted from the MC68HC05C4 to the MC68705R3. Once the clock has been initialized, the MC68705R3 updates the clock every minute. Clock values are stored in memory, and when the circuit is functioning as a thermometer, the values in memory are updated as required to maintain clock accuracy.

USING THE A/D CONVERTOR TO MONITOR TEMPERATURE

Temperature monitoring is performed by the Motorola MTS102 silicon temperature sensor and the LM358 Dual Low-Power Operational Amplifier, as shown in the schematic in

Figure 2. Variations in the base-emitter voltage of the Motorola MTS102 silicon temperature sensor are monitored by the MC68705R3, which converts these analog inputs to equivalent digital values in degrees fahrenheit. The sensor voltage is buffered, inverted, and amplified by a dual differential amplifier before entering the A/D converter. An amplifier gain of 16 is used, resulting in 20-millivolt steps per degree fahrenheit. Using a V_{CC} of 5 volts, the maximum differential amplifier output is 3.8 volts, resulting in a temperature sensing range from -40 degrees to +140 degrees fahrenheit.

The output from the differential amplifier is connected to the A/D converter on the MC68705R3. A block diagram of the successive approximation A/D converter is shown in Figure 3. Provision is made for four separate external inputs and four internal analog channels.

Two different registers associated with the converter control channel selection, initiate a conversion, and store the result of a completed conversion. Both the external and the internal input channels are chosen by setting the lower 3 bits of the A/D Control Register (ACR). The internal input channels are connected to the V_{RH}/V_{RL} resistor chain and may be used for calibration purposes.

The converter operates continuously, requiring 30 machine cycles per conversion. Upon completion of a conversion, the digital value of the analog input is placed in the A/D result register (ARR) and the conversion complete flag, bit 7 of the ACR, is set. Another sample of the selected input is taken, and a new conversion is started.

Conversions are performed internally in hardware by a simple bisection algorithm. The D/A converter (DAC) is initially set to \$80, the midpoint of the available conversion range. This value is compared with the input value and, if the input value is larger, \$80 becomes the new minimum conversion value and the DAC is once again set to the midpoint of the conversion range, which is now \$C0. If the input value is less than \$80, \$80 becomes the maximum conversion value and the DAC is set to the midpoint of the new conversion range, in this case \$40. This process is repeated until all eight bits of the conversion are determined.

Quantizing errors are reduced to +1/2 LSB, rather than +0, -1 LSB, through usage of a built-in 1/2 LSB offset. Ignoring errors, the transition between 00 and 01 will occur at 1/2 LSB above the voltage reference low, and the transition between \$FE and \$FF will occur 1-1/2 LSBs below voltage reference high.

The A/D convertor returns a value of \$30 when given an input of zero degrees fahrenheit, so \$30 must be subtracted from the result before converting to celcius. This offset must also be considered when calibrating the sensor. Calibration of the temperature sensor can be performed by adjusting the variable resistor to produce a display of \$00 after a piece of ice has been placed on the temperature sensor for approximately one minute. A 00 display results from a value of \$50 in the ARR, so the variable resistor should be adjusted until this value is reached.

COMMUNICATION CONSIDERATIONS

In this application, an SPI read or write is initiated via an interrupt from the MCU desiring to write data. When any of

the three function keys, display temperature, set time, or display time, are pressed, the MC68HC05C4, as master, sends the MC68705R3 an interrupt on the MC68705R3's INT pin. The MC68HC05C4 writes the key value to its serial peripheral data register, thereby initiating the SPI. It then waits for the SPIF bit to go high and returns to scanning the keypad.

At the same time the MC68HC05C4 is writing to its SPDR, the MC68705R3 sets a bit counter to eight and waits for the first SCK from the MC68HC05C4. After each clock pulse, the MC68705R3 checks the status of the data bit, sets the carry bit equal to the data bit, and rotates the carry bit left into a result register. The bit counter is decremented, compared to zero, and if not zero, the MC68705R3 waits for the next clock pulse and repeats the cycle.

To ensure proper data transfers, the internal processor clock of the MC68705R3 must be sufficiently faster than the SPI clock of the MC68HC05C4 to allow the MC68705R3 time to complete this routine before the MC68HC05C4 can send another bit. This requires the user to first write the code to handle the software SPI, count machine cycles, and then choose MCU oscillator values that allow the additional machine cycles required in a software SPI to be completed before the master can send another clock pulse to the slave.

For example, consider the following piece of code for the MC68705R3, a slave receiving data from the master.

DATA IN PORTC pin 5
SCK PORTC pin 4

Cycles	Instruction	
2	LDA #08	
5	STA BITCT	Set bit counter
10	NXT BRSET 4,PORTC,*	Wait for clock transition
10	BRSET 5,PORTC,STR	Check data status
6	STR ROL RESULT	Store in result
6	DEC BITCT	Check for end of byte
4	BNE NXT	Get next bit
43		

Execution of this code requires 43 machine cycles. The maximum oscillator speed for an MC68705R3 is 1 MHz, requiring an SPI clock no greater than 1/43 MHz. One way of obtaining this rate for the SPI clock is to run the MC68HC05C4 at 0.5 MHz and choose a divide-by 32 to generate the SPI clock.

If the user has selected a temperature display, it is necessary for the MC68705R3, as a slave, to send data to the MC68HC05C4 master. When the MC68705R3 is ready to send data, it interrupts the MC68HC05C4 via the MC68HC05C4's IRQ line. The MC68HC05C4 then writes to its serial peripheral data register to initiate the transfer and shifts in data bits sent from the MC68705R3 until the SPIF bit goes high. While the MC68HC05C4 is writing to its SPDR, the MC68705R3 program is setting a bit counter to 8. When it detects a clock pulse on the SCK pin, the data register is rotated left one bit, placing the MSB in the carry. The MOSI pin is then set equal to the carry bit, the bit counter is decremented and, if it is greater than zero, the process is repeated.

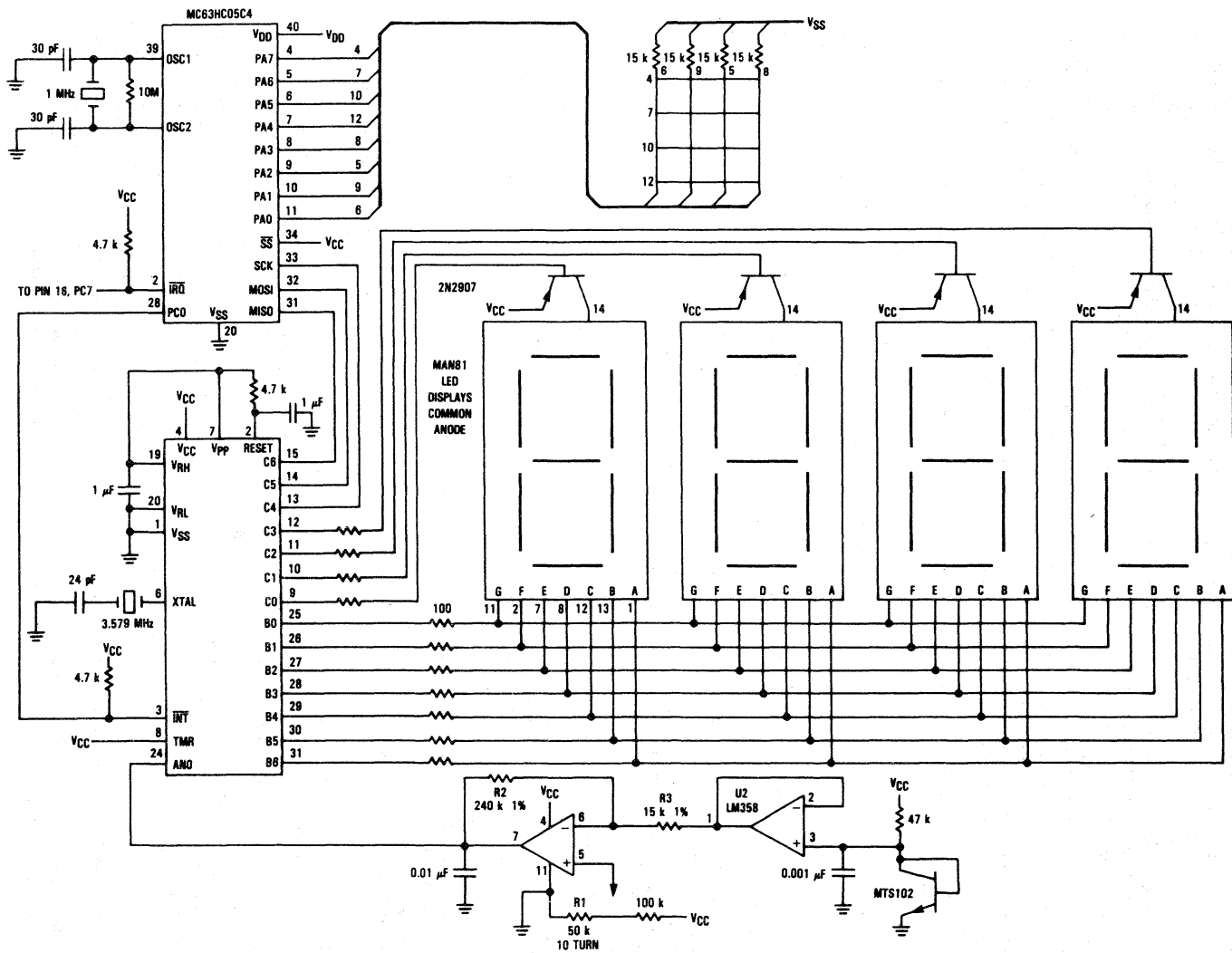


Figure 2. Serial Peripheral Interface Demonstration Schematic

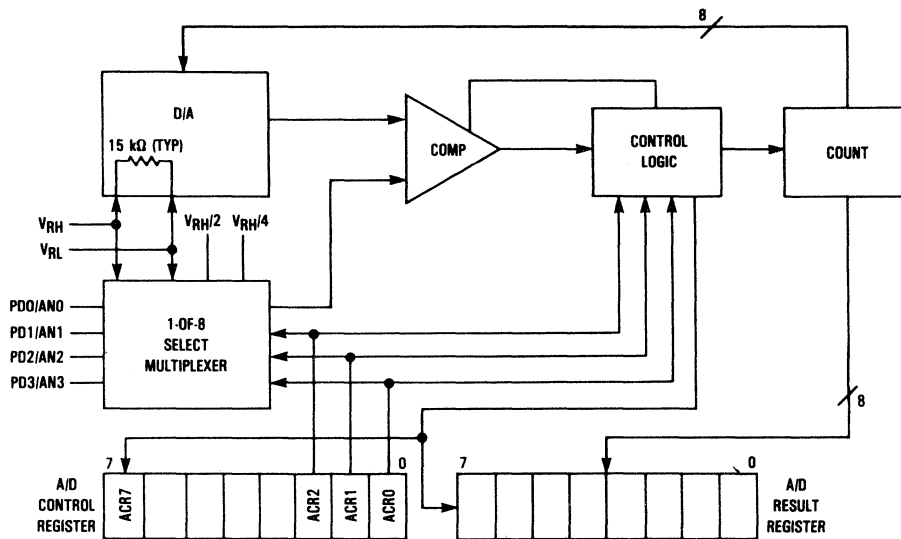


Figure 3. A/D Block Diagram

ADDITIONAL USES OF SPI

Many variations of this usage of the SPI are possible. The three possibilities are hardware SPI at both master and slave, software SPI at the master and hardware at the slave, and software SPI at both master and slave. Table 1 shows the various MCUs that have SPI implemented in hardware.

SPI is fairly straightforward in a circuit where both master and slave have hardware SPI capability. In this case, the MCUs are connected as shown in Figure 4. Figure 4a illustrates a single master system, and Figure 4b shows a system where either MCU can be system master. When both master and slave have SPI capability in hardware, data transfers can be handled full duplex. For a single master system, both master and slave write the data to be transferred to their respective serial peripheral data registers. A data transfer is initiated when the master writes to its serial peripheral data register. A slave device can shift data at a maximum rate equal to the CPU clock, so clock values must be chosen that allow the slave to transfer data at a rate equal to the master's transfer rate. In a multiple master system, the master must pull the slave's SS line low prior to writing to its serial peripheral data register and initiating the transfer.

PROGRAMMING A MASTER FOR SOFTWARE SPI

When the master in an SPI system does not have hardware SPI capabilities, the resulting system is quite different. An SPI system with a master providing the SPI in software is shown in Figure 5. This system only requires two lines between the microcomputers; data and clock. A slave select line can be added for use with multiple slaves. If operated with one data line, the SPI will function half-duplex only. Data is stored in a register, rotated left one bit at a time, and a port pin is set equal to the data bit. The master then provides the serial clock by toggling a different port pin. A bit counter must also be used to count the eight bits in the byte. Bit manipulation instructions are very useful for implementing SPI in software.

One possible software implementation for a write from the master to the slave is shown below.

DATA OUT	PORTC pin 0	
SCK	PORTC pin 1	
	LDX #08	Bit counter
	LDA DATA	Put data in register A
RPT	ROLA	Shift a data bit into carry
	BCS SET	Check for a 1
	BCLR 0,PORTC	Set data out line to 0
CLK	BSET 1,PORTC	
	BCLR 1,PORTC	Toggle clock pin
	DECX	Check for end of byte
	BNE RPT	If not, repeat
SET	BSET 0,PORTC	Set data out line to 1
	BRA CLK	Go to clock

Full duplex operation requires a second data line. One port pin is then devoted to data-out and one to data-in. Data transfer from slave to master is accomplished immediately before the SCK pin is toggled. The state of the data-in pin is tested, and the carry is then set equal to the data-in pin. This value is then rotated in to a result register. The modified code is shown below.

DATA OUT	PORTC pin 0	
SCK	PORTC pin 1	
DATA IN	PORTC pin 2	
	LDX #08	Bit counter
	LDA DATA	Put data in register A
	BCLR 1,PORTC	Clear clock pin
RPT	ROLA	Shift a data bit into carry
	BCS SET	Check for a 1
	BCLR 0,PORTC	Set data out line to 0
	BSET 1,PORTC	Set clock pin
DIN	BRCLR 3,PORTC,CLK	Check state of data
CLK	ROL DATAIN	Rotate input data one bit
	DECX	Check for end of byte
	BNE RPT	If not, repeat
SET	BSET 0,PORTC	Set data out line to 1
	BRA DIN	Go to data input

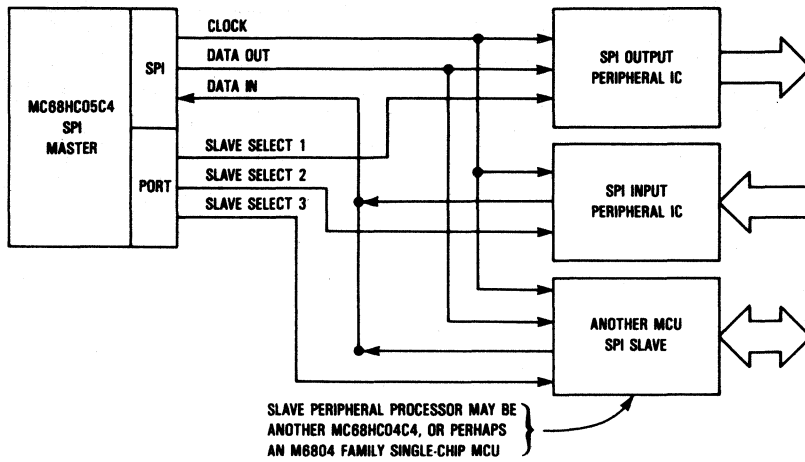


Figure 4a. Single Master SPI

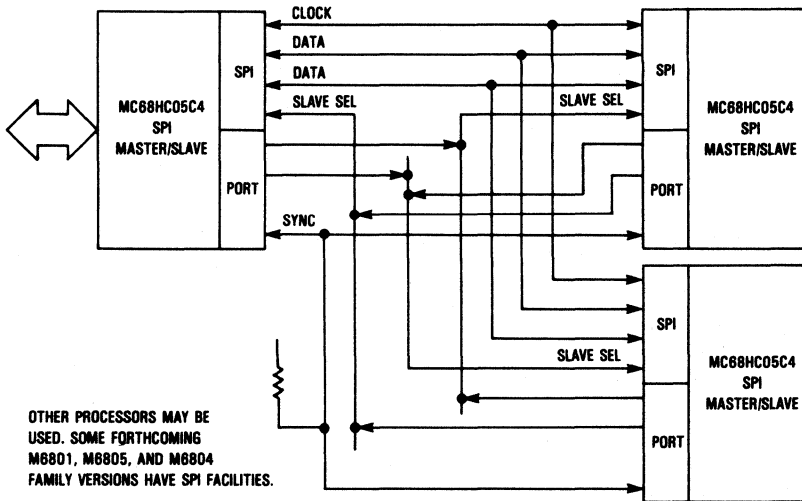


Figure 4b. Multiple Master SPI

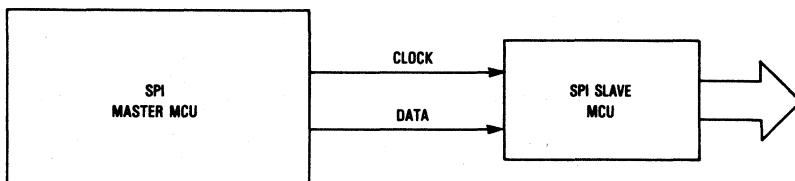


Figure 5. Software SPI

PROGRAMMING A SLAVE FOR SOFTWARE SPI

If the slave in the system is a MCU with hardware SPI capability, the data transfer will happen automatically, one bit per clock pulse. If the slave is a MCU that does not have SPI implemented in hardware, a read requires the following actions. A bit counter is set to eight, the slave polls its SCK pin waiting for a clock transition, once it perceives a clock it checks its data-in pin, sets the carry equal to the data and rotates the carry into a results register. One possible code implementation is shown in the previous timing example.

Converting this to full duplex operation requires the addition of a write from slave to master. The slave rolls a data register to place the data bit to be sent into the carry, and the data-out pin is set equal to the carry. These actions occur prior to the read of data from the master. With these modifications, the code looks as shown below.

DATA OUT	PORTC pin 6		
DATA IN	PORTC pin 5		
SCK	PORTC pin 4		
	LDA	#08	
	STA	BITCT	Set bit counter
AGN	BRSET	4,PORTC,*	Wait for clock
	ROL	RES1	Shift data to send
	BCS	SET1	Check data status
	BCLR	6,PORTC	If 0, clear data out
	BRCLR	4,PORTC,*	Wait for clock transition
	BRSET	5,PORTC,STR	Check input data status
STR	ROL	RESULT	Store in result
	DEC	BITCT	Check for end of byte
	BNE	AGN	

DEBUGGING TIPS

Debugging a circuit containing two microcomputers presents various problems not evident when working with a single microcomputer circuit. The first problem is simultaneously providing emulation for both microcomputers. Once emulation capability is arranged, the designer needs to keep track of the progress of each single-chip, and monitor how the actions of one affects the actions of the other.

One of the easiest methods to debug a circuit of this type is to use two emulator stations, complete with separate terminals. Any emulators can be used, but user confusion is reduced if the emulators have similar commands and syntax. Physical separation also helps reduce confusion. It is somewhat easier to keep track of the concurrent operations if one side of the prototype board is devoted to each single-chip and the majority of peripherals they each must interface with, and the emulator for that microcomputer is placed to that side of the printed circuit board.

Before starting simultaneous debugging, it is best to individually debug the code for each microcomputer wherever possible. Once it becomes necessary for the microcomputers to communicate with one another, halt one of the microcomputers anytime they are not actually talking and work with the remaining microcomputer. As the debugging progresses, keep in mind that an error in the function of one single-chip does not necessarily indicate an error in the corresponding code for that single-chip, but rather, the error may have been caused by an incorrect or unintended transmission from the other single-chip.

Although the aforementioned suggestions reduce debugging problems, some will remain. Long periods of debug can result in an obscuring of the separation of the functions of the two programs. It helps to take periodic breaks to get away from the system and clear the thought processes. Expect to occasionally be confused, be willing to retrace sections of code multiple numbers of times, and the debugging will proceed fairly smoothly.

CONCLUSION

The Serial Peripheral Interface can be used as a tool to interconnect to MCU with various other MCUs or peripherals, and can be used with any microcomputer. A special case occurs when one, or more, of the MCUs in a circuit do not have SPI capability in hardware. In this case, a simple software routine can be written to perform the SPI. Used in this manner, the SPI eliminates the need for costly, inconvenient parallel expansion buses and Universal Asynchronous Receiver/Transmitters (UARTs) and simplifies the design effort.

```

0001
0002          nam spicnt
0003
0004          ***** REGISTER ADDRESS DEFINITION *****
0005
0006 0000          porta equ 0
0007 0002          portc equ 2
0008 0003          portd equ 3
0009 0004          ddra equ 4
0010 0006          ddrc equ 6
0011 000A          sPCR equ $0a
0012 0008          sPSR equ $0b
0013 000C          spDR equ $0c
0014 0012          tCR equ $12
0015
0016
0017 0080          org $b0
0018
0019 0080          rwno rmb 1
0020 0081          tmpa rmb 1
0021 0082          dctr rmb 1
0022 0083          ct1 rmb 1
0023 0084          base rmb 4
0024 0088          lsb rmb 1
0025 0089          msb rmb 1
0026
0027 0020          org $20
0028
0029          ***** KEYPAD LOOKUP TABLE *****
0030
0031 0020          kypd equ *
0032
0033 0020 07          fcb $07
0034 0021 04          fcb $04
0035 0022 01          fcb $01
0036 0023 00          fcb $00
0037 0024 08          fcb $08
0038 0025 05          fcb $05
0039 0026 02          fcb $02
0040 0027 0A          fcb $0a          disp. temp.
0041 0028 09          fcb $09
0042 0029 06          fcb $06
0043 002A 03          fcb $03
0044 002B 0E          fcb $0e          set time
0045 002C 0D          fcb $0d          am
0046 002D 0C          fcb $0c          pm
0047 002E 0F          fcb $0f          disp. time
0048 002F 0B          fcb $0b          blank
0049
0050 0100          org $100 program start
0051
0052 0100 9C          start    rsp
0053 0101 3F 12          clr tcr    mask timer interrupts
0054 0103 AE 7B          ldx #$7b
0055 0105 BF 02          stx portc initialize port c
0056 0107 AE 7F          ldx #$7f
0057 0109 BF 0A          stx sPCR  set spi cont. reg.
0058 010B BF 06          stx ddrc  set c0 as output
0059 010D 3F 00          clr porta clear keypad inputs
0060 010F A6 FD          lda #$f0  set up port a
0061 0111 B7 04          sta ddra  a7-a4 out., a0-a3 in
0062 0113 9B          sei

```

0063

```

0064
0065
0066 0114 CD 01 67
0067 0117 A1 0A
0068 0119 27 0E
0069 011B A1 0E
0070 011D 27 2B
0071 011F A1 0F
0072 0121 27 06
0073 0123 A1 0B
0074 0125 27 02
0075 0127 20 EB
0076
0077
0078
0079 0129 11 02
0080 012B CD 01 59
0081 012E A1 0A
0082 0130 27 02
0083 0132 20 E0
0084
0085 0134 9A
0086 0135 20 DD
0087
0088
0089 0137 CD 01 67
0090 013A A1 0A
0091 013C 27 F9
0092 013E A1 0B
0093 0140 27 F5
0094 0142 A1 0E
0095 0144 27 F1
0096 0146 A1 0F
0097 0148 27 ED
0098 014A 11 02
0099 014C CD 01 59
0100 014F A1 0C
0101 0151 27 C1
0102 0153 A1 0D
0103 0155 27 BD
0104 0157 20 DE
0105
0106
0107
0108 0159
0109 0159 B7 0C
0110 015B 0F 0B FD
0111 015E 10 02
0112 0160 81
0113
0114
0115
0116 0161
0117 0161 BF 0C
0118 0163 0F 0B FD
0119 0166 81
0120

** check keypad **
key      jsr keypad
          cmp #$0a      check for disp. temp
          beq dtmp
          cmp #$0e      check for set time
          beq sttm
          cmp #$0f      check for disp. time
          beq dtmp
          cmp #$0b      check for disp. sec
          beq dtmp
          bra key      wait for next input

** display temp **
dtmp     bclr 0,portc  send interrupt for spi
          jsr spiwr   send byte
          cmp #$0a      check for disp. temp.
          beq clr
          bra key

clr      cli
          bra key

** set time **
nudig    jsr keypad
          cmp #$0a
          beq nudig
          cmp #$0b      look for valid digit
          beq nudig
          cmp #$0e
          beq nudig
          cmp #$0f
          beq nudig
sttm     bclr 0,portc  send int. for spi
          jsr spiwr   send value
          cmp #$0c      check for pm
          beq key      yes, wait for next input
          cmp #$0d      check for am
          beq key      yes, wait for next input
          bra nudig    get next time digit

** spi write subroutine **
spiwr    equ *
          sta spdr     put data in data reg.
          bclr 7,spsr,* wait for end of byte
          bset 0,portc
spiflg   rts         done

** spi read subroutine **
spird    equ *
          stx spdr     initiate transfer
          brclr 7,spsr,* wait for end of byte
rdend    rts

```

```

0121
0122
0123 0167
0124
0125
0126
0127 0167 A6 20
0128 0169 B7 B3
0129 0168 A6 32
0130 0160 4A
0131 016E 26 FD
0132 0170 3A B3
0133 0172 26 F7
0134 0174 5F
0135 0175 A6 80
0136 0177 B7 00
0137 0179 B6 00
0138 0178 A4 0F
0139 0170 A1 00
0140 017F 26 09
0141 0181 37 00
0142 0183 5C
0143 0184 A3 03
0144 0186 23 F1
0145 0188 20 00
0146
0147
0148
0149 018A B7 B1
0150 018C BF B0
0151 018E A6 FF
0152 0190 4A
0153 0191 26 FD
0154 0193 B6 00
0155 0195 A4 0F
0156 0197 B1 B1
0157 0199 26 CC
0158
0159
0160
0161 019B B6 00
0162 019D A4 0F
0163 019F A1 00
0164 01A1 26 F8
0165
0166
0167
0168 01A3 B6 B1
0169 01A5 5F
0170 01A6 44
0171 01A7 25 03
0172 01A9 5C
0173 01AA 20 FA
0174 01AC 58
0175 01AD 58
0176 01AE 9F
0177 01AF BB B0
0178 0181 97
0179 0182 E6 20
0180 0184 B7 B1
0181 0186 81
0182

** keypad scanning routine **
keypad equ *

** 32 msec delay **

wtmp lda #$20      set up outer loop
      sta ct1      counter
otlp  lda #$32      set up inner loop
inlp  decb dec.    inner loop
      bne inlp     when 0,
      dec ct1     decrement outer loop
      bne otlp
      cirx       set up row counter
      lda #$80   check first row
      sta porta
nxtc  lda porta     check for key
      and #$0f   mask upper nibble
      cmp #$00   look for zero
      bne debnc  branch if have a key
      asr porta  try next row
      incx      decrement row counter
      cpx #$03  check for zero
      bis nxtc  test next row
inval bra wtmp     no key pressed

** debounce key input **
debnc sta tmpa     save value
      stx rwno    save row number
      lda #$ff   set up delay
loop  decb
      bne loop   wait
      lda porta  check row again
      and #$0f  mask upper nibble
      cmp tmpa  check for same key
      bne wtmp  return if invalid

** wait for key release **
wtr   lda porta    check value
      and #$0f    mask upper nibble
      cmp #$00    look for zero
      bne wtr     wait for release

** decode key value **
      lda tmpa     restore value
      cirx       set up column ctr.
nxtc  lsra        shift columns
      bcs col    branch if have column
      incx
      bra nxtc   try next column
col   lsix
      lsix       x=4*col. no.
      txa        place x in a
      add rwno   key value =4*col + row
      tax        place a in x
      lda kypd,x
      sta tmpa
      rts

```

```

0183          ***** temperature conversion routine *****
0184          *
0185          * farenheit value is received from 705r3 via
0186          * spi and received in the a register.  the value
0187          * is converted to celcius and the leftmost
0188          * led is blanked.
0189
0190 0187          r3int    equ *
0191 01B7 CD 01 61          jsr spird    read value
0192 01BA B6 0C          lda spdr     transfer value to register
0193 01BC AD 20          sub #$20    subtract 32
0194 01BE 24 05          bhs conv   if pos. convert
0195 01C0 40             nega      negate
0196 01C1 AE 0A          ldx #$0a
0197 01C3 BF B6          stx base+2  '-' pattern
0198
0199          *** temperature conversion ***
0200          ** a 16-bit multiply by 5 is performed on the
0201          ** value received from the r3.  this number
0202          **.is then divided by 9.
0203
0204 01C5 3F B9          conv    clr msb      clear counters
0205 01C7 3F B8          clr lsb
0206 01C9 B7 B8          sta lsb
0207 01CB 48             lsla      multiply by 2
0208 01CC 39 B9          rol msb   load overflow into msb
0209 01CE 48             lsla      multiply by 2
0210 01CF 39 B9          rol msb   load overflow into msb
0211 01D1 BB B8          add lsb   a contains value x5
0212 01D3 24 02          bcc div   if overflow, inc msb
0213 01D5 3C B9          inc msb
0214 01D7 3F B2          div     clr dctr
0215 01D9 B7 B8          sta lsb
0216 01DB 98             cic
0217 01DC 3C B2          nxt9    inc dctr
0218 01DE B7 B8          sta lsb
0219 01E0 B6 B9          lda msb
0220 01E2 A2 00          sbc #$00  subtract borrow from msb
0221 01E4 B7 B9          sta msb
0222 01E6 B6 B8          lda lsb   count factors of 9
0223 01E8 AD 09          sub #$09
0224 01EA 24 F0          bcc nxt9  if no borrow, repeat
0225 01EC 3D B9          tst msb   if borrow, check for end
0226 01EE 26 EC          bne nxt9  repeat if not end
0227
0228
0229          *** end of divide, add last 9 back in and
0230          *** check remainder for rounding
0231
0232 01F0 AB 09          add #$09  find remainder
0233 01F2 A1 04          cmp #$04  if greater
0234 01F4 22 02          bhi done  than 4, round up
0235 01F6 3A B2          dec dctr
0236 01F8 B6 B2          done    lda dctr
0237
0238 01FA AE 0B          pos     idx #$0b  blank pattern
0239 01FC BF B7          stx base+3  blank most sig. digit
0240
0241          **** convert binary value to bcd value ****
0242          *
0243          * the x registers begins with the binary value
0244          * and exits with zero.  each digit, units, tens

```


0245			* and hundreds, is stored separately and checked
0246			* for a value equal to 10.
0247			
0248	01FE	97	tax place a into x
0249	01FF	4F	clra
0250	0200	3F B5	clr base+1 clear values
0251	0202	3F B6	clr base+2
0252	0204	5D	st tstx check for end
0253	0205	27 17	beq send if complete, send to r3
0254	0207	5A	decx decrement hex number
0255	0208	4C	inca increment decimal number
0256	0209	A1 0A	cmp #\$0a equal to 10?
0257	020B	26 F7	bne st no, keep going
0258	020D	3C B5	inc base+1 increment tens
0259	020F	B6 B5	lda base+1 test for 10
0260	0211	A1 0A	cmp #\$0a
0261	0213	27 03	beq hund if equal, set hundreds
0262	0215	4F	zero clra clear ones
0263	0216	2D EC	bra st count next 10
0264	0218	3F B5	hund clr base+1 clear tens
0265	021A	3C B6	inc base+2 increment hundreds
0266	021C	2D F7	bra zero
0267			
0268			* send all digits to 705r3 via spi
0269			* start by interrupting r3 and then
0270			* sequentially sending four values
0271			
0272	021E	B7 B4	send sta base store ones
0273	0220	B6 B6	lda base+2
0274	0222	27 0B	beq blk
0275	0224	E6 B4	nxt dg lda base,x start at base
0276	0226	CD 01 59	jsr spiwr send to r3
0277	0229	5C	incx
0278	022A	A3 03	cp x #\$03 look for end
0279	022C	26 F6	bne nxt dg if no, next digit
0280	022E	8D	rti
0281	022F	A6 0B	blk lda #\$0b
0282	0231	B7 B6	sta base+2
0283	0233	2D EF	bra nxt dg
0284			
0285			
0286			*** initialize interrupt vectors ***
0287			
0288	1FF4		org \$1ff4
0289			
0290	1FF4	01 00	spivec fdb start
0291	1FF6	01 00	scivec fdb start
0292	1FF8	01 00	tmrvec fdb start
0293	1FFA	01 B7	irqvec fdb r3int
0294	1FFC	01 00	swivec fdb start
0295	1FFE	01 00	reset fdb start

```

0001
0002          nam r3disp
0003
0004          ***** REGISTER DEFINITION *****
0005
0006 0001          portb equ 1
0007 0002          portc equ 2
0008 0005          ddrb equ 5
0009 0006          ddrc equ 6
0010 0008          tdr equ 8
0011 0009          tcr equ 9
0012 000E          acr equ 14
0013 000F          arr equ 15
0014
0015
0016 0040          org $40
0017
0018
0019 0040          wrdat rmb 1
0020 0041          timtmp rmb 1
0021 0042          ct rmb 1
0022 0043          ct1 rmb 1
0023 0044          result rmb 1
0024 0045          res1 rmb 1
0025 0046          bitct rmb 1
0026 0047          sec rmb 1
0027 0048          segmnt rmb 1
0028 0049          pm rmb 1
0029 004A          base rmb 4
0030
0031 0080          org $80
0032
0033          **** display look-up table ****
0034
0035 0080          segtab equ *
0036 0080 01          fcb %00000001 0
0037 0081 4F          fcb %01001111 1
0038 0082 12          fcb %00010010 2
0039 0083 06          fcb %00000110 3
0040 0084 4C          fcb %01001100 4
0041 0085 24          fcb %00100100 5
0042 0086 20          fcb %00100000 6
0043 0087 0F          fcb %00001111 7
0044 0088 00          fcb %00000000 8
0045 0089 0C          fcb %00001100 9
0046 008A 7E          fcb %01111110 -
0047 008B 7F          fcb %01111111 blank
0048 008C 7F          fcb %01111111 pm
0049 008D 18          fcb %00011000 p
0050
0051 0090          org $90 program start
0052
0053          *** initialize variables ***
0054
0055 0090          start equ *
0056 0090 A6 07          lda #$07
0057 0092 C7 0F 38          sta $f38 set MOR
0058 0095 A6 FF          lda #$ff
0059 0097 B7 05          sta ddrb set up port b as output
0060 0099 B7 08          sta tdr set timer for prescale of 128
0061 009B B7 4A          sta base
0062 009D B7 48          sta base+1 blank time display

```

```

0063 009F B7 4C      sta base+2
0064 00A1 B7 4D      sta base+3
0065 00A3 A6 EF      lda #$ef
0066 00A5 B7 02      sta portc          set portc to choose msd
0067 00A7 B7 48      sta segmnt
0068 00A9 A6 CF      lda #$cf
0069 00AB B7 06      sta ddrcc          set up c0-3,c6,c7 as outputs
0070 00AD A6 0F      lda #$0f
0071 00AF B7 09      sta tcr            unmask timer interrupt
0072 00B1 3F 41      clr timtmp         start with time disp.
0073 00B3 3F 47      clr sec            set seconds to zero
0074 00B5 3F 49      clr pm             start with am
0075 00B7 A6 38      lda #$3b
0076 00B9 B7 42      sta ct             set up timing loops
0077 00BB A6 08      lda #$08
0078 00BD B7 43      sta ct1
0079 00BF 9A         cli
0080
0081                *      delay for
0082
0083 00C0 AE FF      dlay  ldx #$ff
0084 00C2 A6 FF      lda  #$ff
0085 00C4 4A         deca
0086 00C5 26 FD      bne dlay+4
0087 00C7 5A         decx
0088 00C8 26 F8      bne dlay+2
0089
0090                *      temperature measurement *
0091
0092 00CA 3F 0E      clr acr            clear conv. complete flag
0093 00CC B6 0E      lda  acr
0094 00CE 2A FC      bpl *-2
0095 00D0 B6 0F      lda  arr            get result
0096 00D2 A2 30      sbc  #$30           adjust so 0 deg = $30
0097 00D4 B7 45      sta res1           store in spi data register
0098 00D6 B6 41      lda  timtmp
0099 00D8 A1 07      cmp  #$07           check for temp. update
0100 00DA 26 E4      bne dlay
0101
0102                ***      send temperature value to hc05c4 for
0103                *      conversion into celcius. start by
0104                *      interrupting the hc05c4 and then transmit
0105                *      data via the spi.
0106
0107 00DC 9B         sei
0108 00DD 1F 02      bclr 7,portc       interrupt hc05c4
0109 00DF CD 01 18    jsr  spiwr         write data to hc05c4
0110 00E2 AE 04      lda  #$04
0111
0112                *      wait for return data ~ 140 cycles *
0113
0114 00E4 A6 0B      lda  #$0b
0115 00E6 B7 51      sta base+7
0116 00E8 A6 0E      lda  #$0e
0117 00EA 4A         timp  deca
0118 00EB 26 FD      bne timp
0119
0120                *      get decimal values in celcius from
0121                *      hc05c4
0122
0123 00ED CD 01 01    nxtdg jsr  spird
0124 00F0 B6 44      lda  result        get value

```

```

0125 00F2 E7 4A      sta base,x      store
0126 00F4 5C        incx
0127 00F5 A3 07      cpx #$07        check for end
0128 00F7 26 F4      bne nxdg
0129 00F9 9A        cli
0130 00FA 20 C4      bra dlay
0131
0132                **      select temperature display **
0133
0134 00FC A6 07      temp   lda #$07
0135 00FE 87 41      sta timtmp      choose temp. display
0136 0100 80        rti
0137
0138
0139                *****      spi routines *****
0140                *      the three pins used for the spi are
0141                *      miso bit 6, portc
0142                *      mosi bit 5, portc
0143                *      sck bit 4, portc
0144                *      the r3 waits for a high-to-low
0145                *      transition of the spi clock, which
0146                *      is provided by the hc05c4 and sent
0147                *      on portc pin 4. a bit of data is
0148                *      transferred on each high-to-low
0149                *      transition of the clock.
0150
0151                *      spi read *
0152
0153 0101      spird equ *
0154 0101 A6 08      lda #$08
0155 0103 87 46      sta bitct      set bit counter
0156 0105 08 02 FD  nxt   brset 4,portc,* wait for clock transition
0157 0108 0A 02 00  brset 5,portc, str check data status
0158                *
0159                *      note: the brset command automatically
0160                *      sets the carry bit to be equal to the
0161                *      bit under test
0162                *
0163 0108 39 44      str   rol result      store in result
0164 0100 A6 02      lda #$02        delay loop
0165 010F 4A        stall  deca
0166 0110 26 FD      bne stall
0167 0112 9D        nop
0168 0113 3A 46      dec bitct      check for end of byte
0169 0115 26 EE      bne nxt        get next bit
0170 0117 81        rts
0171
0172                *      spi write *
0173                *      data to be sent is in result at
0174                *      start of write
0175
0176 0118      spiwr equ *
0177 0118 A6 08      lda #$08
0178 011A 87 46      sta bitct      set bit counter
0179 011C 39 45      agn   rol res1        shift data
0180 011E 25 12      bcs set1      check data status
0181 0120 1D 02      bclr 6,portc   if 0, clear miso
0182 0122 1E 02      bset 7,portc   clear interrupt
0183 0124 19 02      bclr 4,portc
0184 0126 9D        nop
0185 0127 9D        nop            timing delay.
0186 0128 08 02 FD  tst   brset 4,portc,* wait for clock trans.

```

0187	012B	3A	46		dec bitct	check for end of byte
0188	012D	26	ED		bne agn	
0189	012F	1E	02		bset 7,portc	clear interrupt
0190	0131	81			rts	
0191						
0192	0132	1C	02	set1	bset 6,portc	if 1, set miso
0193	0134	1E	02		bset 7,portc	clear interrupt
0194	0136	18	02		bset 4,portc	
0195	0138	20	EE		bra tst	
0196						
0197				**	initialization of data read via spi **	
0198				*		
0199				*		
0200				*		a data read is initiated via an interrupt
0201				*		from the hc05c4. the value received is
0202				*		tested to determine which function is
0203				*		requested and the processor jumps to the
0204				*		proper routine.
0205				*		
0206	013A	CD	01 01	c4int	jsr spird	get value
0207	013D	A6	03		lda #\$03	
0208	013F	B7	41		sta timtmp	choose time
0209	0141	B6	44		lda result	
0210	0143	A1	0A		cmp #\$0a	check for disp temp
0211	0145	27	85		beq temp	
0212	0147	A1	0F		cmp #\$0f	check for display time
0213	0149	27	3C		beq rtry	
0214	014B	A1	0E		cmp #\$0e	check for set time
0215	014D	27	39		beq clrtm	
0216	014F	A1	0D		cmp #\$0d	check for am
0217	0151	27	0C		beq am	
0218	0153	A1	0B		cmp #\$0b	check for secs
0219	0155	27	6A		beq dsec	
0220	0157	A1	0C		cmp #\$0c	check for pm
0221	0159	26	39		bne dig	no, set digit
0222	015B	A6	FF		lda #\$ff	set pm address
0223	015D	B7	49		sta pm	
0224						
0225				**	check for valid input **	
0226						
0227	015F	B6	4D	am	lda base+3	check tens of hours
0228	0161	27	08		beq blhr	if zero, blank digit
0229	0163	A1	01		cmp #\$01	
0230	0165	27	46		beq twoc	
0231	0167	A1	0B		cmp #\$0b	look for blank
0232	0169	26	48		bne blank	if not, blank display
0233	016B	A6	0B	blhr	lda #\$0b	
0234	016D	B7	4D		sta base+3	blank tens of hours
0235	016F	B6	4B	mtn	lda base+1	check tens of minutes
0236	0171	A1	05		cmp #\$05	check against 5
0237	0173	22	3E		bhi blank	if greater, blank display
0238						
0239				*	valid input, set timer counter *	
0240						
0241	0175	A6	0F		lda #\$0f	
0242	0177	B7	09		sta tcr	unmask timer interrupt
0243	0179	A6	43		lda #\$43	
0244	017B	B7	42		sta ct	load inner loop counter
0245	017D	A6	06		lda #\$06	
0246	017F	B7	43		sta ct1	load outer loop counter
0247	0181	3F	47		clr sec	
0248	0183	3F	52		clr base+8	

```

0249 0185 3F 53          clr base+9
0250 0187 80          rtry  rti
0251
0252          *          clear displays *
0253
0254 0188 A6 0B        clrtn  lda #$0b
0255 018A B7 4A          sta base
0256 018C B7 4B          sta base+1
0257 018E B7 4C          sta base+2
0258 0190 B7 4D          sta base+3
0259 0192 20 F3        bra rtry
0260
0261
0262          *          input digit *
0263
0264          ***          time setting routine ***
0265          *          time is inputted left to right
0266          *          and the end of input is indicated
0267          *          by pressing either the am or pm
0268          *          button. pm is denoted on the
0269          *          display by lighting the decimal
0270          *          point. counters are set to zero out
0271          *          after each second.
0272          *
0273
0274
0275 0194 9A          dig  cli
0276 0195 B6 4C          lda base+2
0277 0197 B7 4D          sta base+3
0278 0199 B6 4B          lda base+1          shift data left one
0279 019B B7 4C          sta base+2          digit
0280 019D B6 4A          lda base
0281 019F B7 4B          sta base+1
0282 01A1 B6 44          lda result          enter digit 1
0283 01A3 B7 4A          sta base
0284 01A5 A1 09          cmp #$09          check for valid digit
0285 01A7 22 DE          bhi rtry
0286 01A9 B7 4A          sta base
0287 01AB 20 DA          bra rtry          get next number
0288
0289          *          check if time less than 12 o'clock
0290          *          blank display if not
0291
0292 01AD B6 4C          twoc  lda base+2          check hours units
0293 01AF A1 02          cmp #$02
0294 01B1 23 BC          bls mtn          okay, check tens of min.
0295 01B3 3F 4D          blank  clr base+3
0296 01B5 3F 4C          clr base+2
0297 01B7 A6 0D          lda #$0d
0298 01B9 B7 4B          sta base+1          send error message
0299 01BB A6 05          lda #$05
0300 01BD B7 4A          sta base
0301 01BF 20 C6          bra rtry
0302
0303          **** seconds display ****
0304          *          blank first two leds
0305          *
0306 01C1 B7 41          dsec  sta timtmp          set timtmp to $0b
0307 01C3 B7 55          sta base+$b          blank 1st two leds
0308 01C5 B7 54          sta base+$a
0309 01C7 80          rti
0310

```

```

0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321 01C8
0322
0323 01C8 99
0324 01C9 BE 41
0325 01CB A6 FF
0326 01CD B7 01
0327 01CF 36 48
0328 01D1 25 04
0329
0330
0331 01D3 A6 F7
0332 01D5 B7 48
0333 01D7 E6 4A
0334 01D9 5A
0335 01DA 02 48 02
0336 01DD E6 4A
0337 01DF 5A
0338 01E0 04 48 02
0339 01E3 E6 4A
0340 01E5 5A
0341 01E6 06 48 02
0342 01E9 E6 4A
0343 01EB A4 0F
0344 01ED 97
0345 01EE EE 80
0346 01F0 8F 01
0347 01F2 B6 48
0348 01F4 B7 02
0349
0350
0351
0352
0353
0354
0355 01F6 A6 10
0356 01F8 B7 08
0357 01FA A6 0F
0358 01FC B7 09
0359 01FE 3A 42
0360 0200 26 08
0361 0202 A6 3B
0362 0204 B7 42
0363 0206 3A 43
0364 0208 27 01
0365 020A 80
0366
0367
0368
0369
0370
0371
0372

*****      display routine *****
*
*      displays are refreshed every msec
*      when a timer interrupt occurs.  the
*      most significant digit is displayed
*      first.  at the conclusion of each
*      minute, the time is updated
*

tmrint equ *

sec
ldx timtmp      choose time or temp
lda #$ff        blank displays
sta portb       send to leds
ror segmnt      select display
bcs min2        look for restart

lda #$f7        restart with msd
sta segmnt
min2 lda base,x  load a with minutes
     decx        point to next digit
     brset 1,segmnt,hrs1  check hours units
     lda base,x  load a with tens of min.
     hrs1 decx    point to next digit
     brset 2,segmnt,hrs2  check tens of hrs.
     lda base,x  load a with hours units
     hrs2 decx    point to next digit
     brset 3,segmnt,disp  display value
     lda base,x  load a with tens of hrs
     disp and #$0f  mask upper nibble
     tax         set x equal to a
     ldx segtab,x  display value table
     stx portb    enable display drivers
     lda segmnt
     sta portc    enable display

**      count display refreshes.  402 refreshes
*      equals one second.  after 402 refreshes,
*      update clock
*

lda #$10        set timer to interrupt
sta tdr         after 2048 cycles
lda #$0f
sta tcr         reset timer interrupt flag
dec ct         decrement inner loop
bne ret
lda #$3b
sta ct         reset inner loop
dec ct1        decrement outer loop
beq tmchg      if one sec., to time change
ret            rti

*****      time change routine *****
*
*      when 60 seconds are counted,
*      increase minutes by one, if
*      necessary, blank minutes and increase
*      hours.  change am/pm if needed.
*

```

0373						
0374	0208			tmchg	equ *	
0375	0208	3C	47		inc sec	increase seconds
0376	0200	3C	52		inc base+8	inc. secs. units
0377	020F	B6	52		lda base+8	
0378	0211	A1	0A		cmp #\$0a	look for ten
0379	0213	27	38		beq tens	if yes, inc. tens
0380	0215	A6	3C	minck	lda #\$3c	look for a minute
0381	0217	B1	47		cmp sec	
0382	0219	26	54		bne ret1	wait for next second
0383	0218	3F	47		clr sec	
0384	0210	3F	53		clr base+9	zero display
0385	021F	B6	4A		lda base	check min. units
0386	0221	A1	09		cmp #\$09	check tens of min. units
0387	0223	26	20		bne inm1	increase
0388	0225	3F	4A		clr base	min. units = 0
0389	0227	B6	4B		lda base+1	check tens of min. units
0390	0229	A1	05		cmp #\$05	less than 5?
0391	022B	26	1C		bne inm2	increase
0392	0220	3F	4B		clr base+1	tens of min =0
0393	022F	B6	4D		lda base+3	check tens of hrs.
0394	0231	A1	0B		cmp #\$0b	look for blank
0395	0233	27	1E		beq hrck	less than 10:00
0396	0235	B6	4C		lda base+2	check hrs. units
0397	0237	A1	02		cmp #\$02	less than 2?
0398	0239	26	2A		bne inhr1a	increase
0399	023B	A6	0B		lda #\$0b	
0400	023D	B7	4D		sta base+3	set time to 1:00
0401	023F	A6	01		lda #\$01	
0402	0241	B7	4C		sta base+2	
0403	0243	20	2A		bra ret1	done
0404	0245	3C	4A	inm1	inc base	increase min. units
0405	0247	20	26		bra ret1	done
0406	0249	3C	4B	inm2	inc base+1	increase tens of min.
0407	024B	20	22		bra ret1	
0408						
0409	024D	3F	52	tens	clr base+8	zero sec. units
0410	024F	3C	53		inc base+9	inc sec. tens
0411	0251	20	C2		bra minck	
0412						
0413				*	increase hours *	
0414						
0415	0253	B6	4C	hrck	lda base+2	check hours units
0416	0255	A1	09		cmp #\$09	less than 9?
0417	0257	26	0B		bne inhr1	increase
0418	0259	3F	4C		clr base+2	hours units =0
0419	025B	3F	4D		clr base+3	
0420	025D	3C	4D		inc base+3	tens of hours =1
0421	025F	20	0E		bra ret1	done
0422	0261	3C	4C	inhr1	inc base+2	increase hours units
0423	0263	20	0A		bra ret1	done
0424	0265	3C	4C	inhr1a	inc base+2	increase hours units
0425	0267	B6	4C		lda base+2	check value
0426	0269	A1	02		cmp #\$02	for 12:00
0427	026B	26	9D		bne ret	no, done
0428	026D	33	49		com pm	switch pm indicator
0429	026F	A6	3B	ret1	lda #\$3b	
0430	0271	B7	42		sta ct	reset inner loop counter
0431	0273	A6	0B		lda #\$0B	
0432	0275	B7	43		sta ct1	reset outer loop counter
0433	0277	B0			rti	
0434						


```
0435
0436
0437
0438 0FF8
0439
0440 0FF8 01 C8
0441 0FFA 01 3A
0442 0FFC 00 90
0443 0FFE 00 90

*** initialize interrupt vectors ***
org $ff8
tmrvec fdb tmrint
intvec fdb c4int
swivec fdb start
reset fdb start
```

MC68HC11 EEPROM Programming from a Personal Computer

This application note describes a simple and reliable method of programming either the MC68HC11's internal EEPROM, or EEPROM connected to the MCU's external bus. The data to be programmed is downloaded from any standard personal computer (PC) fitted with a serial communications port. In addition to the programming procedure, the software incorporates the facility to verify the contents of the MCU's internal or external memory against code held on a PC disc. Both program and verify options use data supplied in S record format, which is downloaded from the PC to the MC68HC11 using the RS232 protocol supported by the MCU's SCI port.

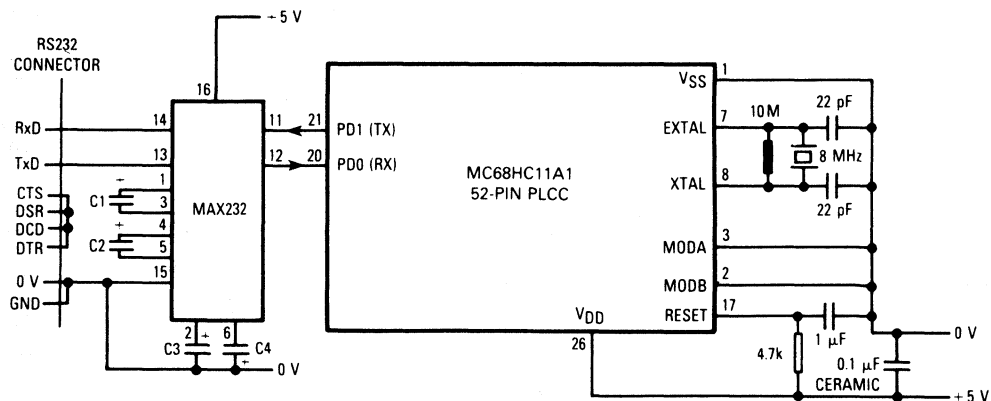
The minimum MCU configuration required to program the MC68HC11's internal EEPROM is shown in Figure 1. This consists only of the MCU, an RS232 level shifting circuit, plus an 8 MHz crystal and a few passive components.

To initiate the download, the PC is connected to the MC68HC11 SCI transmit and receive lines via a level shifter. The circuit of Figure 1 uses a Maxim MAX232 to

eliminate the need for additional ± 12 V supplies. The MCU's special bootstrap mode is invoked by applying a logic zero to the MODA and MODB pins, followed by a hardware RESET.

Removing the RESET condition causes the MCU to start execution of its bootloader program, located in internal ROM, between addresses \$BF40 and \$BFFF. In normal single-chip or expanded modes, the boot ROM is not accessible, and reads from these memory locations will result respectively in irrelevant data or external memory fetches.

An additional consequence of bootstrap operation is that all vectors are relocated to the boot ROM area. With the exception of the RESET vector, which points to the start of the boot ROM, the remaining interrupt vectors all point to an uninitialized jump table in RAM. Three bytes are reserved for each entry in the jump table, to allow for an extended jump instruction. Tables 1 and 2 detail the memory map of the bootstrap vectors and an example RAM jump table.



C1, C2, C3, C4 — 22 μ F 25 V Aluminium or Tantalum

NOTE: To improve reliability of the MCU, all its unused inputs should be connected to VSS or VDD

Figure 1. MC68HC11 Bootstrap Mode Connection to RS232 Line

Table 1. Bootstrap Vector Assignments

Boot ROM		
Address	Vector	Description
BFFE	BF40	Bootstrap Reset
BFFC	00FD	Clock Monitor
BFFA	00FA	COP Fail
BFF8	00F7	Illegal Opcode
BFF6	00F4	SWI
BFF4	00F1	XIRQ
BFF2	00EE	IRQ
BFF0	00EB	Real Time Interrupt
BFEE	00E8	Timer Input Capture 1
BFEC	00E5	Timer Input Capture 2
BFEA	00E2	Timer Input Capture 3
BFE8	00DF	Timer Output Compare 1
BFE6	00DC	Timer Output Compare 2
BFE4	00D9	Timer Output Compare 3
BFE2	00D6	Timer Output Compare 4
BFE0	00D3	Timer Output Compare 5
BFDE	00D0	Timer Overflow
BFDC	00CD	Pulse Accumulator Overflow
BFDA	00CA	Pulse Accumulator Input Edge
BFD8	00C7	SPI
BFD6	00C4	SCI

Table 2. RAM Jump Table

Internal RAM	
Address	Typical Instruction
00FD	JMP CLKMON
00FA	JMP COPFL
....etc	

Note that, before any interrupts are enabled in bootstrap mode, it is the software designer's responsibility to initialize all appropriate entries in the jump table.

As this application note does not make use of the MC68HC11's interrupt system, the jump table is not set up.

The bootstrap program continues by initializing the SCI transmitter and receiver to 7812 baud and proceeds to examine the state of the NOSEC bit in the CONFIG register. If this is at logic zero (security enabled) the boot-loader will erase the entire contents of internal EEPROM and also the CONFIG register.

This feature is particularly useful for security conscious applications, where the internal EEPROM contains information of a proprietary or confidential nature. If the NOSEC bit is at logic one, then the erasing sequence is not carried out.

Note also that erasing the CONFIG register disables the security feature.

The bootstrap program then issues a break condition on the SCI transmit line, and waits for the reception of the first byte. In this application, no use is made of the break transmitted by the SCI.

At this point, it is necessary to initiate the PC S record downloader program, called EELoad.BAS (written in BASIC). It will display a header message, and prompt the user for the number of the COM channel (either one or two) which is connected to the MC68HC11. A listing of EELoad.BAS is given at the back of this application note.

The PC-resident program will now configure the appropriate COM channel to 1200 baud, one stop bit, no parity, and download the binary file EEPROGIX.BOO from the PC to the MC68HC11.

The MC68HC11's boot-loader will automatically detect the fact that the first incoming character is received at a different baud rate, and change its SCI rate to 1200 baud.

It will then proceed to load the binary file into all 256 RAM locations and then jump to address \$0000 (i.e., the first RAM location).

EEPROGIX.BOO consists of the MC68HC11 executable code shown in the source listing at the back of this application note, with the addition of \$FF at the head of the file, and \$00 appended up to the 256th byte. This program is designed to receive S records from the PC and program the data fields into the appropriate EEPROM memory locations.

A point to note is that the initial \$FF byte in EEPROGIX.BOO is only used to detect the baud rate of the PC, and is not echoed back, while the remaining 256 bytes are echoed by the MC68HC11's SCI transmitter. However, during download of EEPROGIX.BOO, the PC does not detect the echo, as this feature is unnecessary at this stage.

Once the newly downloaded S record programmer starts execution in the MC68HC11, it configures the SCI to 9600 baud, then waits for a control character from the PC. This character will determine the operating mode of the S record programmer. The options available are shown in Table 3. Note that these programming utilities can be used to load and verify external RAM as well as external EEPROM.

Table 3. S Record Downloader Operating Mode Options

Control Character	Operating Mode
X	Program External EEPROM RAM
I	Program Internal EEPROM
V	Verify Internal or External EEPROM RAM

If the S record programmer has been downloaded successfully, the PC resident program will now —

1. Request whether the downloaded data must be echoed to the screen.
2. Prompt the user for the required operating mode.
3. Request the name of the S record file to be downloaded from the PC.

Once the download starts, every character in the S record file is immediately echoed back to the PC. This ensures synchronism between the PC and the MC68HC11, and at the same time, removes some of the overhead associated with the EEPROM programming delay time. It also removes the need for a hardware handshake.

VERIFY OPTION

If a verify error occurs, the actual stored byte value is returned to the PC, where it is displayed with a preceding colon delimiter. In this way, EEPROM data and address faults can be quickly identified by inspection. At the end of the verify download, the total number of errors is displayed.

INTERNAL OR EXTERNAL OPTION

If a programming error occurs in either internal or external programming mode, i.e., if the read back data after programming does not correspond to the expected data, the MC68HC11-resident software will hang up. This condition is detected by the PC-resident program, which will then abort the download and display an error message. This same error message is displayed if a fault or incorrect connection exists on the serial link between the PC and MC68HC11.

There is one exception to this operation. It stems from the fact that changes to the MC68HC11's CONFIG register can only be detected after a subsequent hardware RESET. If the CONFIG register address (\$103F) is detected, then the CONFIG register is not read directly after programming. This prevents premature termination of the download.

To allow programming of the CONFIG register in all mask set versions of the MC68HC11A series, and to permit expanded mode operation, the MCU resident program switches from bootstrap mode to special test mode, by setting the MDA bit (bit 5) in the HPRI0 register (address \$103C).

If the user wishes to maintain operation in bootstrap mode, (to verify internal ROM code, for instance), then the 'BSET HPRI0,X,#MDA' instruction on the 8th line of program code in EEPROG.ASC should be removed, and the program reassembled.

PROGRAMMING INTERNAL EEPROM

The techniques for programming internal and external EEPROM are quite different.

With internal EEPROM, it is first generally necessary to erase the required byte (erased state is \$FF), and follow with a write of data to the same address.

The internal programming sequence involves accessing the PPROG register (address \$103B) to latch the EEPROM address and data buses for the duration that the

programming voltage is applied. Also, the programming time delay must be implemented or initiated by software. In this application, a software timing loop is used, but one of the internal MC68HC11 timer functions could equally well be used to provide the time delay.

Figures 2 and 3 show the flowcharts of the internal EEPROM erase and write sequences.

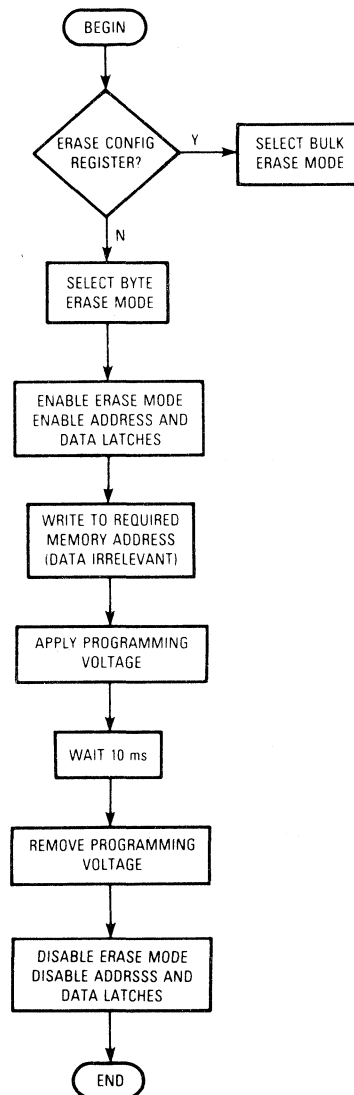


Figure 2. Internal EEPROM Erase Sequence

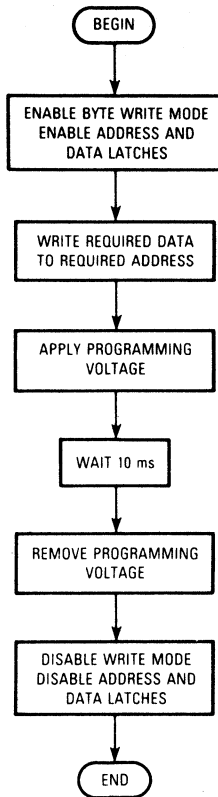


Figure 3. Internal EEPROM Write Sequence

PROGRAMMING EXTERNAL EEPROM

Figure 4 shows the hardware needed to interface the MC68HC11 to an external 2864 EEPROM, which provides a total of 8K bytes of reprogrammable memory. The ad-

dition of the MC68HC24 gives a minimal component count implementation of a circuit which accurately emulates the MC68HC11A8 single-chip MCU. The added benefit of using the 2864 is that the software designer's program and/or data can be modified without removing the emulator from the target system. This can be particularly useful in applications where the emulator may be enclosed in a confined space or in an environmental chamber.

To program the 2864 from the PC, the external operating mode option (X) must be selected from the EELoad menu.

Programming the 2864 involves fewer operations than are needed for internal EEPROM, as the former has no equivalent of the PPROG control register. In addition, the erase sequence and delay time are handled automatically by the 2864 on-chip logic.

A data polling technique is used to determine the end of the programming delay time. This involves examining the most significant bit of the data programmed, by reading from the address just written to, until the data becomes true. (During the programming delay time, the MS bit will read as the complement of the expected data).

This means that the same software algorithm can be used to download code or data to external RAM as well as external EEPROM.

EMULATOR ADDRESS DECODING

The emulator circuit in Figure 4 shows the MC68HC11's address line A13 connected to pin 26 of the 2864. Though this pin is actually unused by the 2864, its inclusion permits the replacement of the 2864 with a 27128 16K byte EEPROM memory.

An important outcome of this is that, when a 2864 is used, the memory range \$C000-\$DFFF is mapped over the normally used 8K byte range of \$E000-\$FFFF. In practice, this should never pose a problem. When a 27128 memory is used, its full 16K byte address range of \$C000-\$FFFF is available to the MCU.

Included in the S record programmer, irrespective of the selected programming mode, is a feature to force program execution at the address specified in the S9, S record address field, provided the address is not \$0000.

Figure 5 shows the general format of S record files.

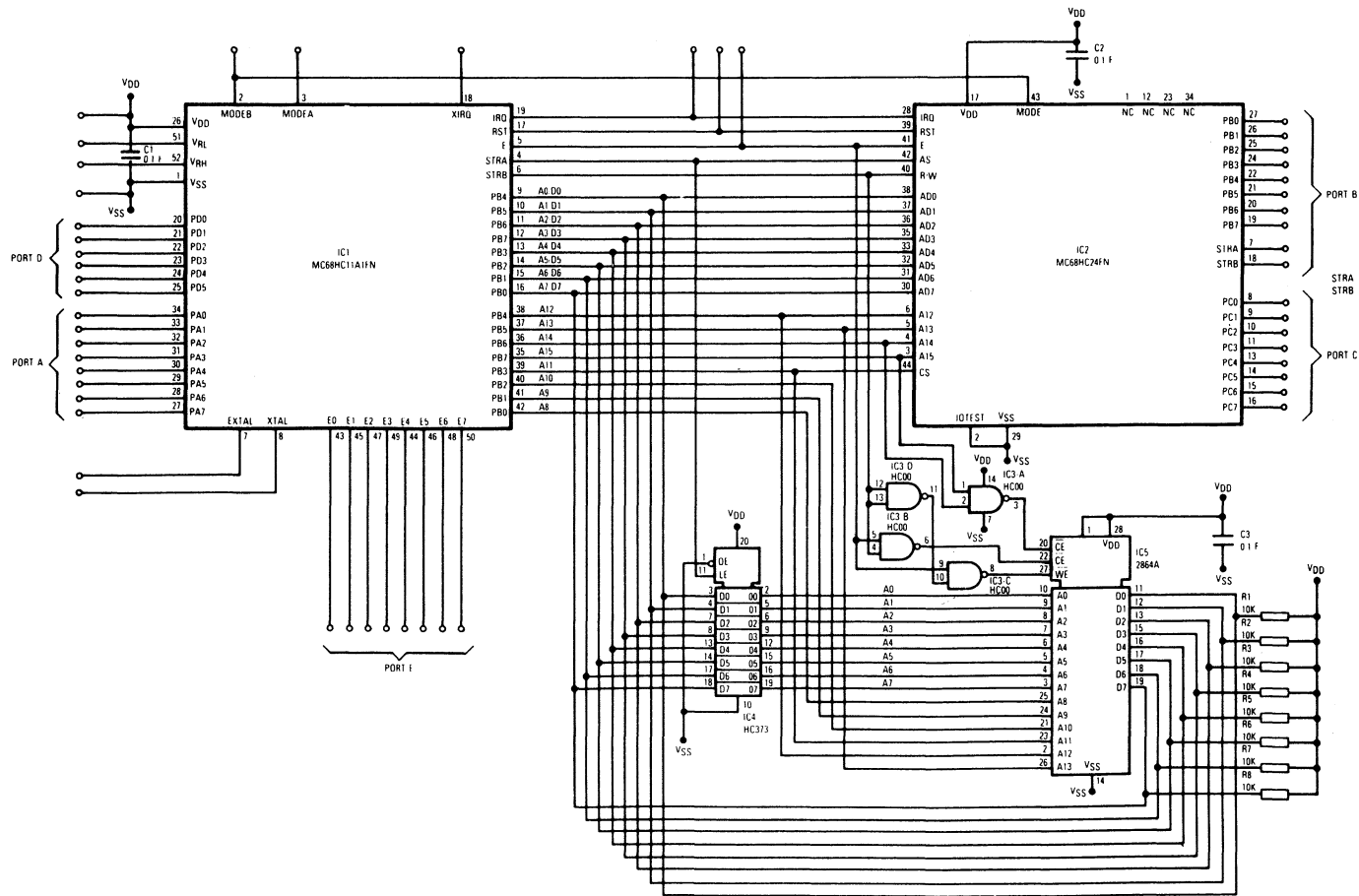
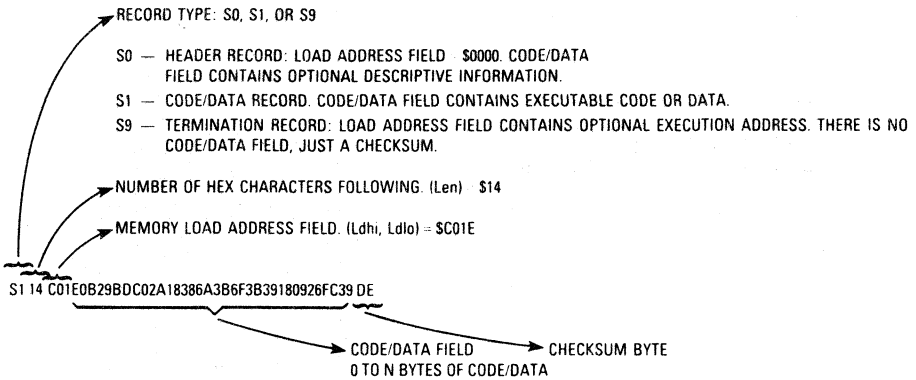


Figure 4. MC68HC11A8 Emulator Using 2864 EEPROM



APART FROM THE LETTER S AT THE START, ALL CHARACTERS IN THE RECORDS ARE HEXADECIMAL DIGITS REPRESENTED IN ASCII FORMAT.

CHECKSUM ALGORITHM: LSB OF
$$\left[\text{Len} \cdot \text{Ldhi} + \text{Ldlo} \cdot \sum_{k=0}^n \text{byte}_k \right]$$

NOTE: The S-record programmer in this application ignores the checksum byte.

Figure 5. S-Record Format

```

10 ' ***** ELOAD.BAS 20/3/87  Version 1.0 *****'
20 ' Written by R.Soja, Motorola East Kilbride'
30 ' Motorola Copyright 1987'
40 ' This program downloads S record file to the MC68HC11 through special'
50 ' bootstrap program, designed to program either internal or external '
60 ' EEPROM in the 68HC11's memory map'
70 ' The loader can also verify memory against an S record file.'
80 ' Downloaded data is optionally echoed on terminal.'
90 ' =====
100 CRS=CHR$(13)
110 MINS=CHR$(32)
120 MAX$=CHR$(127)
130 ERMS="Can't find "
140 LOADERS$="EEPROM.GIB.BOO"
150 CLRLN$=SPACE$(80)
160 VERS$="1.0": 'Version number of ELOAD'
170 ERRTOT%=0: 'Number of errors found by verify operation'
180 CLS
190 PRINT " <<<<<<<          ELOAD Version ";VER$;"          >>>>>>>"
200 PRINT " <<<<<<< 68HC11 Internal/External EEPROM loader/verifier >>>>>>>"
210 PRINT
220 PRINT "==> Before continuing, ensure 68HC11 is in bootstrap mode,"
230 PRINT "      RESET is off, and COM1 or COM2 is connected to the SCI"
240 PRINT
250 ' First make sure loader program is available'
260 ON ERROR GOTO 880
270 OPEN LOADERS$ FOR INPUT AS #2
280 CLOSE #2
290 ON ERROR GOTO 0
300 CHAN$=""
310 ROW=CSRLIN: 'Store current line number'
320 WHILE CHAN$<>"1" AND CHAN$<>"2"
330   GOSUB 1070
340   LINE INPUT "Enter COM channel number (1/2):",CHAN$
350 WEND
360 CMS$="COM"+CHAN$
370 ' Now set baud rate to 1200 and load EEPROM through boot loader'
380 ' by executing DOS MODE and COPY commands'
390 SHELL "MODE "+CMS$+":1200,N,8,1"
400 SHELL "COPY "+LOADERS$+" "+CMS$
401 GOSUB 1070
402 FOR I%=1 TO 4:PRINT CLRLN$;NEXT I%:PRINT: 'Clear DOS commands from screen'
410 ECHO$=""
420 WHILE ECHO$<>"Y" AND ECHO$<>"N"
430   GOSUB 1070
440   LINE INPUT "Do you want echo to screen (Y/N):",ECHO$
450 WEND
470 ROW=CSRLIN: 'Store current line number'
480 EEOPT$="" ': 'Initialise option char'
490 WHILE EEOPT$<>"X" AND EEOPT$<>"I" AND EEOPT$<>"V"
500   GOSUB 1070
510   LINE INPUT "Select Internal,eXternal or Verify EEPROM option (I/X/V):",EEOPT$
520 WEND
530 OPT$="Verify"
540 IF EEOPT$="I" THEN OPT$="Internal"
550 IF EEOPT$="X" THEN OPT$="External"
560 ROW=CSRLIN: 'Store current line position in case of file error'
570 RXERR=0: 'Initialise number of RX errors allowed'
580 ON ERROR GOTO 910
590 GOSUB 1070

```



```

600 IF OPT$="Verify" THEN INPUT "Enter filename to verify: ",F$ ELSE INPUT "Enter filename to download:",F$
610 CLOSE
620 OPEN F$ FOR INPUT AS #2
630 ON ERROR GOTO 0
640 'COM1 or 2 connected to SCI on HC11'
650 OPEN CMS+":9600,N,8,1" AS #1
660 'Establish contact with HC11 by sending CR char & waiting for echo'
670 ON ERROR GOTO 860: 'Clear potential RX error'
680 PRINT #1,CR$;
690 GOSUB 990: 'Read char into B$'
700 'Transmit Internal,External or Verify EEPROM option char to 68HC11'
710 PRINT #1,EEOPT$;:GOSUB 990: 'No echo to screen'
720 ON ERROR GOTO 930
730 PRINT "Starting download of <";F$;"> to: ";OPT$;" Eeprom"
732 IF ECHO$="Y" THEN E%=1 ELSE E%=0
734 IF EEOPT$="V" THEN V%=1 ELSE V%=0
740 WHILE NOT EOF(2)
750 INPUT #2,S$
751 L%=LEN(S$)
752 FOR I%=1 TO L%
760 PRINT #1,MID$(S$,I%,1);:GOSUB 990:IF E% THEN PRINT B$;
770 IF V% THEN GOSUB 1030:IF C$<>" THEN PRINT ":",HEX$(ASC(C$));
785 NEXT I%
787 IF E% THEN PRINT
790 WEND
795 PRINT
800 PRINT "Download Complete"
810 IF V% THEN PRINT ERRTOT%;" error(s) found"
820 CLOSE #2
830 SYSTEM
840 END
850 '-----'
860 IF RXERR>5 THEN 940 ELSE RXERR=RXERR+1:RESUME 610
870 '-----'
880 PRINT:PRINT ERMS$;LOADER$:PRINT "Program aborted"
890 GOTO 830
900 '-----'
910 PRINT ERMS$;F$;SPACES(40)
920 RESUME 580
930 '-----'
940 PRINT:PRINT "Communication breakdown: Download aborted"
950 GOTO 820
960 '-----'
970 '--SUB waits for received character, with time limit'
980 '-- returns with char in B$, or aborts if time limit exceeded'
990 TOX=0:WHILE LOC(1)=0:IF TOX>100 THEN 940 ELSE TOX=TOX+1:WEND
1000 B$=INPUT$(1,#1):RETURN
1010 '-----'
1020 '--SUB waits for received character, with time limit'
1025 '-- returns with char in C$, or null in C$ if time limit exceeded'
1030 TOX=0:C$="":WHILE LOC(1)=0 AND TOX<1:TOX=TOX+1:WEND
1040 IF LOC(1)>0 THEN C$=INPUT$(1,#1):ERRTOT%=ERRTOT%+1
1050 RETURN
1060 '-----'
1070 '--SUB Clear line '
1080 LOCATE ROW,1,1:PRINT CLRLNS
1090 LOCATE ROW,1,1:RETURN
1100 '-----'

```

```

1 A      *****
2 A      *          EEPROMIX.ASC 19/3/87   Revision 1.0      *
3 A      *
4 A      * Written by R.Soja, Motorola, East Kilbride        *
5 A      * Motorola Copyright 1987.                          *
6 A      *
7 A      *          This program loads S records from the host to
8 A      * either a 2864 external EEPROM on the 68HC11 external bus, *
9 A      * or to the 68HC11's internal EEPROM. It can also be used *
10 A     * verify memory contents against an S record file or just *
11 A     * load RAM located on the 68HC11's external bus.      *
12 A     * Each byte loaded is echoed back to the host.      *
13 A     * When programming a 2864, data polling is used to detect *
14 A     * completion of the programming cycle.                *
15 A     * As the host software always waits for the echo before *
16 A     * downloading the next byte, host transmission is suspended *
17 A     * during the data polling period.                      *
18 A     * Because the serial communication rate (~1mS/byte) is *
19 A     * slower than the 2864 internal timer timeout rate (~300uS) *
20 A     * page write mode cannot be used. This means that data *
21 A     * polling is active on each byte written to the EEPROM, *
22 A     * after an initial delay of approx 500uS.              *
23 A     *
24 A     * When the internal EEPROM is programmed, instead of data *
25 A     * polling, each byte is verified after programming.   *
26 A     * In this case, the 500uS delay is not required and is *
27 A     * bypassed.
28 A     * If a failure occurs, the program effectively hangs up. It *
29 A     * is the responsibility of the host downloader program to *
30 A     * detect this condition and take remedial action.     *
31 A     * The BASIC program EELOAD just displays a 'Communication *
32 A     * breakdown' message, and terminates the program.     *
33 A     *
34 A     * When used in the verify mode, apart from the normal echo *
35 A     * back of each character, all differences between memory *
36 A     * and S record data are also sent back to the host.   *
37 A     * The host software must be capable of detecting this, and *
38 A     * perform the action required.
39 A     * The BASIC loader program EELOAD simply displays the *
40 A     * returned erroneous byte adjacent to the expected byte, *
41 A     * separated by a colon.
42 A     *
43 A     * Before receiving the S records, a code byte is received *
44 A     * from the host. i.e.:
45 A     *          ASCII 'X' for external EEPROM
46 A     *          ASCII 'I' for internal EEPROM
47 A     *          ASCII 'V' for verify EEPROM
48 A     *
49 A     * This program is designed to be used with the BASIC EELOAD *
50 A     * program.
51 A     * Data transfer is through the SCI, configured for 8 data *
52 A     * bits, 9600 baud.
53 A     *
54 A     PAGE

```

```

55 A
56 A 0080 TDRE EQU $80
57 A 0020 RDRF EQU $20
58 A 0020 MDA EQU $20
59 A 0040 SMOD EQU $40
60 A 0D05 mS10 EQU 10000/3 10ms delay with 8MHz xtal.
61 A 00A6 uS500 EQU 500/3 500us delay.
62 A
63 A
64 A 002B BAUD EQU $2B
65 A 002C SCCR1 EQU $2C
66 A 002D SCCR2 EQU $2D
67 A 002E SCSR EQU $2E
68 A 002F SCDR EQU $2F
69 A 003B PPRG EQU $3B
70 A 003C HPR10 EQU $3C
71 A 103F CONFIG EQU $103F
72 A
73 A
74 A 0000 * Variables. Note: They overwrite initialisation code!!!!
75 P 0000 0001 ORG $0
76 P 0001 0001 EEOPT RMB 1
77 P 0002 0001 MASK RMB 1
78 P 0003 0001 TEMP RMB 1
79 A
80 A
81 A 0000 * Program
82 A 0000 8E00FF ORG $0
83 A 0003 CE1000 LDS #$FF
84 A 0006 6F2C LDX #$1000 Offset for control registers.
85 A 0008 CC300C CLR SCCR1,X Initialise SCI for 8 data bits, 9600 baud
86 A 000B A728 LDD #$300C
87 A 000D E72D STAA BAUD,X
88 A 000F 1C3C20 STAB SCCR2,X
89 A BSET HPR10,X,#MDA Force Special Test mode first,
90 A * BCLR HPR10,X,#SMOD and then expanded mode. (From Bootstrap mode)
91 A 0012 9F00 ReadOpt STS <EEOPT Default to internal EEPROM: EEOPT=0; MASK=$FF;
92 A 0014 8D7C BSR READC Then check control byte for external or internal
93 A 0016 C149 CMPB #'1' EEPROM selection.
94 A 0018 2714 BEQ LOAD
95 A 001A C158 CMPB #'X' If external EEPROM requested
96 A 001C 2609 BNE OptVerf
97 A 001E 7C0000 INC EEOPT then change option to 1
98 A 0021 8680 LDAA #$80
99 A 0023 9701 STAA <MASK and select mask for data polling mode.
100 A 0025 2007 BRA LOAD
101 A
102 A 0027 C156 OptVerf CMPB #'V' If not verify then
103 A 0029 26E7 BNE ReadOpt get next character else
104 A 002B 7A0000 DEC EEOPT make EEOPT flag negative.
105 A
106 A 002E LOAD EQU *
107 A 002E 8D62 BSR READC
108 A 0030 C153 CMPB #'S' Wait until S1 or S9 received,
109 A 0032 26FA BNE LOAD discarding checksum of previous S1 record.
110 A 0034 8D5C BSR READC
111 A 0036 C131 CMPB #'1'
112 A 0038 2719 BEQ LOAD1

```

113 A 003A C139		CMPB	#'9	
114 A 003C 26F0		BNE	LOAD	
115 A 003E 8D5F		BSR	RDBYTE	Complete reading S9 record before terminating
116 A 0040 17		TBA		
117 A 0041 8002		SUBA	#2	# of bytes to read including checksum.
118 A 0043 8D6B		BSR	GETADR	Get execution address in Y
119 A 0045 8D58	LOAD9	BSR	RDBYTE	Now discard remaining bytes,
120 A 0047 4A		DECA		including checksum.
121 A 0048 26FB		BNE	LOAD9	
122 A 004A 188C0000		CPY	#0	If execution address =0 then
123 A 004E 27FE		BEQ	*	hang up else
124 A 0050 186E00		JMP	,Y	jump to it!
125 A	*			
126 A 0053	LOAD1	EQU	*	
127 A 0053 8D4A		BSR	RDBYTE	Read byte count of S1 record into ACCB
128 A 0055 17		TBA		and store in ACCA
129 A 0056 8003		SUBA	#3	Remove load address & checksum bytes from count
130 A 0058 8D56		BSR	GETADR	Get load address into X register.
131 A 005A 1809		DEY		Adjust it for first time thru' LOAD2 loop.
132 A 005C 2017		BRA	LOAD1B	
133 A	*			
134 A 005E D600	LOAD1A	LDAB	EEOPT	Update CC register
135 A 0060 2B25		BMI	VERIFY	If not verifying EEPROM then
136 A 0062 2705		BEQ	DATAPOLL	If programming external EEPROM
137 A 0064 C6A6		LDAB	#uS500	
138 A 0066 5A	WAIT1	DECB		then wait 500uS max.
139 A 0067 26FD		BNE	WAIT1	
140 A 0069 18E600	DATAPOLL	LDAB	,Y	Now either wait for completion of programming
141 A 006C D803		EORB	<LASTBYTE	cycle by testing MS bit of last data written to
142 A 006E D401		ANDB	<MASK	memory or just verify internal programmed data.
143 A 0070 26F7		BNE	DATAPOLL	
144 A 0072 4A	LOAD1E	DECA		When all bytes done,
145 A 0073 27B9		BEQ	LOAD	get next S record (discarding checksum) else
146 A 0075 8D28	LOAD1B	BSR	RDBYTE	read next data byte into ACCB.
147 A 0077 1808		INY		Advance to next load address
148 A 0079 7D0000		TST	EEOPT	
149 A 007C 2B05		BMI	LOAD1D	If verifying, then don't program byte!
150 A 007E 2743		BEQ	PROG	If internal EEPROM option selected then program
151 A 0080 18E700		STAB	,Y	else just store byte at address.
152 A 0083 D703	LOAD1D	STAB	<LASTBYTE	Save it for DATA POLLING operation.
153 A 0085 2007		BRA	LOAD1A	
154 A	*			
155 A 0087 18E600	VERIFY	LDAB	,Y	If programmed byte
156 A 008A D103		CMPB	<LASTBYTE	is correct then
157 A 008C 27E4		BEQ	LOAD1E	read next byte
158 A 008E 8D08		BSR	WRITEC	else send bad byte back to host
159 A 0090 20E0		BRA	LOAD1E	before reading next byte.
160 A	*			
161 A 0092	READC	EQU	*	ACCA, X, Y regs unchanged by this routine.
162 A 0092 1F2E20FC		BRCLR	SCSR,X,#RDRF,*	
163 A 0096 E62F		LDAB	SCDR,X	Read next char
164 A 0098 1F2E80FC	WRITEC	BRCLR	SCSR,X,#TDRE,*	
165 A 009C E72F		STAB	SCDR,X	and echo it back to host.
166 A 009E 39		RTS		Return with char in ACCB.
167 A	*			
168 A 009F 8DF1	RDBYTE	BSR	READC	1st read MS nibble
169 A 00A1 8D17		BSR	HEXBIN	Convert to binary
170 A 00A3 58		LSLB		and move to upper nibble

```

171 A 00A4 58          LSLB
172 A 00A5 58          LSLB
173 A 00A6 58          LSLB
174 A 00A7 D702        STAB      <TEMP
175 A 00A9 80E7        BSR      READC      Get ASCII char in ACCB
176 A 00AB 800D        BSR      HEXBIN
177 A 00AD DA02        ORAB     <TEMP
178 A 00AF 39          RTS      Return with byte in ACCB
179 A                  *
180 A      00B0        GETADR EQU      *
181 A 00B0 36          PSHA
182 A 00B1 80EC        BSR      RDBYTE     Save byte counter
183 A 00B3 17          TBA      Read MS byte of address
184 A 00B4 80E9        BSR      RDBYTE     and put it in MS byte of ACCD
185 A 00B6 188F        XGDY     Now read LS byte of address into LS byte of ACCD
186 A 00B8 32          PULA     Put load address in Y
187 A 00B9 39          RTS      Restore byte counter
188 A                  *
189 A      00BA        HEXBIN EQU      *
190 A 00BA C139        CMPB     #'9        If ACCB>9 then assume its A-F
191 A 00BC 2302        BLS     HEXNUM
192 A 00BE C809        ADDB     #9
193 A 00C0 C40F        HEXNUM  ANDB     #$F
194 A 00C2 39          RTS
195 A                  *
196 A      00C3        PROG  EQU      *
197 A 00C3 36          PSHA     Save ACCA.
198 A 00C4 8616        LDAA     #$16      Default to byte erase mode
199 A 00C6 188C103F    CPY     #CONFIG    If byte's address is CONFIG then use
200 A 00CA 2602        BNE     PROGA
201 A 00CC 8606        LDAA     #$06      bulk erase, to allow for A1 & A8 as well as A2.
202 A 00CE 8010        PROGA   BSR      PROGRAM Now erase byte, or entire memory + CONFIG.
203 A 00D0 8602        LDAA     #2
204 A 00D2 800C        BSR      PROGRAM    Now program byte.
205 A 00D4 188C103F    CPY     #CONFIG    If byte was CONFIG register then
206 A 00D8 2603        BNE     PROGX
207 A 00DA 18E600      LDAB     ,Y        load ACCB with old value, to prevent hangup later.
208 A 00DD 32          PROGX   PULA     Restore ACCA
209 A 00DE 20A3        BRA     LOAD1D     and return to main bit.
210 A                  *
211 A      00E0        PROGRAM EQU      *
212 A 00E0 A73B        STAA    PPROG,X   Enable internal addr/data latches.
213 A 00E2 18E700      STAB    ,Y        Write to required address
214 A 00E5 6C3B        INC     PPROG,X   Enable internal programming voltage
215 A 00E7 3C          PSHX
216 A 00E8 CE0D05      LDX     #mS10     and wait 10mS
217 A 00EB 09          WAIT2   DEX
218 A 00EC 26FD        BNE     WAIT2
219 A 00EE 38          PULX
220 A 00EF 6A3B        DEC     PPROG,X   Disable internal programming voltage
221 A 00F1 6F3B        CLR     PPROG,X   Release internal addr/data latches
222 A 00F3 39          RTS      and return
223 A                  *
224 A                  END

```

SYMBOL TABLE: Total Entries= 41

BAUD	002B	PROGA	00CE
CONFIG	103F	PROGRAM	00E0
DATAPOLL	0069	PROGX	00D0
EEOPT	0000	RDBYTE	009F
GETADR	0080	RDRF	0020
HEXBIN	008A	READC	0092
HEXNUM	00C0	ReadOpt	0012
HPRIO	003C	SCCR1	002C
LASTBYTE	0003	SCCR2	002D
LOAD	002E	SCDR	002F
LOAD1	0053	SCSR	002E
LOAD1A	005E	SMOD	0040
LOAD1B	0075	TDRE	0080
LOAD1D	0083	TEMP	0002
LOAD1E	0072	VERIFY	0087
LOAD9	0045	WAIT1	0066
MASK	0001	WAIT2	00EB
MDA	0020	WRITEC	0098
OptVerf	0027	mS10	0005
PPROG	003B	us500	00A6
PROG	00C3		

DESIGNING FOR ELECTROMAGNETIC COMPATIBILITY (EMC) WITH HCMOS MICROCONTROLLERS

by
Mike Catherwood
Motorola Inc.
Austin, Tx.

INTRODUCTION

The operating speed of present HCMOS devices is approaching that of the fastest bipolar logic families of only a few years ago. Associated with this increase in performance are some new design challenges for the MCU-based system designer. This application note addresses one of these issues, the electromagnetic compatibility (EMC) of the finished product. EMC may be considered from either an emission or a susceptibility point of view. Although the following discussion relates primarily to emission control (in particular, radiated emission), most techniques to limit emission also reduce susceptibility. Furthermore, minimizing electromagnetic interference (EMI) will reduce overall system noise, the benefits of which are higher digital noise immunity and accurate operation of local analog subsystems — i.e., better design margin and a more reliable end product.

EMC can only exist when the system functions correctly within the intended electromagnetic environment and does not exceed the EMI levels specified in the appropriate standards documents. EMI, which encompasses interference in a bandwidth of 'dc to daylight' is a generalization of a much older term, radio frequency interference (RFI), which is now defined to encompass 10 kHz to 3 GHz. Failure to consider EMC during early phases of the design process may result in expensive modifications (possibly with many additional components), printed circuit board (PCB) re-layout, product introduction delays, and EMC consultant fees to conform to the required standards.

LEGAL REQUIREMENTS

The Federal Communications Commission (FCC) have a set of standards to regulate EMI in electronic equipment and systems for use in the United States. Compliance with the appropriate sections of these regulations is mandatory to market or sell a product except for certain subclasses of digital devices that are temporarily exempt. Engineering models (including field-trial prototypes which are not sold) are also exempt; however, the display of a product at an electronics show is considered a marketing function subject to regulations.

FCC rules and regulations (part 15, subpart J of Title 47 of The Federal Regulations) apply to almost all digital devices (see Reference 1), defining standards and operational requirements for all devices capable of emitting RF energy within the range 450 kHz to 1 GHz.

Equipment for use within West Germany must comply with a different set of standards defined and administered by the Verband Deutscher Electro-Techniker (VDE). Digital equipment is generally required to meet both VDE0871 standards. In other countries, compliance to a standard is not always mandatory, however, the European Economic Community (EEC) member states intend to introduce a mandatory RFI performance standard after 1st January 1992. The current proposal is based on International Special Committee on Radio Interference specification CISPR22 and is referred to as European norm EN55022. As the FCC is a member of the CISPR, and has voted in favor of the CISPR22 standard, it is likely that the FCC will ultimately adopt the same standards. CISPR22 is somewhat more stringent than FCC part 15, subpart J in the 88 to 230 MHz frequency range, though it is less stringent than some aspects of the VDE0871.

RFI PROBLEM OVERVIEW

The frequency spectrum of a periodic waveform has been shown, through Fourier analysis, to be composed of discrete frequency components that include the fundamental (f_0) and multiple harmonics ($n \times f_0$). For a typical trapezoidal waveform, the relative amplitude of each frequency component is related to the fundamental frequency, the rise time, and mark-to-space ratio (duty cycle) of the waveform (see Reference 2). Doubling the frequency, halving the rise time, or halving the mark-to-space ratio will double (+ 6 dB) the amplitude of a specific harmonic frequency.

It is possible to graphically predict the harmonic spectrum of a specific trapezoidal waveform by plotting the amplitude of two corner frequencies and a reference (0 dB) point. This plot is referred to as a Fourier envelope, a Bode plot, or a nomogram. An example is shown in Figure 1 where:

$$\begin{aligned} 0 \text{ dB reference} &= 20 \log(2A\delta) && \text{dB} \\ f_1 &= 1/\pi P && \text{Hz} \\ f_2 &= 1/\pi t_r && \text{Hz} \end{aligned}$$

where:

$$\begin{aligned} V &= \text{amplitude} && \text{V} \\ P &= \text{pulse width} && \text{s} \\ t_r &= \text{rise time} && \text{s} \\ T &= \text{period} && \text{s} \\ \delta &= (P - t_r) T \end{aligned}$$

At frequencies beyond f_1 ($1/\pi P$), the amplitude of the harmonics falls off at -20 dB/decade. Above f_2 ($1/\pi t_r$), the amplitude of the harmonics falls off at -40 dB/decade. For many applications, these latter harmonics are considered small enough to be ignored; thus, the bandwidth of a system is generally defined to be $1/\pi t_r$. For example, an HCMOS device which can produce an external periodic signal with edge times on the order of 2 ns can generate significant harmonics (i.e., have a bandwidth) of up to 160 MHz. Any PCB tracks, component leads, cables, or connectors attached directly or capacitively to signal sources, such as those previously described, can act as antennas and radiate the harmonics with varying degrees of efficiency. Radiated emission from a system may be either differential-mode or common-mode radiation; common-mode radiation is typically more difficult to reduce.

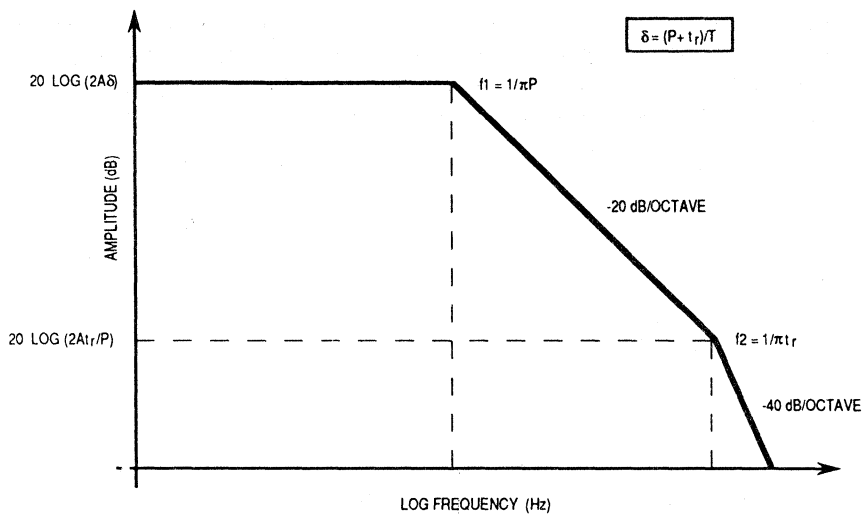
DIFFERENTIAL-MODE RADIATION

Differential-mode radiation is caused by the flow of RF current loops around the system conductors. For a small loop area, the far-field electric term, when operating in an field above a ground plane (free space is not a typical environment), can be shown to be approximately (see Reference 3):

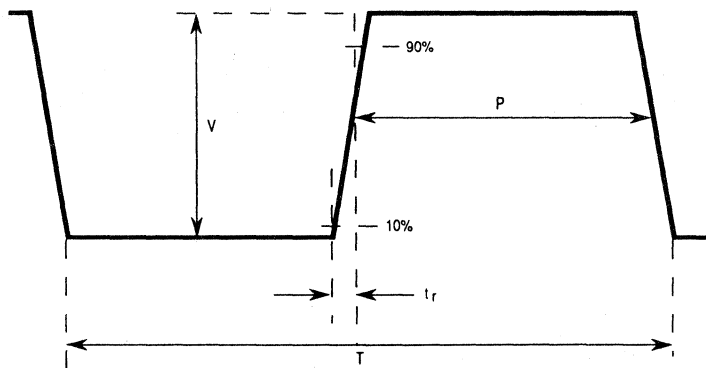
$$E = 2.6(A I_L f^2)/R \quad \mu\text{V/m} \quad (1)$$

where:

$$\begin{aligned} A &= \text{loop area} && \text{cm}^2 \\ I_L &= \text{loop current} && \text{A} \\ f &= \text{frequency} && \text{MHz} \\ R &= \text{distance} && \text{m} \end{aligned}$$



(a) Nomogram



(b) Trapezoidal Waveform

Figure 1. Nomogram of a Trapezoidal Waveform

For a constant current and loop area, the electric field at a prescribed distance is proportional to the square of the frequency (i.e., it increases at 40 dB/decade). Adding this term to the Fourier envelope indicates that the differential-mode radiated emission increases at 20 dB/decade up to f_2 , after which it remains flat. R is fixed by both the FCC and VDE rules and regulations, and f is usually not a system variable; however, A and L can be reduced through thoughtful board layout and careful circuit design.

COMMON-MODE RADIATION

Common-mode (CM) radiation is caused by unintentional voltage drops in a circuit, which cause some grounded parts of the circuit to rise above the real ground potential (see Figure 2). Cables connected to the affected ground act like antennas and radiate the components of the CM potential. The far-field electric term can be shown as follows (see Reference 3):

$$E \approx (f I_{CM} L)/R \quad \text{V/m} \quad (2)$$

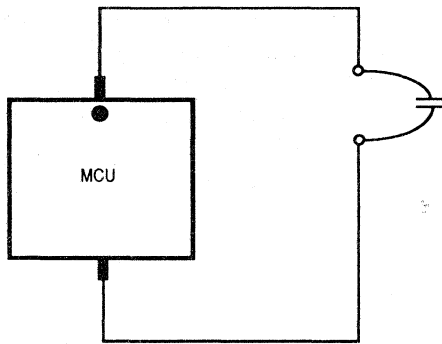
where:

L = antenna length	m
I_{CM} = common-mode current	A
f = frequency	Hz
R = distance	m

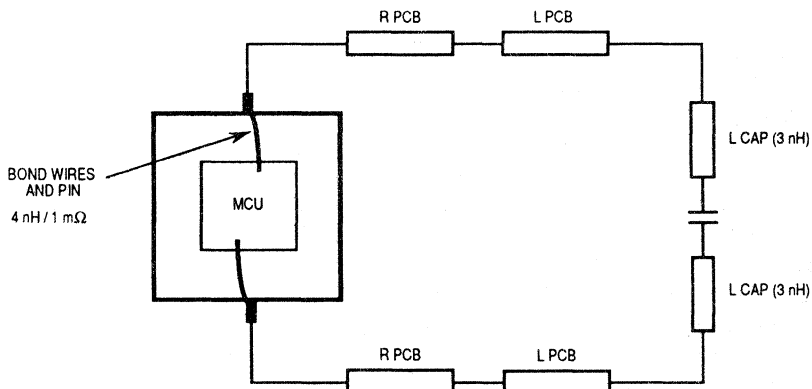
For a constant current and antenna length, the electric field at a prescribed distance is proportional to the frequency (i.e., it increases at 20 dB/decade). Adding this term to the Fourier envelope indicates that the CM radiated emission remains flat up to f_2 , then decreases at -20 dB/decade for frequencies above f_2 . Unlike differential-mode radiation, which is relatively easy to reduce through careful product design, CM radiation is more difficult to control since the only variables available to the designer are typically the common path impedances and CM current. Obviously, to eliminate the radiation, the CM current must approach zero, which can be achieved through a sensible grounding scheme and the addition of inductors or capacitors to increase the cable (antenna) impedance.

SUPPLY DECOUPLING

Inadequate decoupling decreases system noise margins and ultimately leads to incorrect, unreliable, or unstable operation. For example, the MC68HC11A8 can generate peak supply-current transients of approximately 100 mA, which is typical of an HCMOS microcontroller. Although the average supply current is only a few milliamps, the power supply must be able to source the peak supply-current levels



(a) Poor Supply Decoupling



FOR TOTAL PCB TRACK LENGTH OF 15 cm : L = 89 nH, R = 77 mΩ
 (ASSUMING 5 nH/cm AND 5 mΩ/cm)

(b) Equivalent Circuit of (a)

Figure 2. PCB Layout

to guarantee correct operation. Also, for fast digital logic, the peak supply-current transients are large enough to create an EMI problem if the decoupling layout is poor.

A decoupling network is used to reduce the supply impedance at the device. To calculate the value of a decoupling capacitor, the acceptable supply ripple must first be determined. An appropriate goal is to achieve a maximum ripple of 20% of the minimum noise immunity voltage — e.g., for the MC68HC11A8 with $V_{DD} = 5\text{ V}$ and no loads:

$$\begin{aligned}V_{IL} - V_{OL} &= (0.2 \times V_{DD}) - 0.1 = 0.9\text{ V} \\V_{OH} - V_{IH} &= 4.9 - (0.7 \times V_{DD}) = 1.4\text{ V}\end{aligned}$$

Therefore,

$$\begin{aligned}\text{Minimum noise immunity} &= 0.9\text{ V} \\ \text{Maximum ripple} &= 0.2 \times 0.9 = 180\text{ mV} \\ \text{Transient period} &= 10\text{ ns}\end{aligned}$$

Now,

$$\begin{aligned}C &= I_{DD} / (dv/dt) \\ &= 100\text{ mA} / (180\text{ mV} / 10\text{ ns}) \\ &= 0.006\text{ }\mu\text{F}\end{aligned}$$

Rounding up to the nearest preferred value gives $0.01\text{ }\mu\text{F}$. When operating the device in expanded mode, the transient currents generated by bus switching can be significantly larger. Consequently, the recommended decoupling configuration is a $1\text{ }\mu\text{F}$ tantalum in parallel with a high-frequency $0.01\text{ }\mu\text{F}$ multilayer ceramic (or similar) capacitor. The parallel $0.01\text{ }\mu\text{F}$ capacitor extends the upper frequency response of the network which might otherwise be reduced due to the internal inductance of the $1\text{ }\mu\text{F}$ capacitor. However, with the exception of VLSI devices, decoupling capacitors rarely need to exceed $0.01\text{ }\mu\text{F}$ per device. It is also recommended to bulk decouple the board at the supply-line entry point with a $10\text{--}100\text{ }\mu\text{F}$ capacitor, depending upon the total board-supply requirements. Because it is desirable to prevent unwanted supply noise from going off-board and radiating from the connecting cables, a ferrite bead can be added between the decoupling capacitor and the connector. Care must be taken to ensure that the DC current will not saturate the ferrite, making it ineffective.

For a decoupling network to operate successfully, the impedance between the network and its load must be very low, and, to reduce EMI, its loop area must be as small as possible. Consider the PCB layout of Figure 2(a); the equivalent circuit is shown in Figure 2(b).

For DC current,

$$\begin{aligned}V_{\text{drop}} &= (77) \times 0.1 \text{ mV} \\ &= 7.7 \text{ mV}\end{aligned}$$

For AC current, assuming 100 mA peak current with a minimum rise time of 10 ns,
Total inductance = 89 nH

$$\begin{aligned}V_{\text{drop}} &= L \text{ di/dt} \\ &= 89 \text{ nH}(100 \text{ mA}/10 \text{ ns}) \\ &= 890 \text{ mV}\end{aligned}$$

A drop of 0.9 V between the decoupling network and the MCU exceeds the maximum acceptable ripple, even if the recommended network is used. As shown in this example, for fast current transients containing many high-frequency components, the circuit inductance is by far the most critical factor when considering decoupling effectiveness.

Parasitic loop impedance can be effectively reduced through the use of thicker PCB tracks, ground/supply planes, and more direct routing. Decoupling networks should be located as close as possible to the device supply pins. Surface-mount capacitors, which have lower inductance than their leaded counterparts, may be used to the full advantage as decouplers if mounted on the noncomponent side of a PCB across a component, which is the closest possible location. Reducing loop impedance also tends to reduce loop area, which has been previously shown (see Equation (1)) to be directly proportional to radiated field strength.

SELF-RESONANCE

The inductance and capacitance within the decoupling loop, essentially results in a series-resonant tuned circuit where:

$$\text{resonant frequency, } f = 1 / 2\pi \sqrt{LC} \quad \text{Hz} \quad (3)$$

At frequencies above f , the impedance of the circuit becomes inductive and results in a less effective decoupler. At resonance, f , the impedance is purely resistive and at a minimum, which can be used to advantage in solving narrow-band RFI problems by tuning suspect decoupling networks to resonate at the problem frequency. For example, to reduce harmonics in the area of 100 MHz (the FM radio band) for a total loop inductance of 10 nH, equate Equation (3) to 100 MHz and solve for C . In this example, C would equal approximately 250 pF.

LINE TERMINATION

A signal will propagate down a PCB track at approximately 0.6 the speed of light (0.6 ft ns) until it reaches a load. If the line is unterminated (e.g., a high-impedance input), then the degree of impedance mismatch between the load and the line will cause a proportional amount of the signal to be reflected back down the line toward the source. These reflections can induce ringing and overshoot, causing significant EMI problems. If the load equals the characteristic impedance of the line, Z_0 , then from viewpoint of the line, the load looks like an infinite line and nothing will be reflected.

In the case of a mismatched line, if the source-signal rise time is sufficiently slow with respect to the line propagation time, then the reflections will be absorbed by the source during the signal rise time. In all other cases, the line should be treated as a transmission line and terminated accordingly (see Reference 3). As a general guide, there should be no need to terminate a line if the one-way propagation delay of a line is less than one-fourth of the signal rise time. For example, for HCMOS with a rise time of 10 ns, the maximum unterminated line length can be estimated as follows:

$$\begin{aligned}t \text{ delay} &< 0.25 \times 10 \text{ ns} \\ &< 2.5 \text{ ns}\end{aligned}$$

$$\begin{aligned}\text{length} &< \text{velocity} \times t \text{ delay} \\ &< 0.6 \times 2.5 \\ &< 1.5 \text{ ft}\end{aligned}$$

Therefore, for the majority of cases, termination will not be necessary when using HCMOS devices. Applying the same criteria to Schottky TTL, which has rise times on the order of 3 ns, provides a maximum length of 5.5 in.

FERRITE BEADS

Ferrite beads have excellent high-frequency characteristics and are especially effective in damping high-frequency switching transients or parasitic ringing due to line reflections. Their low impedance (usually below 100 Ω) makes them particularly suitable to filter out supply noise above approximately 1 MHz, preventing the noise from going off-board or into another circuit. However, care must be taken to ensure that the DC current does not saturate the ferrite if it is to be an effective filter. Ferrites having a variety of characteristics are available in many different packages, including surface mount.

GROUNDING TECHNIQUES

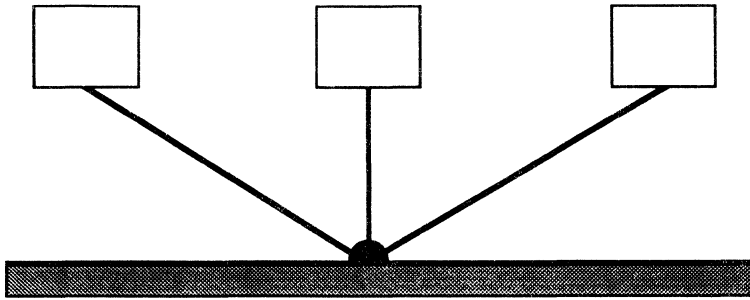
A ground is supposed to be an equipotential point or plane used as a reference potential within a system. In reality, this is untrue due to inevitable parasitic inductance and high ground currents causing significant voltage drops, which can result in common-mode radiation problems. To design a successful grounding scheme, the designer must be aware of the paths that ground currents will take to identify possible common-mode impedance problems, reduce loop areas, and prevent noisy return currents from interfering with low-level circuits.

Signal grounds can be classified as single-point, multipoint, or hybrid grounds (see Figure 3). Single-point is acceptable for low frequencies but may have too much impedance at higher frequencies to operate correctly. The ground wire length should be kept as short as possible to reduce inductance and radiating ability. A multipoint ground is used in high-frequency systems, such as digital circuitry, in which each element is connected to the nearest low-impedance ground plane. A hybrid ground looks like a single-point ground at low frequencies and a multipoint ground at high frequencies. A typical system is often a mixture of grounding techniques.

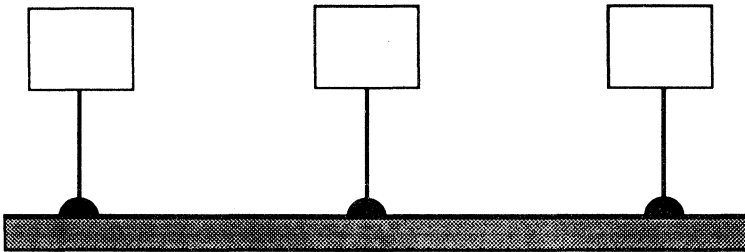
Figure 4 shows a typical MCU application grounding scheme, categorized into low-level analog, digital, input/output (I/O) buffer, high-current switching, and hardware grounds. A single-point ground is located at the source of primary power, which is typically the power supply. The on-board digital logic has a multipoint ground, though it is grounded off-board through a single-point ground. To prevent radiation, no high-frequency components of digital return current should be allowed off-board; thus, the board power-supply lines should only carry DC current, which is suitable for single-point grounding. A block diagram, such as the one shown in Figure 4, is a useful starting point for the design of a good grounding scheme.

ANALOG-DIGITAL MIX

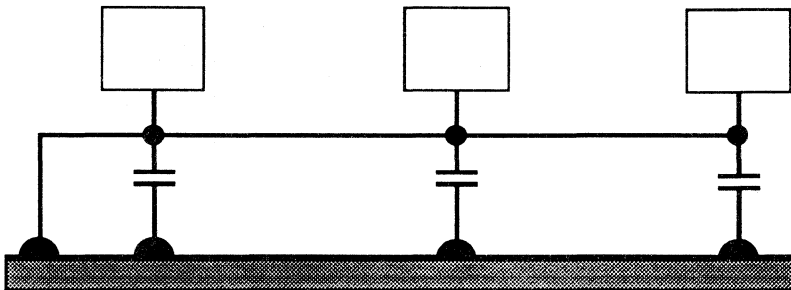
Combining analog and digital circuitry onto a single board requires special attention to PCB layout. Figure 5(a) demonstrates how common-mode impedance ground coupling can superimpose noise on an analog input signal. For example, if the analog section were a 12-bit A/D converter, the added digital noise would significantly reduce the achievable accuracy of the measurements, possibly by several bits. In Figure 5(a), the analog circuit shares its ground and supply with the noisy digital section and is therefore within the digital supply loop. The PCB tracks are also very thin, increasing the parasitic inductance and voltage drop. A better layout of the board is shown in Figure 5(b) in which the digital supply and ground tracks are substantially wider and the analog circuitry is provided with its own supply and ground reference. Any voltage drop occurring on the digital ground track no longer affects the analog input signal because the digital current no longer passes through the analog input loop.



(a) Single Point



(b) Multipoint



(c) Hybrid

Figure 3. Grounding Techniques

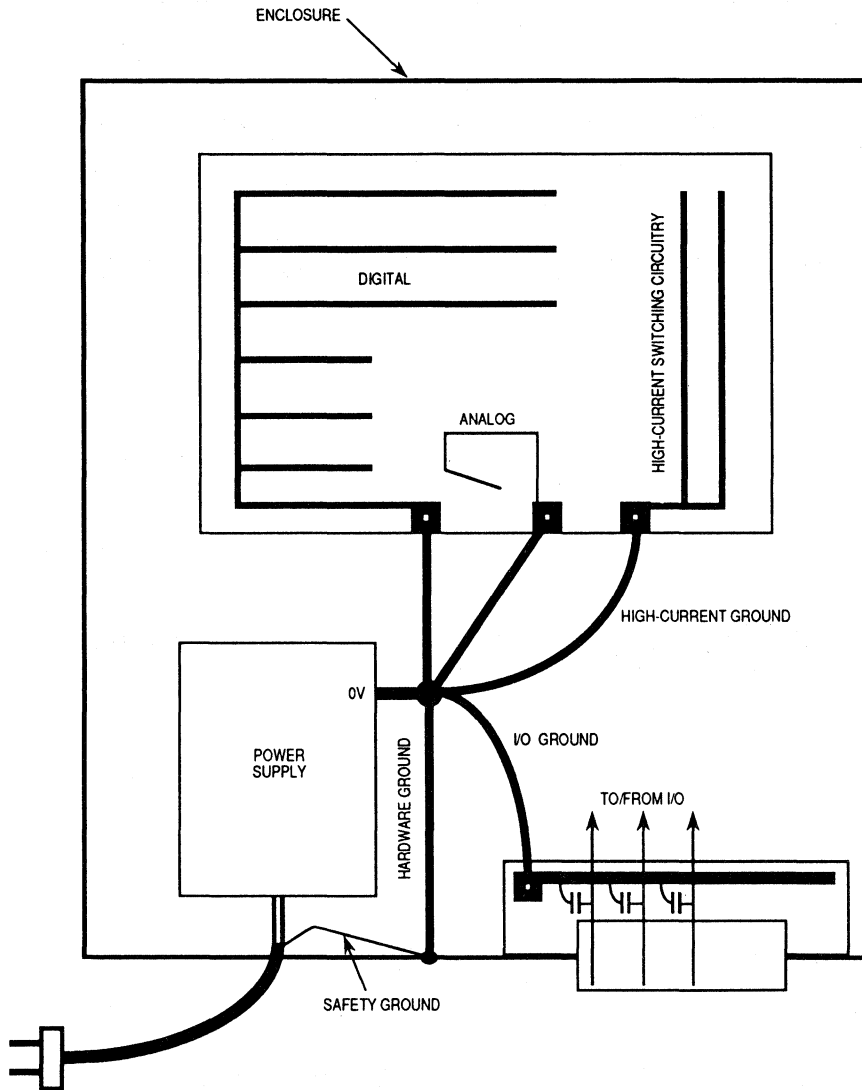
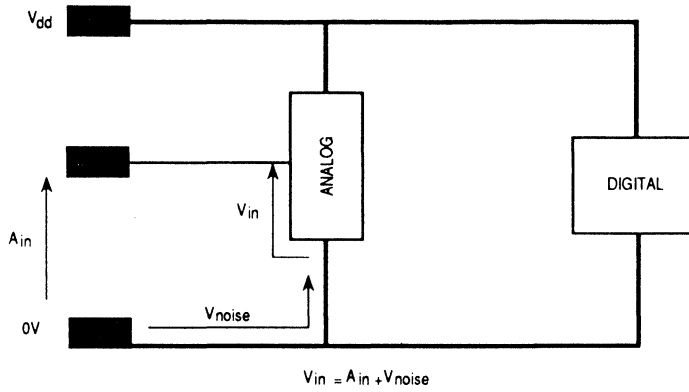
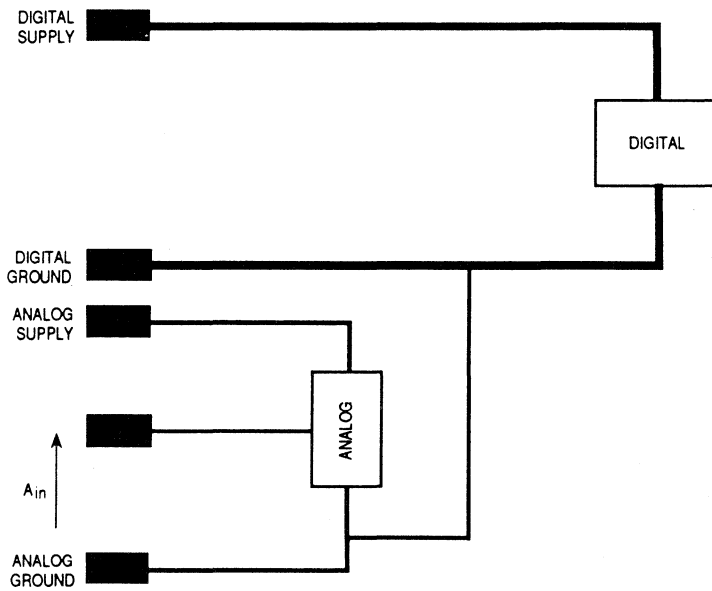


Figure 4. Typical MCU Application Grounding Example



(a) Poor Scheme



(b) Improved Scheme

Figure 5. Analog Circuit Grounding

Adequate supply decoupling is also a prerequisite to minimizing noise in an analog subsystem. With regard to the MC68HC11 on-chip A/D converter, the recommended decoupling network for the analog reference inputs is shown in Figure 6.

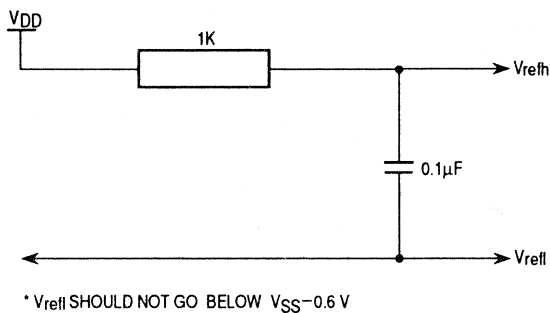


Figure 6. Recommended A/D Reference Voltage Decoupling for the MC68HC11

I/O CABLES AND SHIELDING

Providing a low-noise ground for I/O enables I/O shunt filters to be used to remove common-mode voltages from I/O cables that extend beyond the enclosure (see Figure 4). In addition, externally shielding I/O cables is ineffective if the termination grounds are themselves noisy. Alternatively, an inductor (choke) may also be used to increase I/O cable impedance and reduce radiation.

Generally, the shield surrounding low-frequency signals should be grounded at one end, and that for high-frequency signals, at both ends. For example, in Figure 4 where the analog section is grounded, an input cable shield would only be grounded at the analog circuitry end. Shielded cables carrying digital signals (e.g., MC68HC11 SCI data) should be grounded at both ends to ensure that the shield be as close as possible to ground potential throughout its length. If this configuration is not practical, the next best configuration is to ground only the signal source end of the shield. Caution, grounding at both ends of a long cable can cause large power-frequency ground-loop currents to flow due to potential differences between the shield grounding points. This problem can also be removed through filtering or the addition of a common-mode choke or balun (see Reference 3).

PCB LAYOUT GUIDELINES

A successful EMI design starts with good board design. As discussed earlier, the two criteria of most concern are signal-path inductance and loop area. The inductance of a flat conductor (e.g., a PCB track) above a current-return path is as follows:

$$L = 2 \ln(2 \pi h/w) \quad \text{nH/cm} \quad (4)$$

where:

h = height above current-return path

w = track width

Evaluating Equation (4) for a height above a current-return path of 1 mm and a track width of 0.5 mm, $L = 5.1$ nH/cm. The relationship is logarithmic, so doubling the track width will not halve the inductance, however, it will make a significant difference and is always worth doing. For example, doubling the track width to 1 mm makes $L = 3.7$ nH/cm. A track width of 5 mm makes $L = 0.5$ nH/cm, which is of the order required for effective decoupling loops and reducing common-mode radiation problems.

Use of a multilayer PCB will provide low-inductance supplies, though at an additional cost. The recommended arrangement is to place the supply and ground planes on the outside, sandwiching the signal lines between them. This arrangement will also provide some shielding. To minimize crosstalk, signals on adjacent layers should be routed perpendicular to each other wherever possible. If a multilayer board is not used, fill all unused area with ground plane; avoid creating ground loops that can cause EMI problems. For example, a ground loop is discovered and subsequently broken with a small gap. This technique is acceptable at DC, but at high frequencies the gap capacitance may effectively close the loop and create a large loop antenna. Apart from the radiation problems, large ground loops can also make a system more susceptible to malfunction when subjected to an electrostatic discharge (e.g., through a membrane keypad) or other external EMI source.

Reducing loop area through decoupling and careful layout will reduce RFI. The smaller footprint of surface-mount components can be used advantageously in reducing loop area. For PCBs without a ground plane, signal lines should ideally have a ground-return path as close as possible to them to minimize loop area. In the case of address/data lines, this arrangement may be impractical; thus, routing at least one ground-return track adjacent to each of the eight lines and keeping the lines as short as possible is a good compromise⁴. For the address lines, route the ground return next to A0 (in the case of a word-sized bus, A1), since this line is likely to be the most active. Ground and supply loops with long or thin tracks can

be easily identified by tracing them on a printed copy of the PCB artwork using colored marker pens. As previously mentioned, any unused area should be filled with ground plane.

The system clock is often a primary source of radiation. The clock components should be closely grouped, and all clock lines should be as short as possible and have adjacent ground tracks or ground plane. To avoid crosstalk contamination and subsequent radiation problems, the clock circuitry should be located away or shielded from any I/O signal lines or circuitry. For example, mixing clock and I/O buffers in one package is not good practice.

Another source of RFI is an abrupt change of direction of a PCB track which effectively look like impedance discontinuities and will radiate accordingly. For HCMOS designs, it is important to ensure that 90-degree track-direction changes do not occur (see Figure 7).

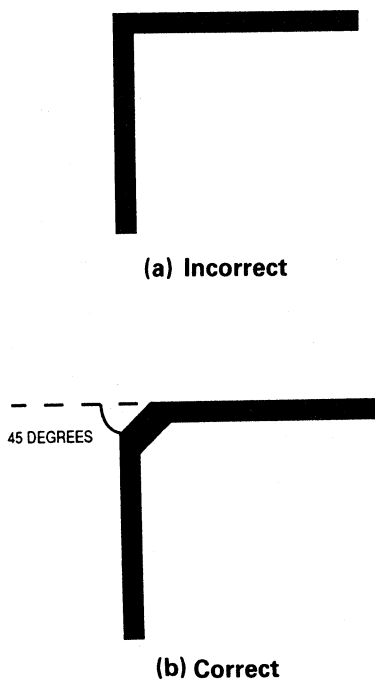


Figure 7. Incorrect (a) and Correct (b) PCB Track Layout for HCMOS Designs

Finally, all unused inputs to HCMOS devices should be terminated to prevent unintentional random switching and noise generation. Also, unterminated CMOS inputs tend to self-bias into the linear region of operation, which can significantly increase DC current drawn. They are also more susceptible to electrostatic discharge damage.

SIMPLE RFI DIAGNOSTIC TOOL

An article (reprinted by permission of *EMC Technology* magazine, Reference 5) detailing the construction of a set of RFI diagnostic tweezers is included with this application note. After applying the previously discussed techniques to attain EMC, if a radiated EMI problem exists, this simple tool may be used to speed up the identification of potential problem areas on a PCB.

CONCLUSION

EMI control has left the specialized realms of electronic design (e.g., military) and is rapidly becoming an industry-wide phenomenon. Although the application of good system design will always be a prerequisite to achieving EMC, it is reasonable to suppose that similar design concepts could also be applied to the source of most of the radiation, the VLSI HCMOS device. To respond to these and other customer demands for higher performance machines, Motorola is investigating new system and circuit design, layout, and alternative packaging techniques. This research may help to reduce the likelihood of problematic RFI when using HCMOS MCU devices; however, the user's awareness and understanding of the problem will remain the most vital step towards product EMC.

REVIEW OF KEY POINTS

Differential-mode radiation features are as follows:

1. The system clock is often the primary source of radiation. Avoid ground loops and long tracks (always take the most direct route). Wherever possible, clock tracks (or any other signal) should have adjacent ground-return tracks. Minimize the number of devices requiring the system clock. Ensure that clock circuitry and associated lines are located well away or shielded from PCB I/O tracks or circuitry. Never mix clock and bus or I/O drivers in the same package — use separate buffer drivers for clock and buses.

2. Ensure that decoupling capacitors are as close as possible to the device supply pins to reduce the loop area through the capacitor. Always parallel decouple large-value (DC ballast) capacitors with one or more smaller high-frequency capacitors (check their equivalent series inductance (ESL) and maximum frequency rating).
3. In addition to local device decoupling, decouple the power supply where it enters the PCB. A ferrite bead (e.g., $Z > 50 \Omega$ at 100 MHz) will also help prevent switching transients from going off-board.
4. For PCBs without a ground plane, minimize address/data line loop areas by routing a minimum of one ground-return track adjacent to each of the eight lines and by keeping the lines as short as possible. For the address lines, route the ground return next to A0 because this line is likely to be the most active. Note also that long address lines will ring, which is another potential source of RFI. These lines may need to be individually terminated (see Reference 4). Operating an MCU in single-chip mode will almost eliminate radiation from address/data lines (still exists internally, of course).
5. Avoid ground loops. Remember that breaking a loop with a small gap may be fine at DC but gap capacitance may effectively close the loop at RF frequencies, creating a large loop antenna. Apart from the radiation problems, large ground loops can make a system more susceptible to malfunction when subjected to external EMI sources.
6. Using a printed copy of the PCB artwork and a marker pen, trace the ground and supply tracks. Long, thin, or looped tracks can then be easily identified and subsequently modified.
7. Terminate all unused inputs to prevent unintentional random switching and noise generation (in addition, unterminated CMOS inputs tend to self-bias into the linear region of operation, significantly increasing the DC current drawn).
8. The smaller footprint of surface-mount components may be used advantageously to reduce loop areas.

Common-mode radiation features are as follows:

1. Ensure a good ground plane and choose the external ground connection to minimize the overall common-mode voltage drop (see Reference 4). Increase both supply and return-supply track widths (as a general rule, cover as much as possible of the unused part of a PCB).
2. A grounding scheme that isolates digital and I/O (including any analog sections) reduces radiation from I/O cables. Shielding these cables is ineffective if the shield termination grounds are noisy. In digital systems, the shield should be connected to noise-free grounds at both ends. If this configuration is not possible, then ground only the source end.

3. A choke may be effective in reducing the radiation from an I/O cable. Also available are a variety of other passive RFI filter elements which shunt the common-mode current to ground. The effectiveness of these devices will depend upon the condition of the shunt ground.

REFERENCES

1. FCC. "Understanding the FCC Regulations Concerning Computing Devices." *OST Bulletin*, vol. 62, 1984.
2. Mardiguian, Michel. *Interference Control in Computers and MPU-Based Equipment*, Gainesville, Va.: Don White Consultants Inc., 1984.
3. Ott, Henry W. *Noise Reduction in Electronic Systems*. 2nd Edition. New York: Wiley Interscience, 1988.
4. Ott, Henry W. "Digital Circuit Grounding and Interconnection," *IEEE International Symposium on Electromagnetic Compatibility*, August 1984.
5. EMC Technology and Interference Control News magazine, Gainesville, Va: Don White.

EMI/RFI Diagnostic Tweezer Probes: A Construction Article

by Frank Moriarty
General DataComm, Inc.
Middlebury, CT 06762

The RF bypassing tweezer probes described in this article will give their users the ability to quickly and conveniently locate EMI trouble spots at which a permanent suppression component would likely lower the overall EMI. Equally important, these probes will quickly eliminate those circuit points at which a permanent bypassing fix would not be effective. Though simple (isolated tweezer blades spanned with a capacitive element), their effectiveness greatly reduces EMI troubleshooting time.

This article includes three different models of the basic idea, i.e., a fixed, broadband, non-tuning probe; a tunable model; and a band-switching (three bands) model. The practical performance of each probe was initially tested in both active square wave and coaxial sine wave test circuits. Also, these probes have been used for several years in "on the job" RF suppression work. The time saved has been priceless, and head scratching is held to a minimum. I am sure that the tweezer probe will also become an indispensable suppression tool in your bag of EMI tricks.

While the usefulness of these probes is perhaps priceless, the material count and cost will be extremely low—under \$5.00—as shown in Table 1. These probes are also available pre-built.

To help in choosing which probe would be best suited to your particular situation, see Table 2. It lists the frequency bandwidths and average attenuation for each of the three types of probes investigated. The Table 2 data was derived from comparing sine wave (50 Ω coaxial) and active circuit probe frequency/attenuation signatures, and it reflects the most conservative data produced from these two test methods. The active circuit chosen, a 5.376 MHz,

TTL, square-wave hybrid oscillator, produced an output rich in harmonics. The controlled impedance data (sinewave, 50 Ω coaxial) was, of course, much more predictable. Though different circuits will produce different data, it is felt that Table 2 can be considered a general guide in probe frequency selection and can also be used as an aid in bypass selection.

The base for either of these RF bypassing probes is a simple pair of special tweezers. Tweezers enable quick, one-hand, firm mechanical connection to circuit points. They can be spread to reach circuit points as much as 9 cm (3.5 in) apart, which will cover most applications. Longer tweezers could be substituted for greater spans; however, the increased blade inductance would have to be taken into consideration as the bandwidth parameters would change. The tweezer blades' inherent low impedance over the frequency range tested (82 nH average) is low enough for them to be of practical use in this application, over the frequency range covered in this article. Figure 1 illustrates the tweezer assembly, and Fig. 2 is a photo of a completed assembly.

General Construction Steps

1. The first problem is to split a pair of tweezer blades into two halves. High-quality, spot-welded tweezers are very difficult to split. The lower-quality type, listed in the Table 2 parts list, are very easy to split as they are the unwelded, single-piece, fold-over type. These are quite adequate for this application. If you choose to split the high-quality variety, I recommend drilling

Table 1—Parts List

Qty.	Item	Vendor	Part #	Cost
1 ea	Tweezers	**	7948	\$1.97
A/R	Epoxy, shrink tubing, (3/16 & 1/2") Insulator, Wire	*	N/A	\$0.10
GENERAL, (all types)				
Fixed Probes				
A/R	Capacitors, (Fixed) 220, 47, 20 10 pF	***	C40 series	\$0.43
Tunable Probe				
1	Tuning Capacitor (25 tp 150 pF)	ARCO	Arco 424	\$1.20
Bandswitching Probe				
1	Toggle Switch, SPDT, Center Off	*	Any	\$1.95
1	Capacitor 220, 39, 20 pF	***	C40 series	\$0.43

Table 2—Tweezer Performance Chart

Probe Type	Bypass Element	Span in MHz	Bandwidth in MHz	Average Atten.,dB
Fixed	220 pF	31 - 85	54	17
Fixed	47 pF	70 - 129	59	13
Fixed	20 pF	108 - 150	42	11
Fixed	10 pF	140 - 220	80	11
Band-Switching (Low Band)	220 pF	27 - 71	44	12
Bandswitching (Mid Band)	39 pF	68 - 120	52	12
Bandswitching (High Band)	20 pF	105 - 140	35	11
Tuning	25/150 pF	30 - 140	110	14
SUMMARY				
Probe	Best BW in MHz**	Best Atten. in dB *	Highest Freq.	Most * Convenient
Fixed	X		X	
Bandsw.				X
Tuning		X		

* Based on 30 to 140 MHz Span

** Evaluation based on a set of four fixed probes.

NOTE: Tweezer parameters derived from sine, and TTL square wave test circuits, (worst case listed), and are subject to variations from circuit to circuit.

out the spot welds first. After splitting the blades, you may want to shape the tips on a grinding stone. For getting between narrow chip pins, these tips should be thin. Sharp points are also helpful for good probe contact.

- The next task will be to electrically isolate each tweezer blade from the other and then to bond the two blades together. Sandwich a piece of thin insulator (a piece of wooden ice cream stick 25 mm [1 in] long will suffice) at the upper portion of and between the tweezer halves as shown in Fig. 1. Secure the assembly with alligator clips or shrink tubing. Be sure to keep the tweezer tips parallel. Cover the top 3 mm (0.125 in) portion of the assembly with a small piece of shrink tubing so that you will have an epoxy-free area for soldering the capacitor later on.
- Liberaly coat all sides of the top 33 mm (1.3 in) of your assembly with a generous amount of epoxy. Allow 24 hours hardening time to ensure a good set.
- Choose the assembly you wish to build; i.e., fixed, tunable or band-switching probe per Table 2. Each probe type has its advantages and disadvantages in bandwidth covered, attenuation characteristics and convenience. The chief advantage of the band-switching and fixed probes are their non-tuning convenience. The chief disadvantage of the fixed probes is the need to have more than one probe to cover a wider frequency range. On the other hand, the tunable probe covers a wide range of frequencies but requires tuning. Overall, the band-switching probe offers the most convenience with a slight loss in attenuation capability. Personally, I have made good use of all three types. Your situation, bandwidth of interest, etc. will be major factors in this decision.

If you choose to build the single-element probe and, depending on the frequency range you wish to cover, you may want to build three or more with different capacitive elements to cover a wider frequency range. Please note that there is an upper frequency limit in the practicality of any of these probes, as covered later in this article.

Fixed-Element Probe, Final Assembly

- Referring to Fig. 1 as a general guideline, choose the capacitor element you wish to solder to the tweezer blades. Note that the upper frequency limit is roughly 300 MHz here, due to hand capacitance, the basic tweezer capacitance of 10 pF average and the tweezer blades' inductance of 82 nH average. The author limited testing and use to 220 MHz.

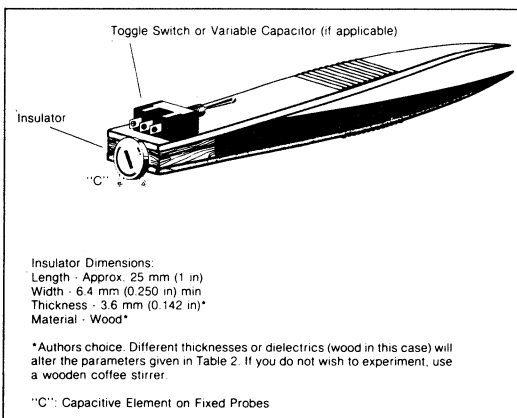


Figure 1—Tweezer Assembly (Typical)

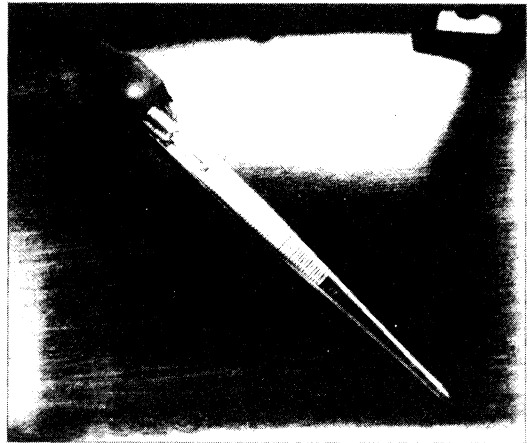


Figure 2—(Bandswitching Probe Shown)

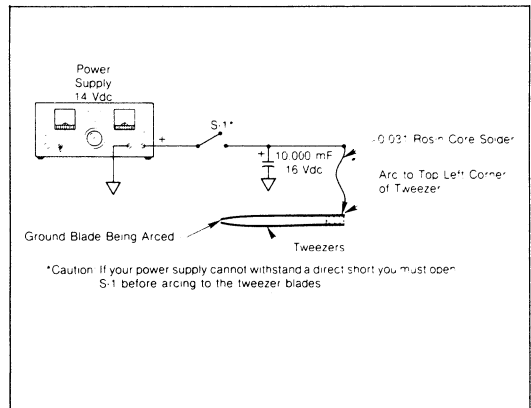


Figure 3—Electro-Flux

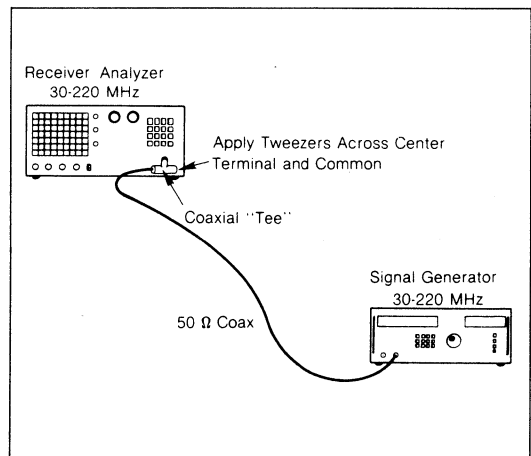


Figure 4—Probe Testing Setup

2. Solder the capacitor across the top of each of the tweezer halves as in Fig. 1. Keep capacitor lead lengths short and use heat sinks on them.

Note: Soldering to stainless steel tweezers requires a special flux. I tried several fluxes, including ammonia, with no success. If you do not have the proper flux available, then try a method I call "electro flux." Figure 3 shows a schematic for the electrical part of this method. With this hookup, alternately charge and then discharge the large electrolytic capacitor, through a length of solder, to the top left corners of each tweezer half. The arcing between the solder element and the tweezer will eventually cause a solder buildup at this point. This buildup will not be readily obvious, but it will be quite adequate when tinned. For tinning, use a good electronic grade of rosin core solder. A paste flux would also aid in this tinning.

3. Decide on the desired spread of the tweezer blades. For instance, if the largest spread you will require is for a 40-pin chip, then spread your tweezer blades equally for a 50 mm (2 in) gap. Caution: when bending the tweezer blades, avoid putting excessive stress on the epoxy bond.
4. Finish off the probe with shrink tubing on each blade and on the top portion of the tweezer assembly to help reduce hand capacitance effects and possible PC card shorts (should you happen to drop the probe into a circuit card). Note: this top covering could be of a removable type which would enable the removal and soldering of different capacitors for experimental purposes or to avoid building two or more probes to cover wider frequency ranges.

Testing Your Probe

1. Make the 50 Ω coaxial test setup shown in Fig. 4.
2. Connect your probe across the open end of the coaxial "Tee" connector.
3. Your readings, being in a controlled impedance test circuit, will only approximate the data listed in Table 2, which was derived by comparing data from differing circuits and indicates the worst-case effects of that comparison.
4. Of course, if you are testing the tunable probe, you will have to tune for each frequency being tested to get maximum attenuation.

Use of the Single-Element Probe

As a typical use of the probe would be in chasing down radiated EMI signals, the following will cover that aspect of its use. Adjustments can be made for other situations.

Caution: Do not use these probes in circuits containing voltages in excess of 50 Vdc.

1. Begin by tuning in the EMI signal in question. Determine its exact frequency. A schematic analysis of various EMI harmonic frequencies or circuit probing with a "sniffer probe" (see Table 1) will determine the location of suspected circuit points which might be generating the offending frequencies.
2. Assuming a typical radiated test setup using an antenna for a pickup device and a receiver/analyzer to observe the EMI, tune in the EMI signal to be investigated and use your tweezer probe to probe the suspected circuit "hot spots" as determined in step 1, above.

Note: As you are using a bypass device, one blade of the tweezer must be in firm contact with a circuit common point, and the other blade should be in contact with the circuit point under investigation. The best common point is generally the common pin on the chip being probed.

Should probing show little or no decrease in the observed EMI at a particular circuit point, or be seen to increase the EMI at this point, is generally an indication that this circuit point would not yield to a successful, permanent EMI bypass fix. However, the points which indicate a small decrease (3 or 4 dB) may be part of a dual EMI problem, i.e., a circuit which has two or more problem areas. Keep these points in mind if later probing is inconclusive. Such a condition may require two or more permanent fixes. In any event, with experience your probing analysis will keep time-consuming "cut and try" methods to a minimum.

Any circuit point which indicates a moderate or large decrease due to probing should be noted as a likely point for a permanent EMI bypass fix. As will all things, experience will bring more knowledge. As you use your probes more often, you will be able to quickly determine those areas at which a permanent fix would be successful. Also, you will be able to more accurately choose the type and size of fix to use, i.e., a series or bypassing fix. Other types of EMI fixes which might be required, such as shielding or grounding improvements, will also be determined by these methods.

Tuning Probe

1. Refer to the previous "general construction" and applicable "fixed probe" paragraphs to create a completed tweezer assembly, minus the capacitive element. One exception here would be to apply the shrink tubing on the tweezer blades before joining them. Shrink tubing should stop approximately 25 mm (1 in) from the top of each blade for bonding purposes.

Remove any unneeded mounting parts from the bottom of the variable capacitor so that it will fit as closely as possible to the tweezer blade. You may also want to consider a clearance hole in the tweezer blades for the bottom of the capacitor's adjustment screw. Be sure the variable capacitor's active parts do not come in contact with either of the tweezer blades.

Solder the variable capacitor to the upper half of the tweezer blades. Solder one element of the variable capacitor directly to a tweezer blade. The other half of the capacitor element will have to be soldered through a short length of wire. Keep this wire as short as possible to hold your usable bandwidth as wide as possible.

It is good practice to have the movable portion of the variable capacitor (rotor) as the circuit ground connection. The rotor part of the tweezer assembly can easily be identified by stripping back a longer piece of the shrink tubing on the tip portion of the rotor's tweezer blade.

2. Carefully epoxy the variable capacitor to a tweezer blade. Be sure not to get epoxy on any of the capacitor's moving parts. Allow ample hardening time, then cover the assembly with shrink tubing. Be sure the shrink tubing allows for free movement of the capacitor's moving parts. The minimum movement of the assembly must allow for three full turns on the variable capacitor's adjustment screw.

Using the Tuning Probe

1. With your radiated test setup intact, place a "Tee" coaxial adaptor at the input of the receiver/analyzer.
2. Tune in the EMI signal to be investigated.
3. Connect your tuning probe across the open end of the coaxial "Tee" and tune the variable capacitor for a minimum signal as observed on your receiver. For maximum attenuation due to circuit detuning, an optional, additional in-circuit tweak might be desirable.

Note: Resonant tuning begins at approximately 60 MHz and ends around 130 MHz. Frequencies below 50 MHz are probed with the adjusting screw fully clockwise (tight). Likewise, fre-

quencies above 130 MHz are probed with minimum capacitance (three turns counterclockwise on the adjustment screw). The frequencies from 60 to 130 MHz should be tuned to resonance (minimum amplitude as seen on your receiver/analyzer). Of course, different assemblies will vary somewhat from these parameters.

Your tuning probe is now tuned for optimum attenuation at the EMI frequency under investigation.

4. Refer to the preceding section, "Using the Single-Element Probe," for a guide to probing.

Band-Switching Probe

The band-switching probe was the next obvious step in the evolution of these probes. It performs quite well and is very convenient to use.

1. Proceed with the basic construction steps as outlined in the "general construction" paragraphs, with the exception of leaving an epoxy-free area for mounting the band switch and applying the shrink tubing on the tweezer blades before bonding.
2. Mount, secure and epoxy the band switch to one of the tweezer blades (see Fig. 1). Allow ample hardening time for the epoxy.
3. Referring to Fig. 5, mount the three capacitors as shown and wire the band switch. Use the capacitor leads for wiring, keep lead lengths short and apply heat sinks to the capacitors before soldering. Note that capacitor logic is single capacitor, series, hand series parallel (moving toggle from left to right). This method was chosen to obtain closer overlapping bands and to take advantage of the band switch's terminals for mounting the capacitors.
4. Cover the capacitors and exposed band switch terminals with heat shrink tubing.

Using the Band-Switching Probe

Use this probe as with the fixed probes, with the exception of the band-switching feature. If wired as shown in Fig. 5 (pictorial view), and if you are using a typical toggle switch, the low band will be with the toggle left of center, the high band will be with the toggle in mid position (off), and the mid band will be with the toggle in the far-right position.

Note: If it is desired to know the actual capacitance involved in a particular fixed-probe effect, simply add 15 pF (residual capacitance of the tweezers when constructed as per Fig. 1) to the probe's capacitive element value. The capacitance of the tuning probe, however, would have to be measured. For the band-switching probe, being series and series parallel wired, add the 15 pF to the following:

Low band, 220 pF
 Mid band, 47 pF
 High band, 19 pF

This information is useful when looking for a minimum capacitance for a permanent bypass fix when working with a circuit which cannot drive a large capacitance without unacceptable distortion of the wave shape. □

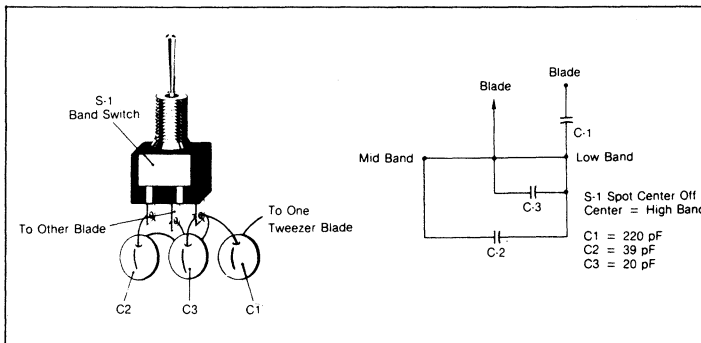


Figure 5—Bandswitching Wiring

Note: the probes discussed in this article are also available in pre-built form. For a price list and ordering information, contact F&M Electronics, 41 Chestnut St., Seymour, CT 06483, (203) 888-4847.

M6805 16-Bit Support Macros

If your microcontroller (MCU) application requires a small amount of program memory and not much raw computing power, the M6805 MCU Family is a most logical choice, given their low cost. But do not cross the M6805 Family off your selection list just because you need some 16-bit indexing and/or 16-bit operations, such as the higher cost M68HC11 Family provides. While the 8-bit X index register of the M6805 Family cannot access the entire M6805 12/13-bit address space and its single 8-bit accumulator cannot directly do 16-bit operations, advanced software techniques can be employed to work around the limitations of the M6805 Family. This application note gives specific details and examples of these techniques.

The code samples given here are available in source code form on the Motorola FREEWARE Bulletin Board Service (BBS), by telephoning 512/891-FREE (512/891-3733). The FREEWARE line operates continuously (except for maintenance) at 300-2400 baud, 8 data bits, 1 stop bit, and no parity. Sample test files are also included. Download the archive file, MACROS05.ARC, to get all the files. All files are suitable for use with the Motorola Development Systems M6805 Portable Assembler for MS-DOS, known as PASM05. Other assemblers may also be supported, but consult the BBS for details.

The techniques used here involve a combination of macros and RAM-based subroutines which use instruction modification. Macros allow programming on a higher level of thought with less chance for introducing errors.

Instruction modification is a technique of altering an instruction just prior to its execution. The modification requires that the instruction be in RAM and can involve the opcode and/or the operand portion of the instruction. By determining the exact instruction/operand needed in advance of execution, greater efficiency in execution speed and code size can be obtained. There is a significant risk in using the instruction modification technique because if used improperly, the program will either fail to work properly at best, or crash at worst. When the technique is used, great care must be exercised to ensure correct operation in all possible cases.

To illustrate the instruction modification technique, consider the following instruction at location \$0050 in RAM memory.

```
0050 C6 04FF LDA $4FF
```

The LDA instruction consists of three bytes: an opcode byte (\$C6) and two bytes that hold the extended address (\$04FF) of the operand. When executed, the A accumulator will be loaded with the contents of location \$04FF. Now consider what happens when the following instructions at location \$870 are executed and how the previous memory locations are changed.

```
0870 A6 05 LDA #$05
0872 B7 51 STA $51
0874 A6 2C LDA #$2C
0872 B7 52 STA $52
```

These instructions store \$05 into location \$0051 and \$2C into location \$0052, which is the operand address of the opcode byte at location \$0050. This has the effect of changing the instruction at location \$0050 to the following.

```
0050 C6 052C LDA $52C
```

When location \$0050 is executed, the A accumulator will now be loaded with the contents of location \$052C, i.e., the instruction at location \$0050 has been modified! Note that the original source listing will only show "LDA \$4FF", as the instruction is only changed at execution time, not assembly time.

When instruction modification is used in a ROM-based system, the RAM code must be initialized (from the ROM) before being used. This can be as simple as a few LOAD/STORE instructions for a small routine, or a MOVE BLOCK routine may be required for larger routines.

As with all things of value, there is a price to be paid. The price for using these macros is rather small, namely, 23 bytes (16 for RAM subroutines and 7 for local storage) of direct addressing memory, i.e., in the range of \$0-\$00FF. The macros have a small size/execution speed penalty associated with them that varies from zero to 50 percent, depending on the frequency and type of macros used. An estimated typical cost for an entire program with moderate macro usage would be in the 5 to 20 percent range. But this is small cost to pay for error-free code generation in areas of the program where speed is not critical. By judicious usage/choice of macros, the cost can be held closer to the 5 percent range.

Part of this cost is associated with the structured code technique of preserving registers and another part is involved with setting up the proper condition codes. The rest of the cost is inherent in the fact that M6805 MCUs must do extra work in software to simulate the hardware capabilities built in M68HC11 MCUs. Because these macros have been fine-tuned for size and speed efficiencies, the overhead cost of register preservation is typically 4 bytes and the overhead cost of condition code setting is typically 2 to 4 bytes, per macro invocation. It can be as high as 200 percent, as is the case of the MOV.B macro for extended addressing operands (16 versus 8 bytes if no register preservation/condition code initialization is done) or as low as zero percent for direct addressing operands.

The DREG macros have almost no overhead associated with them since the DREG is implemented using the already existing A and X registers. The overhead price for DREG macros is only 2 bytes of local storage (TEMPA\$, TESTA\$), because the RAM subroutines are only needed for XREG and YREG support. The Add and Subtract DREG macros (ADDD, SUBD) have the most overhead because they must save and restore a working register (A), but even this is rather minimal. If only DREG macros are used, it is estimated that the code size could expand from zero to 5 percent over straight assembly language, depending on how many and which type of DREG macros are used.

The XREG and YREG macros have the most overhead, since they are implemented using RAM memory locations. The Load and Store via XREG/YREG macros (LDAXY, STAXY) have the most overhead when nonzero offset values are used (26 bytes versus 2 bytes for zero offset, or 1300 percent). Thus, nonzero offset usages should be avoided unless absolutely necessary. There is also some inefficiency associated with internally maintaining two copies of each register, but it actually helps in the overall implementation.

The Increment, Decrement, and Move macros (INC.B, INC.W, DEC.B, DEC.W, MOV.B, MOVE.W, MOVE) have zero to high overhead. The DEC.B and INC.B macros have zero overhead when used with direct addressing operands, while the INC.W and DEC.W macros have high overhead due to setting the proper condition code based on the resultant 16-bit value.

To use these 16-bit macros, here is a quick summary of the steps involved for use on an IBM-PC with the Development Systems PASM05 macro assembler, which should already have been installed per the instructions supplied with the product. The sample MS-DOS commands are shown in upper case for clarity only (except as noted), as MS-DOS accepts either upper or lower case.

1. Create a new directory for your project and change directory to it as shown below.

```
C>MKDIR PROJ
C>CD PROJ
```

2. Download the archive file, MACROS05.ARC, from the Motorola FREEWARE Bulletin Board to your project directory and then de-archive it as shown below. See the FREEWARE bulletin file, archive.bul, for de-archiving information.

```
C>PKARC MACROS05.ARC
```

3. Read the READ.ME file first, and then read the Notes header in the MACROS05.MAC and RAMSBR.INI files. In the MACROS05.MAC file, study the individual macro headers of the macros you intend to use, especially the examples shown.

4. Write your source code using the example shown in the Notes header of the MACROS05.MAC file, as illustrated in Listing 1, lines 1030-1460. If the two ORG statements, lines 1130 and 1180, are not valid for your application's memory map, change them to the appropriate values. Notice especially how the initialization of the RAM subroutines is accomplished by the MOVE macro in line 1200 using the ROM code generated by the INCLUDE statement in line 1400. Or you can expand the BBS EXAMPLE.S file into your source file by first making a copy of it and then editing the copy as shown below (PE is the IBM Personal Editor command, but you can use any editor you feel comfortable with).

```
C>COPY EXAMPLE.S MYFILE.S
C>PE MYFILE.S
```

5. Invoke PASM05 to assemble your new source file as shown below. Because the options for PASM05 are case sensitive, be sure to enter them just as shown.

```
C>PASM05 -eq -l MYFILE.LST MYFILE.S
```

This produces listing file MYFILE.LST and a COFF relocatable object file MYFILE.O.

6. To produce an absolute S-record object file suitable for programming an EPROM, the COFF relocatable object file must first be linked to an absolute object file (MYFILE.OUT) and then converted to an S-record file (MYFILE.MX), which many commercial EPROM programmers, such as Data I/O, recognize. Enter the commands shown below to create the S-record file.

```
C>PLD -o MYFILE.OUT MYFILE.O
C>UBUILDS MYFILE.OUT
```

Steps 5 and 6 can be simplified by copying and editing the BBS batch file, MAKE1.BAT, in a manner similar to that shown in Step 4, except here we want the new batch file, MAKE.BAT, to process MYFILE instead of TEST1. The resultant MAKE.BAT file should be similar to the text shown below. The two DEL commands at the end are optional, as these files are no longer needed.

```
pasm05 -eq -l myfile.lst myfile.s
pld -o myfile.out myfile.o
ubuilds myfile.out
del myfile.o
del myfile.out
```

To accomplish Steps 5 and 6, invoke the batch file MAKE.BAT by simply typing its name as shown below. MS-DOS will execute each line of the file as if you had typed it on the keyboard.

```
C>MAKE
```

7. Consult your EPROM programmer's user manual for how to load the resultant S-record object file (MYFILE.MX) and program your chosen device.

Because of the length of the source listings involved and because these listings are intended to be self-explanatory, the rest of this discussion will only deal with salient points which might need clarification for the reader. Line numbers have been added at the beginning of each line for identification purposes in this Application Note, i.e., they are not present in the actual source file.

These macros have been written so that any syntax error will result in falling to the end of the macro where the FAIL directive will force a "Macro syntax error detected!" message. Proper usage results in exiting the macro early via the MEXIT directive and thus avoids the FAIL directive at the end.

The file header of **Listing 1** (lines 20-1800) gives specific details of the macros supported and their general operation and restrictions. Line 1810 defines where a seven byte block of low memory will be allocated for support of these macros (see lines 20430-20490). Line 1830 disables the output listing to avoid repetitious output, while line 19630 re-enables the listing.

A lot of conditionals (IFxx) involve testing to see if direct addressing can be used, as it is more efficient (one less byte and one less execution cycle per instruction) than extended addressing, as illustrated by lines 2090-2210. Also, conditionals check to see if shorter instruction forms can be used, like "CLRA" instead of "LDA #0", as in these same lines. For this same reason, it is most efficient to place the RAM subroutines, and thus the XREG and YREG, into direct addressing space (\$0000-\$00BF). Remember that the M6805 uses a fixed stack area in direct addressing space of \$00C0-\$00FF.

The COMPARE macros (CPD, CPXR, CPYR) use a shortened form when immediate addressing is specified and either

of the halves is zero, i.e., no "CMP #0" instruction is generated. Lines 3480-3660 are typical of this technique. As always, testing for zero is most efficient in computer architecture.

The 16-BIT INDEXING macros (LDAXY, STAXY) are most efficient when used with a zero offset, as typified in lines 3980-4060, but will function with any sized offset (lines 4070-4210). Nonzero offsets must first be added to the indexing register (XREG or YREG), the operation performed via the appropriate RAM subroutine, and lastly, the indexing register must be restored to its original value.

Lines 19640-20410 comprise the RAM subroutines which are the keystone of the macros using the XREG and YREG pseudo 16-bit index registers. It is here that the XREG and YREG pseudo-registers are defined and stored as the second and third bytes of extended addressing LDA and STA instructions using EQU directives (lines 19910-19920, 20060-20070, 20210-20220, and 20360-20370). Instructions that store values to XREG or YREG will thus modify the instruction's effective address, hence the name instruction modification. These RAM subroutines must be initialized before they can be used, and so a mirror image of this code with altered labels is provided in the RAMSBR.INI file (as seen in **Listing 2**). This file can then be simply INCLUDED in the ROM section of the user's code and copied to RAM via the MOVE macro, as discussed in lines 19660-19760 of **Listing 1**.

Listing 2 is the RAM subroutine initialization file and is essentially a copy of the RAM subroutines defined in lines 19640-20410 of **Listing 1**, but the labels have all been prefixed with a period (.). This allows using the MOVE macro to copy this RAM initialization code from a ROM to RAM (see lines 370-410).

Lines 1070-1090 verify that the number of RAM subroutine initialization bytes is identical.

Listing 1 – MACROS05.MAC File

```

00010 PAGE
00020 *****
00030 * macros05.mac 1.0
00040 * -----
00050 * Module Name:      MACROS05 - M6805 Macros
00060 * -----
00070 *
00080 * Description:
00090 * This file contains macros and subroutines to support pseudo-registers
00100 * on the 6805 that simulate registers and addressing modes available on
00110 * the 68HC11. It is suitable for "black box" operation, i.e., the
00120 * macros may be used without knowledge of how they work. A list of the
00130 * supported macros follows. Consult the individual macro headers for
00140 * usage details and see the "Notes" below.
00150 *
00160 * LDD Load DREG
00170 * STD Store DREG
00180 * ADDD Add DREG
00190 * SUBD Add DREG
00200 * CPD Compare DREG
00210 * LDAXY Load A via 16-bit pseudo-register (XREG or YREG)
00220 * STAXY Store A via 16-bit pseudo-register (XREG or YREG)
00230 * LDXR Load XREG
00240 * STXR Store XREG
00250 * INCXR Increment XREG
00260 * DECXR Decrement XREG
00270 * CPXR Compare XREG
00280 * LDYR Load YREG
00290 * STYR Store YREG
00300 * INCYR Increment YREG
00310 * DECYR Decrement YREG
00320 * CPYR Compare YREG
00330 * DEC.B Decrement byte
00340 * DEC.W Decrement word
00350 * INC.B Increment byte
00360 * INC.W Increment word
00370 * MOV.B Move byte
00380 * MOV.W Move word
00390 * MOVE Move block of memory
00400 *
00410 * General Information:
00420 * The following pseudo-registers are supported.
00430 * DREG = pseudo 16-bit accumulator (A,X registers, A is MS half)
00440 * XREG = pseudo 16-bit index register
00450 * YREG = pseudo 16-bit index register
00460 *
00470 * The following terms are used.
00480 * # = specifies immediate addressing mode
00490 * <address> = address/value operand (absolute or immediate)
00500 * <offset> = unsigned 16-bit offset for indexed addressing
00510 *
00520 * Notes:
00530 * 1. Motorola reserves the right to make changes to this file.
00540 * Although this file has been carefully reviewed and is believed
00550 * to be reliable, Motorola does not assume any liability arising
00560 * out of its use. This code may be freely used and/or modified at
00570 * no cost or obligation by the user.
00580 * 2. This file was made for use with the Motorola Development Systems
00590 * MC6805 Portable Assembler/Linker for MS-DOS, known as PASM05 and
00600 * PLD, as released on 82HCVBASM B* and 82HCVBLNK B*. Consult the
00610 * PASM and PLD reference manuals, part numbers M68HASM/D1 and
00620 * M68HLINK/D1 for more details.

```

```

00630 *      3. These macros were made for ABSOLUTE assemblies only, i.e., for
00640 *      use with ORG directives. While most of the macro concepts will
00650 *      work in relocatable assemblies (BSCT, DSCT, PSCT, ASCT, XDEF,
00660 *      and XREF), errors will be generated because PASM limits the use
00670 *      of external symbols in expressions and because the value of an
00680 *      expression must be known at assembly time for IFxx directives to
00690 *      assemble the proper code. The first restriction is a result of
00700 *      limitations in the COFF object file format. If it is desired to
00710 *      have these macros work with relocatable assemblies, modifica-
00720 *      tions similar to below should be made, but be forewarned of the
00730 *      increased inefficiencies in size and speed. Consider the
00740 *      following code to change the LDD macro so that an XREF parameter,
00750 *      \1, can be loaded as an immediate value.
00760 *              LDA    \.8
00770 *              LDX    \.8+1
00780 *              BRA    \.9
00790 *              \.8    FDB    \1
00800 *              \.9    EQU    *
00810 *
00820 *      4. In order to efficiently support both LOAD and STORE operations
00830 *      for the pseudo 16-bit index registers, there are actually two
00840 *      such "registers", i.e., one for LOAD and one for STORE as
00850 *      defined below. These routines maintain both "registers" with
00860 *      the same value, and so the programmer may think of them as one
00870 *      register.
00880 *      XREG1$ = 16-bit XREG for LOAD operations
00890 *      XREG2$ = 16-bit XREG for STORE operations
00900 *      YREG1$ = 16-bit YREG for LOAD operations
00910 *      YREG2$ = 16-bit YREG for STORE operations
00920 *
00930 *      5. These macros can be used to create in-line code (speed
00940 *      efficient) or they be placed in a subroutine (byte efficient).
00950 *
00960 *      6. Instruction modified code is used here and is denoted by use
00970 *      of the unique string "0-0" (RAM subroutines).
00980 *
00990 *      7. Some macros use temporary storage locations (TEMPA$, TESTA$,
01000 *      etc.), so these macros should not be used in any interrupt
01010 *      routine in order to avoid corrupted values!
01020 *
01030 *      8. The user must ensure that the code is appropriately placed in
01040 *      the target M6805's memory map, i.e., the RAM subroutines MUST
01050 *      be located in RAM but must not overlap the stack area ($00C0-
01060 *      00FF) unless it can be GUARANTEED there is no conflict! See
01070 *      LO$MEM below to set low memory data storage area!
01080 *
01090 *      9. To use this file, either use an INCLUDE statement or just
01100 *      merge this file into your source code file. Consult your
01110 *      assembler's user's manual for the details specific to your
01120 *      situation. When using a ROM controlled system, the MOVE
01130 *      macro should be used to copy the RAM subroutines from ROM to
01140 *      RAM (see the comments after where RAMSBR$ is defined below
01150 *      and note the INCLUDE statement for the RAMSBR.INI file).
01160 *      Reference the code segment example below for usage ideas
01170 *      (shown in PASM05 for MS-DOS syntax).
01180 *
01190 *      ORG    $50
01200 *      TOTAL  RMB    2
01210 *      RTABLE  RMB    5
01220 *      INCLUDE  MACROS05.MAC
01230 *
01240 *      ORG    $400
01250 *      RESET  RSP
01260 *      MOVE   #, .RAMSBR$, #, RAMSBR$, #, RAMS2$  Init ram.
01270 *      START  MOV.W #, 0, TOTAL
01280 *      LDD   COST
01290 *      ADDD  #, 1000
01300 *      SUBD  #, ADJUST
01310 *      ADDD  TOTAL
01320 *      STDD  TOTAL

```

```

01270 *          CPD      #,1500
01280 *          BEQ      MATCH
01290 *
01300 *          LDXR     #,0
01310 *          LDYR     #,0
01320 *          LOOP    LDAXY  TABLE,XREG
01330 *          STAXY   RTABLE,YREG
01340 *          INCXR
01350 *          INCYR
01360 *          CPY     #,5
01370 *          BNE     LOOP
01380 *
01390 *
01400 *          INCLUDE  RAMSBR.INI
01410 *          TABLE  FCB    1,2,3,4,5
01420 *          ADJUST  FDB    150
01430 *          COST   FDB    859
01440 *
01450 *
01460 *          END
01470 *
01480 *
01490 *          10. To assemble, use the following sample invocation lines:
01500 *          pasm05 -eq -l filename.lst filename.s      (debug= expanded)
01510 *          or
01520 *          pasm05 -bf -l filename.lst filename.s      (black box)
01530 *          11. Notations used by PASM05 are as follows:
01540 *          OPERATORS: Special two character operators used are...
01550 *                   1. Logical AND                !.
01560 *                   2. Shift Right (0 fill on left)  !>
01570 *          MACROS:   Special notations used are...
01580 *                   1. Parameters are positionally named using \0,
01590 *                      \1, \2, etc.
01600 *                   2. Labels within macros are designated via \.a,
01610 *                      where "a" is an alphanumeric character. The
01620 *                      assembler generates a unique label to avoid
01630 *                      multiply defined label problems.
01640 *          12. Some macros access 16-bit values as LS byte then MS byte in order
01650 *          to be more efficient for condition code (CC) setting. This is
01660 *          the reverse order that the 68HC11 would access the two byte
01670 *          halves. This difference would only be a concern in accessing
01680 *          hardware registers, as normal RAM makes no difference. Those
01690 *          macros with this difference have an entry in their Notes section.
01700 *          13. The latest version of this file is maintained on the Motorola
01710 *          FREEMWARE Bulletin Board, 512/891-FREE (512/891-3733). It operates
01720 *          continuously (except for maintenance) at 1200-2400 baud, 8 bits,
01730 *          no parity. Sample test files for PASM05 are also included.
01740 *          Download the archive file, MACROS05.ARC, to get all the files.
01750 *
01760 * *****
01770 * REVISION HISTORY (add new changes to top):
01780 *
01790 * 05/16/90 P.S. Gilmour
01800 * 1. Created for Application Note AN1055.
01810 * *****
01820 * LO$MEM EQU $00C0-7 Low memory ($0000-00FF) storage (7 bytes)

```

```

01830          OPT    NOL
01840 *****
01850 * LDD    = load DREG
01860 *      LDD    [#,<address>
01870 *
01880 * Examples:
01890 *   1. "LDD  #,START"      puts the value of symbol 'START' into
01900 *                          the DREG (=A,X).
01910 *   2. "LDD  START"        puts the contents of location 'START'
01920 *                          and 'START'+1 into the DREG (=A,X).
01930 *
01940 * Register Usage:
01950 *   A,X = loaded with new value (DREG).
01960 *   CC = reflects MS half (=A).
01970 *   All other registers preserved.
01980 *
01990 * Notes:
02000 *   1. Byte access order is LS, then MS (reversed from 68HC11).
02010 *
02020 LDD      MACR
02030   IFEQ   NARG-1
02040     LDX   (\0)+1
02050     LDA   (\0)
02060     MEXIT
02070   ENDC
02080   IFEQ   NARG-2
02090     IFC   '\0','#'
02100       IFEQ (\1)!.$FF
02110         CLRX
02120       ENDC
02130       IFNE (\1)!.$FF
02140         LDX  #(\1)!.$FF
02150       ENDC
02160       IFEQ (\1)!>8
02170         CLRA
02180       ENDC
02190       IFNE (\1)!>8
02200         LDA  #(\1)!>8
02210       ENDC
02220     MEXIT
02230   ENDC
02240   ENDC
02250   FAIL   Macro syntax error detected!
02260   ENDM
02270
02280 *****
02290 * STD    = store DREG
02300 *      STD    <address>
02310 *
02320 * Examples:
02330 *   1. "STD  START"        stores the DREG (=A,X) into locations
02340 *                          'START' and 'START'+1.
02350 *
02360 * Register Usage:
02370 *   CC = reflects MS half (=A).
02380 *   All other registers preserved.
02390 *
02400 * Notes:
02410 *   1. Byte access order is LS, then MS (reversed from 68HC11).
02420 *
02430 STD      MACR
02440     STX   (\0)+1
02450     STA   (\0)
02460   ENDM
02470

```

```

02480 *****
02490 * ADDD = add DREG
02500 * ADDD [#,<address>
02510 *
02520 * Examples:
02530 * 1. "ADDD #, START" adds the value of symbol 'START' to the
02540 * DREG (=A,X).
02550 * 2. "ADDD START" adds the contents of location 'START'
02560 * and 'START'+1 to the DREG (=A,X).
02570 *
02580 * Register Usage:
02590 * A,X = contains new value (DREG).
02600 * CC = reflects MS half (=A).
02610 * All other registers preserved.
02620 *
02630 ADDD MACR
02640 IFEQ NARG-1
02650 STA TEMPAS
02660 TXA
02670 ADD (\0)+1
02680 TAX
02690 LDA TEMPAS
02700 ADC (\0)
02710 MEXIT
02720 ENDC
02730 IFEQ NARG-2
02740 IFC '\0', '#'
02750 STA TEMPAS
02760 TXA
02770 ADD #(\1)!.SFF
02780 TAX
02790 LDA TEMPAS
02800 ADC #(\1)!>8
02810 MEXIT
02820 ENDC
02830 ENDC
02840 FAIL Macro syntax error detected!
02850 ENDM
02860
02870 *****
02880 * SUBD = add DREG
02890 * SUBD [#,<address>
02900 *
02910 * Examples:
02920 * 1. "SUBD #, START" subtracts the value of symbol 'START' from
02930 * the DREG (=A,X).
02940 * 2. "SUBD START" subtracts the contents of location 'START'
02950 * and 'START'+1 from the DREG (=A,X).
02960 *
02970 * Register Usage:
02980 * A,X = contains new value (DREG).
02990 * CC = reflects MS half (=A).
03000 * All other registers preserved.
03010 *
03020 SUBD MACR
03030 IFEQ NARG-1
03040 STA TEMPAS
03050 TXA
03060 SUB (\0)+1
03070 TAX
03080 LDA TEMPAS
03090 SBC (\0)
03100 MEXIT
03110 ENDC

```

```

03120 IFEQ NARG-2
03130 IFC '\0', '#'
03140 STA TEMPAS
03150 TXA
03160 SUB #(\1)!.$FF
03170 TAX
03180 LDA TEMPAS
03190 SBC #(\1)!>8
03200 MEXIT
03210 ENDC
03220 ENDC
03230 FAIL Macro syntax error detected!
03240 ENDM
03250
03260 *****
03270 * CPD = compare DREG
03280 * CPD [#,<address>
03290 *
03300 * Examples:
03310 * 1. "CPD #,BLOCKSZ" compares the value of symbol 'BLOCKSZ'
03320 * with the DREG (=A,X).
03330 * 2. "CPD START" compares the contents of location
03340 * 'START' and 'START'+1 with the DREG.
03350 *
03360 * Register Usage:
03370 * CC = reflects DREG comparison (Z-bit only).
03380 * All other registers preserved.
03390 *
03400 CPD MACR
03410 IFEQ NARG-1
03420 CPX (\0)+1
03430 BNE \.0
03440 CMP (\0)
03450 \.0 EQU *
03460 MEXIT
03470 ENDC
03480 IFEQ NARG-2
03490 IFC '\0', '#'
03500 IFEQ (\1)!.$FF
03510 TSTX
03520 ENDC
03530 IFNE (\1)!.$FF
03540 CPX #(\1)!.$FF
03550 ENDC
03560 BNE \.0
03570 IFEQ (\1)!>8
03580 TSTA
03590 ENDC
03600 IFNE (\1)!>8
03610 CMP #(\1)!>8
03620 ENDC
03630 \.0 EQU *
03640 MEXIT
03650 ENDC
03660 ENDC
03670 FAIL Macro syntax error detected!
03680 ENDM
03690

```



```

03700 *****
03710 * LDAXY = load A via 16-bit pseudo-register (XREG or YREG)
03720 *     LDAXY <offset>,XREG
03730 *     LDAXY <offset>,YREG
03740 *
03750 * Examples:
03760 * 1. "LDAXY 0,XREG"      loads the contents of the memory location
03770 *                        specified by XREG+0 into the A accumulator.
03780 * 2. "LDAXY ,XREG"      loads the contents of the memory location
03790 *                        specified by XREG+0 into the accumulator.
03800 * 3. "LDAXY TBL,XREG"   loads the contents of the memory location
03810 *                        specified by XREG+'TBL' into the A accum-
03820 *                        ulator.
03830 * 4. Above examples can be repeated with substituting YREG for XREG.
03840 *
03850 * Register Usage:
03860 *   A   = loaded with new value.
03870 *   DREG = destroyed.
03880 *   CC  = reflects value loaded.
03890 *   All other registers preserved.
03900 *
03910 LDAXY MACR
03920 IFNC '\1','XREG'
03930 IFNC '\1','YREG'
03940 FAIL Macro syntax error detected!
03950 MEXIT
03960 ENDC
03970 ENDC
03980 IFC '\0',''
03990 JSR LDA\1      Default offset= 0
04000 MEXIT
04010 ENDC
04020 IFNC '\0',''
04030 IFEQ \0
04040 JSR LDA\1      Offset= 0
04050 MEXIT
04060 ENDC
04070 IFNE \0
04080 LDA \11$+1     Set nREG= offset + nREG
04090 ADD #(\0)!. $FF
04100 STA \11$+1
04110 LDA \11$
04120 ADC #(\0)!>8
04130 STA \11$
04140 JSR LDA\1      Offset= 0
04150 STA TEMPAS$
04160 LDA \12$      Restore nREG
04170 STA \11$
04180 LDA \12$+1
04190 STA \11$+1
04200 LDA TEMPAS$
04210 MEXIT
04220 ENDC
04230 ENDC
04240 FAIL Macro syntax error detected!
04250 ENDM
04260

```

```

04270 *****
04280 * STAXY = store A via 16-bit pseudo-register (XREG or YREG)
04290 * STAXY <offset>,XREG
04300 * STAXY <offset>,YREG
04310 *
04320 * Examples:
04330 * 1. "STAXY 0,XREG" stores the accumulator (=A) into the memory
04340 * location specified by XREG+0.
04350 * 2. "STAXY ,XREG" stores the accumulator (=A) into the memory
04360 * location specified by XREG+0.
04370 * 3. "STAXY TBL,XREG" stores the accumulator (=A) into the memory
04380 * specified by XREG+'TBL'
04390 * 4. Above examples can be repeated with substituting YREG for XREG.
04400 *
04410 * Register Usage:
04420 * CC = reflects value stored.
04430 * All other registers preserved.
04440 *
04450 STAXY MACR
04460 IFNC '\1','XREG'
04470 IFNC '\1','YREG'
04480 FAIL Macro syntax error detected!
04490 MEXIT
04500 ENDC
04510 ENDC
04520 IFC '\0',''
04530 JSR STA\1 Default offset= 0
04540 MEXIT
04550 ENDC
04560 IFNC '\0',''
04570 IFEQ \0
04580 JSR STA\1 Offset= 0
04590 MEXIT
04600 ENDC
04610 IFNE \0
04620 STA TEMPA$
04630 LDA \12$+1 Set nREG= offset + nREG
04640 ADD #(\0)!.$FF
04650 STA \12$+1
04660 LDA \12$
04670 ADC #(\0)!>8
04680 STA \12$
04690 LDA TEMPA$
04700 JSR STA\1 Offset= 0
04710 LDA \11$ Restore nREG
04720 STA \12$
04730 LDA \11$+1
04740 STA \12$+1
04750 LDA TEMPA$
04760 MEXIT
04770 ENDC
04780 ENDC
04790 FAIL Macro syntax error detected!
04800 ENDM
04810

```

```

04820 *****
04830 * LDXR = load XREG
04840 * LDXR [#,<address>
04850 *
04860 * Examples:
04870 * 1. "LDXR #,START" puts the value of symbol 'START' into the
04880 * XREG.
04890 * 2. "LDXR START" puts the contents of location 'START' and
04900 * 'START'+1 into the XREG.
04910 *
04920 * Register Usage:
04930 * CC = reflects MS half (=A).
04940 * All other registers preserved.
04950 *
04960 * Notes:
04970 * 1. Byte access order is LS, then MS (reversed from 68HC11).
04980 *
04990 LDXR MACR
05000 IFEQ NARG-1
05010 STA TEMPAS
05020 LDA (\0)+1
05030 STA XREG1$+1
05040 STA XREG2$+1
05050 LDA (\0)
05060 STA XREG1$
05070 STA XREG2$
05080 IFEQ XREG1$!.$FF00
05090 LDA TEMPAS
05100 TST XREG1$
05110 ENDC
05120 IFNE XREG1$!.$FF00
05130 STA TESTAS
05140 LDA TEMPAS
05150 TST TESTAS
05160 ENDC
05170 MEXIT
05180 ENDC
05190 IFEQ NARG-2
05200 IFC '\0', '#'
05210 IFEQ XREG1$!.$FF00 ! XREG in low memory?
05220 IFEQ \1 ! #0 value?
05230 CLR XREG1$+1
05240 CLR XREG2$+1
05250 CLR XREG1$
05260 CLR XREG2$
05270 MEXIT
05280 ENDC
05290 IFNE \1 ! not #0 value?
05300 STA TEMPAS
05310 IFEQ (\1)!.$FF
05320 CLR XREG1$+1
05330 CLR XREG2$+1
05340 ENDC
05350 IFNE (\1)!.$FF
05360 LDA #(\1)!.$FF
05370 STA XREG1$+1
05380 STA XREG2$+1
05390 ENDC
05400 IFEQ (\1)!>8
05410 CLR XREG1$
05420 CLR XREG2$
05430 ENDC

```

```

05440      IFNE  (\1)!>8
05450          LDA  #(\1)!>8
05460          STA  XREG1$
05470          STA  XREG2$
05480      ENDC
05490          LDA  TEMPAS$
05500          TST  XREG1$
05510      MEXIT
05520      ENDC
05530  ENDC
05540  IFNE  XREG1$!.$FF0      ! XREG in high memory?
05550      IFEQ  \1          ! #0 value?
05560          STA  TEMPAS$
05570          CLRA
05580          STA  XREG1$+1
05590          STA  XREG2$+1
05600          STA  XREG1$
05610          STA  XREG2$
05620          CLR  TESTAS$
05630          LDA  TEMPAS$
05640          TST  TESTAS$
05650      MEXIT
05660      ENDC
05670  IFNE  \1          ! not #0 value?
05680          STA  TEMPAS$
05690  IFEQ  (\1)!.$FF
05700          CLRA
05710      ENDC
05720  IFNE  (\1)!.$FF
05730          LDA  #(\1)!.$FF
05740      ENDC
05750          STA  XREG1$+1
05760          STA  XREG2$+1
05770  IFEQ  (\1)!>8
05780          CLRA
05790      ENDC
05800  IFNE  (\1)!>8
05810          LDA  #(\1)!>8
05820      ENDC
05830          STA  XREG1$
05840          STA  XREG2$
05850          STA  TESTAS$
05860          LDA  TEMPAS$
05870          TST  TESTAS$
05880      MEXIT
05890      ENDC
05900      ENDC
05910  ENDC
05920  ENDC
05930      FAIL  Macro syntax error detected!
05940      ENDM
05950

```

```

05960 *****
05970 * STXR = store XREG
05980 * STXR <address>
05990 *
06000 * Examples:
06010 * 1. "STXR START" stores the XREG into locations 'START' and
06020 * 'START'+1.
06030 *
06040 * Register Usage:
06050 * CC = reflects MS half (=A).
06060 * All other registers preserved.
06070 *
06080 * Notes:
06090 * 1. Byte access order is LS, then MS (reversed from 68HC11).
06100 *
06110 STXR MACR
06120 STA TEMPA$
06130 LDA XREG1$+1
06140 STA (\0)+1
06150 LDA XREG1$
06160 STA (\0)
06170 IFEQ XREG1$!. $FF00
06180 LDA TEMPA$
06190 TST XREG1$
06200 ENDC
06210 IFNE XREG1$!. $FF00
06220 STA TESTA$
06230 LDA TEMPA$
06240 TST TESTA$
06250 ENDC
06260 ENDM
06270
06280 *****
06290 * INCR = increment XREG
06300 * INCR [[#,<address>]
06310 *
06320 * Examples:
06330 * 1. "INCR" adds one (1) to the XREG.
06340 * 2. "INCR #, START" adds the value of symbol 'START' to the
06350 * XREG.
06360 * 3. "INCR START" adds the contents of location 'START' and
06370 * 'START'+1 to the XREG.
06380 * 4. "INCR ! comment" adds one (1) to the XREG (comment present!).
06390 *
06400 * Register Usage:
06410 * CC = reflects value incremented (Z-bit only).
06420 * All other registers preserved.
06430 *
06440 * Notes:
06450 * 1. Explicit comment character (!) MUST be used when comment field is
06460 * present to prevent confusion with parameters!
06470 * 2. Assumes XREG1$ = XREG2$.
06480 * 3. When parameters are present, this macro becomes "ADD to XREG"
06490 *
06500 INCR MACR
06510 IFEQ NARG
06520 IFEQ XREG1$!. $FF00
06530 INC XREG1$+1
06540 INC XREG2$+1
06550 BNE \.0
06560 INC XREG1$
06570 INC XREG2$
06580 \.0 EQU *
06590 MEXIT
06600 ENDC

```

```

06610      IFNE  XREG1$!.$FF00
06620          STA  TEMPAS$
06630          LDA  XREG1$+1
06640          ADD  #1
06650          STA  XREG1$+1
06660          STA  XREG2$+1
06670          LDA  XREG1$
06680          ADC  #0
06690          STA  XREG1$
06700          STA  XREG2$
06710          ORA  XREG1$+1
06720          STA  TESTAS$
06730          LDA  TEMPAS$
06740          TST  TESTAS$
06750          MEXIT
06760      ENDC
06770  ENDC
06780  IFEQ  NARG-1
06790          STA  TEMPAS$
06800          LDA  XREG1$+1
06810          ADD  (\0)+1
06820          STA  XREG1$+1
06830          STA  XREG2$+1
06840          LDA  XREG1$
06850          ADC  \0
06860          STA  XREG1$
06870          STA  XREG2$
06880          ORA  XREG1$+1
06890          STA  TESTAS$
06900          LDA  TEMPAS$
06910          TST  TESTAS$
06920          MEXIT
06930  ENDC
06940  IFEQ  NARG-2
06950      IFC  '\0', '#'
06960          STA  TEMPAS$
06970          LDA  XREG1$+1
06980          ADD  #(\1)!.$FF
06990          STA  XREG1$+1
07000          STA  XREG2$+1
07010          LDA  XREG1$
07020          ADC  #(\1)!>8
07030          STA  XREG1$
07040          STA  XREG2$
07050          ORA  XREG1$+1
07060          STA  TESTAS$
07070          LDA  TEMPAS$
07080          TST  TESTAS$
07090          MEXIT
07100  ENDC
07110  ENDC
07120      FAIL  Macro syntax error detected!
07130  ENDM
07140

```

```

07150 *****
07160 * DECXR = decrement XREG
07170 *   DECXR [[#,]<address>]
07180 *
07190 * Examples:
07200 *   1. "DECXR"                subtracts one (1) from the XREG.
07210 *   2. "DECXR #,START"       subtracts the value of symbol 'START' from
07220 *                               the XREG.
07230 *   3. "DECXR START"         subtracts the contents of location 'START'
07240 *                               and 'START'+1 from the XREG.
07250 *   4. "DECXR ! comment"     subtract one from the XREG (comment present!).
07260 *
07270 * Register Usage:
07280 *   CC = reflects value decremented (Z-bit only).
07290 *   All other registers preserved.
07300 *
07310 * Notes:
07320 *   1. Explicit comment character (!) MUST be used when comment field is
07330 *       present!
07340 *   2. Assumes XREG1$ = XREG2$.
07350 *   3. When parameters are present, this macro becomes "SUBTRACT from XREG".
07360 *
07370 DECXR MACR
07380     IFEQ     NARG
07390         STA     TEMPAS
07400         LDA     XREG1$+1
07410         SUB     #1
07420         STA     XREG1$+1
07430         STA     XREG2$+1
07440         LDA     XREG1$
07450         SBC     #0
07460         STA     XREG1$
07470         STA     XREG2$
07480         ORA     XREG1$+1
07490         STA     TESTAS
07500         LDA     TEMPAS
07510         TST     TESTAS
07520         MEXIT
07530     ENDC
07540     IFEQ     NARG-1
07550         STA     TEMPAS
07560         LDA     XREG1$+1
07570         SUB     (\0)+1
07580         STA     XREG1$+1
07590         STA     XREG2$+1
07600         LDA     XREG1$
07610         SBC     \0
07620         STA     XREG1$
07630         STA     XREG2$
07640         ORA     XREG1$+1
07650         STA     TESTAS
07660         LDA     TEMPAS
07670         TST     TESTAS
07680         MEXIT
07690     ENDC

```

```

07700 IFEQ NARG-2
07710 IFC '\0', '#'
07720 STA TEMPA$
07730 LDA XREG1$+1
07740 SUB #(\1)!.$FF
07750 STA XREG1$+1
07760 STA XREG2$+1
07770 LDA XREG1$
07780 SBC #(\1)!>8
07790 STA XREG1$
07800 STA XREG2$
07810 ORA XREG1$+1
07820 STA TESTA$
07830 LDA TEMPA$
07840 TST TESTA$
07850 MEXIT
07860 ENDC
07870 ENDC
07880 FAIL Macro syntax error detected!
07890 ENDM
07900
07910 *****
07920 * CPXR = compare XREG
07930 * CPXR [#,<address>
07940 *
07950 * Examples:
07960 * 1. "CPXR #,BLOCKSZ" compares the value of symbol 'BLOCKSZ'
07970 * with the XREG.
07980 * 2. "CPXR START" compares the contents of location
07990 * 'START' and 'START'+1 with the XREG.
08000 *
08010 * Register Usage:
08020 * CC = reflects XREG comparison (Z-bit only).
08030 * All other registers preserved.
08040 *
08050 CPXR MACR
08060 IFEQ NARG-1
08070 STA TEMPA$
08080 BSET 0,TESTA$ Preset for .NE. condition!
08090 LDA XREG1$+1
08100 CMP (\0)+1
08110 BNE \.0 Branch if LS half is .NE.
08120 LDA XREG1$
08130 CMP (\0)
08140 BNE \.0 Branch if MS half is .NE.
08150 CLR TESTA$ Set for .EQ. condition!
08160 \.0 LDA TEMPA$
08170 TST TESTA$ Set proper Z-bit (.EQ. or .NE.)!
08180 MEXIT
08190 ENDC

```



```

08200  IFEQ  NARG-2
08210  IFC   '\0', '#'
08220  IFEQ  \1
08230  IFEQ  XREG1$!. $FF00
08240  TST   XREG1$+1
08250  BNE   \.0      Branch if LS half is .NE.
08260  TST   XREG1$
08270  \.0  EQU   *
08280  MEXIT
08290  ENDC
08300  IFNE  XREG1$!. $FF00
08310  STA   TEMPA$
08320  BSET  0, TESTA$  Preset for .NE. condition!
08330  LDA   XREG1$+1
08340  BNE   \.0      Branch if MS half is .NE.
08350  LDA   XREG1$
08360  BNE   \.0      Branch if MS half is .NE.
08370  CLR   TESTA$    Set for .EQ. condition!
08380  \.0  LDA   TEMPA$
08390  TST   TESTA$    Set proper Z-bit (.EQ. or .NE.)!
08400  MEXIT
08410  ENDC
08420  ENDC
08430  STA   TEMPA$
08440  BSET  0, TESTA$  Preset for .NE. condition!
08450  LDA   XREG1$+1
08460  IFNE  (\1)!. $FF
08470  CMP   #(\1)!. $FF
08480  ENDC
08490  BNE   \.0      Branch if LS half is .NE.
08500  LDA   XREG1$
08510  IFNE  (\1)!>8
08520  CMP   #(\1)!>8
08530  ENDC
08540  BNE   \.0      Branch if MS half is .NE.
08550  CLR   TESTA$    Set for .EQ. condition!
08560  \.0  LDA   TEMPA$
08570  TST   TESTA$    Set proper Z-bit (.EQ. or .NE.)!
08580  MEXIT
08590  ENDC
08600  ENDC
08610  FAIL   Macro syntax error detected!
08620  ENDM
08630

```

```

08640 *****
08650 * LDYR = load YREG
08660 * LDYR [#,<address>
08670 *
08680 * Examples:
08690 * 1. "LDYR #,START" puts the value of symbol 'START' into the
08700 * YREG
08710 * 2. "LDYR START" puts the contents of location 'START' and
08720 * 'START'+1 into the YREG.
08730 *
08740 * Register Usage:
08750 * CC = reflects MS half.
08760 * All other registers preserved.
08770 *
08780 LDYR MACR
08790 IFEQ NARG-1
08800 STA TEMPAS
08810 LDA (\0)
08820 STA YREG1$
08830 STA YREG2$
08840 LDA (\0)+1
08850 STA YREG1$+1
08860 STA YREG2$+1
08870 IFEQ YREG1$!. $FF00
08880 LDA TEMPAS
08890 TST YREG1$
08900 ENDC
08910 IFNE YREG1$!. $FF00
08920 STA TESTAS
08930 LDA TEMPAS
08940 TST TESTAS
08950 ENDC
08960 MEXIT
08970 ENDC
08980 IFEQ NARG-2
08990 IFC '\0', '#'
09000 IFEQ YREG1$!. $FF00 ! YREG in low memory?
09010 IFEQ \1 ! #0 value?
09020 CLR YREG1$+1
09030 CLR YREG2$+1
09040 CLR YREG1$
09050 CLR YREG2$
09060 MEXIT
09070 ENDC
09080 IFNE \1 ! not #0 value?
09090 STA TEMPAS
09100 IFEQ (\1)!. $FF
09110 CLR YREG1$+1
09120 CLR YREG2$+1
09130 ENDC
09140 IFNE (\1)!. $FF
09150 LDA #(1)!. $FF
09160 STA YREG1$+1
09170 STA YREG2$+1
09180 ENDC
09190 IFEQ (\1)!>8
09200 CLR YREG1$
09210 CLR YREG2$
09220 ENDC
09230 IFNE (\1)!>8
09240 LDA #(\1)!>8
09250 STA YREG1$
09260 STA YREG2$
09270 ENDC

```

```

09280         LDA     TEMPAS
09290         TST     YREG1$
09300         MEXIT
09310     ENDC
09320 ENDC
09330     IFNE YREG1$!. $FF00      ! YREG in high memory?
09340     IFEQ  \1                ! #0 value?
09350         STA     TEMPAS
09360         CLRA
09370         STA     YREG1$+1
09380         STA     YREG2$+1
09390         STA     YREG1$
09400         STA     YREG2$
09410         CLR     TESTAS
09420         LDA     TEMPAS
09430         TST     TESTAS
09440         MEXIT
09450     ENDC
09460     IFNE  \1                ! not #0 value?
09470         STA     TEMPAS
09480     IFEQ  (\1)!. $FF
09490         CLRA
09500     ENDC
09510     IFNE  (\1)!. $FF
09520         LDA     #(\1)!. $FF
09530     ENDC
09540         STA     YREG1$+1
09550         STA     YREG2$+1
09560     IFEQ  (\1)!>8
09570         CLRA
09580     ENDC
09590     IFNE  (\1)!>8
09600         LDA     #(\1)!>8
09610     ENDC
09620         STA     YREG1$
09630         STA     YREG2$
09640         STA     TESTAS
09650         LDA     TEMPAS
09660         TST     TESTAS
09670         MEXIT
09680     ENDC
09690 ENDC
09700 ENDC
09710 ENDC
09720     FAIL  Macro syntax error detected!
09730     ENDM
09740
09750 *****
09760 *   STYR = store YREG
09770 *   STYR<address>
09780 *
09790 *   Examples:
09800 *   1. "STYR START"           stores the YREG into locations 'START' and
09810 *                               'START'+1.
09820 *
09830 *   Register Usage:
09840 *   CC = reflects MS half.
09850 *   All other registers preserved.
09860 *
09870 STYR  MACR
09880     STA  TEMPAS
09890     LDA  YREG1$
09900     STA  (\0)
09910     LDA  YREG1$+1
09920     STA  (\0)+1

```

```

09930     IFEQ     YREG1$!. $FF00
09940         LDA     TEMPAS$
09950         TST     YREG1$
09960     ENDC
09970     IFNE     YREG1$!. $FF00
09980         STA     TESTAS$
09990         LDA     TEMPAS$
10000         TST     TESTAS$
10010     ENDC
10020     ENDM
10030
10040 *****
10050 * INCYR = increment YREG
10060 *     INCYR [[#,]<value>]
10070 *
10080 * Examples:
10090 *     1. "INCYR"                adds one (1) to the YREG.
10100 *     2. "INCYR #,START"       adds the value of symbol 'START' to the
10110 *                               YREG.
10120 *     3. "INCYR START"         adds the contents of location 'START' and
10130 *                               'START'+1 to the YREG.
10140 *     4. "INCYR ! comment"     adds one (1) to the YREG (comment present!).
10150 *
10160 * Register Usage:
10170 *     CC = reflects value incremented (Z-bit only).
10180 *     All other registers preserved.
10190 *
10200 * Notes:
10210 *     1. Explicit comment character (!) MUST be used when comment field is
10220 *         present!
10230 *     2. Assumes YREG1$ = YREG2$.
10240 *     3. When parameters are present, this macro becomes "ADD to YREG".
10250 *
10260 INCYR MACR
10270     IFEQ     NARG
10280         IFEQ     YREG1$!. $FF00
10290         INC     YREG1$+1
10300         INC     YREG2$+1
10310         BNE     \.0
10320         INC     YREG1$
10330         INC     YREG2$
10340 \.0     EQU     *
10350     MEXIT
10360     ENDC
10370     IFNE     YREG1$!. $FF00
10380         STA     TEMPAS$
10390         LDA     YREG1$+1
10400         ADD     #1
10410         STA     YREG1$+1
10420         STA     YREG2$+1
10430         LDA     YREG1$
10440         ADC     #0
10450         STA     YREG1$
10460         STA     YREG2$
10470         ORA     YREG1$+1
10480         STA     TESTAS$
10490         LDA     TEMPAS$
10500         TST     TESTAS$
10510     MEXIT
10520     ENDC
10530     ENDC

```

```

10540   IFEQ     NARG-1
10550       STA     TEMPAS
10560       LDA     YREG1$+1
10570       ADD     (\0)+1
10580       STA     YREG1$+1
10590       STA     YREG2$+1
10600       LDA     YREG1$
10610       ADC     \0
10620       STA     YREG1$
10630       STA     YREG2$
10640       ORA     YREG1$+1
10650       STA     TESTAS
10660       LDA     TEMPAS
10670       TST     TESTAS
10680       MEXIT
10690   ENDC
10700   IFEQ     NARG-2
10710       IFC     '\0', '#'
10720       STA     TEMPAS
10730       LDA     YREG1$+1
10740       ADD     #(\1)!.$FF
10750       STA     YREG1$+1
10760       STA     YREG2$+1
10770       LDA     YREG1$
10780       ADC     #(\1)!>8
10790       STA     YREG1$
10800       STA     YREG2$
10810       ORA     XREG1$+1
10820       STA     TESTAS
10830       LDA     TEMPAS
10840       TST     TESTAS
10850       MEXIT
10860   ENDC
10870   ENDC
10880   FAIL     Macro syntax error detected!
10890   ENDM
10900
10910 *****
10920 * DECYN = decrement YREG
10930 *     DECYN    [[#,]<value>]
10940 *
10950 * Examples:
10960 *   1. "DECYN"           subtracts one (1) from the YREG.
10970 *   2. "DECYN #,START"  subtracts the value of symbol 'START' from
10980 *                       the YREG.
10990 *   3. "DECYN START"    subtracts the contents of location 'START'
11000 *                       and 'START'+1 from the YREG.
11010 *   4. "DECYN ! comment" subtracts one from the YREG (comment present!).
11020 *
11030 * Register Usage:
11040 *   CC = reflects value decremented (2-bit only).
11050 *   All other registers preserved.
11060 *
11070 * Notes:
11080 *   1. Explicit comment character (!) MUST be used when comment field is
11090 *      present!
11100 *   2. Assumes YREG1$ = YREG2$.
11110 *   3. When parameters are present, this macro becomes "SUBTRACT from YREG".
11120 *

```

```

11130  DECVR    MACR
11140    IFEQ    NARG
11150        STA    TEMPAS
11160        LDA    YREG1$+1
11170        SUB    #1
11180        STA    YREG1$+1
11190        STA    YREG2$+1
11200        LDA    YREG1$
11210        SBC    #0
11220        STA    YREG1$
11230        STA    YREG2$
11240        ORA    YREG1$+1
11250        STA    TESTAS
11260        LDA    TEMPAS
11270        TST    TESTAS
11280        MEXIT
11290    ENDC
11300    IFEQ    NARG-1
11310        STA    TEMPAS
11320        LDA    YREG1$+1
11330        SUB    (\0)+1
11340        STA    YREG1$+1
11350        STA    YREG2$+1
11360        LDA    YREG1$
11370        SBC    \0
11380        STA    YREG1$
11390        STA    YREG2$
11400        ORA    YREG1$+1
11410        STA    TESTAS
11420        LDA    TEMPAS
11430        TST    TESTAS
11440        MEXIT
11450    ENDC
11460    IFEQ    NARG-2
11470        IFC    '\0', '#'
11480            STA    TEMPAS
11490            LDA    YREG1$+1
11500            SUB    #(\1)!.$FF
11510            STA    YREG1$+1
11520            STA    YREG2$+1
11530            LDA    YREG1$
11540            SBC    #(\1)!>8
11550            STA    YREG1$
11560            STA    YREG2$
11570            ORA    YREG1$+1
11580            STA    TESTAS
11590            LDA    TEMPAS
11600            TST    TESTAS
11610            MEXIT
11620        ENDC
11630    ENDC
11640        FAIL    Macro syntax error detected!
11650        ENDM
11660

```

```

11670 *****
11680 * CPYR = compare YREG
11690 *   CPYR  [#,<address>
11700 *
11710 * Examples:
11720 *   1. "CPYR #,BLOCKSZ"           compares the value of symbol 'BLOCKSZ'
11730 *                                   with the YREG.
11740 *   2. "CPYR START"             compares the contents of location
11750 *                                   'START' and 'START'+1 with the YREG.
11760 *
11770 * Register Usage:
11780 *   CC = reflects YREG comparison (Z-bit only).
11790 *   All other registers preserved.
11800 *
11810 CPYR MACR
11820   IFEQ   NARG-1
11830       STA   TEMPAS
11840       BSET  0,TESTAS   Preset for .NE. condition!
11850       LDA   YREG1$+1
11860       CMP   (\0)+1
11870       BNE   \.0       Branch if LS half is .NE.
11880       LDA   YREG1$
11890       CMP   (\0)
11900       BNE   \.0       Branch if MS half is .NE.
11910       CLR   TESTAS    Set for .EQ. condition!
11920 \.0   LDA   TEMPAS
11930       TST   TESTAS    Set proper Z-bit (.EQ. or .NE.)!
11940       MEXIT
11950   ENDC
11960   IFEQ   NARG-2
11970       IFC   '\0', '#'
11980       IFEQ   \1
11990       IFEQ   YREG1$!. $FF00
12000       TST   YREG1$+1
12010       BNE   \.0       Branch if LS half is .NE.
12020       TST   YREG1$
12030 \.0   EQU   *
12040       MEXIT
12050   ENDC
12060   IFNE   YREG1$!. $FF00
12070       STA   TEMPAS
12080       BSET  0,TESTAS   Preset for .NE. condition!
12090       LDA   YREG1$+1
12100       BNE   \.0       Branch if MS half is .NE.
12110       LDA   YREG1$
12120       BNE   \.0       Branch if MS half is .NE.
12130       CLR   TESTAS    Set for .EQ. condition!
12140 \.0   LDA   TEMPAS
12150       TST   TESTAS    Set proper Z-bit (.EQ. or .NE.)!
12160       MEXIT
12170   ENDC
12180   ENDC
12190       STA   TEMPAS
12200       BSET  0,TESTAS   Preset for .NE. condition!
12210       LDA   YREG1$+1
12220   IFNE   (\1)!. $FF
12230       CMP   #(\1)!. $FF
12240   ENDC
12250       BNE   \.0       Branch if LS half is .NE.
12260       LDA   YREG1$
12270   IFNE   (\1)!>8
12280       CMP   #(\1)!>8
12290   ENDC

```

```

12300         BNE     \.0           Branch if MS half is .NE.
12310         CLR     TESTA$        Set for .EQ. condition!
12320 \.0     LDA     TEMPAS$
12330         TST     TESTAS$        Set proper Z-bit (.EQ. or .NE.!)
12340         MEXIT
12350         ENDC
12360        ENDC
12370         FAIL     Macro syntax error detected!
12380         ENDM
12390
12400 *****
12410 * DEC.B = decrement byte
12420 * DEC.B [[#,]<value>,<address>
12430 *
12440 * where:
12450 * <value> = value to decrement the contents of the <address>
12460 *          location by; immediate ("#", present) or absolute
12470 *          addressing ("#", not present). If only <address> is
12480 *          specified, a default immediate value of one is used.
12490 *
12500 * Examples:
12510 * 1. "DEC.B START"             subtracts one from the contents of
12520 *                               location 'START'.
12530 * 2. "DEC.B #,5,START"        subtracts five from the contents of
12540 *                               location 'START'.
12550 * 3. "DEC.B CNT,START"        subtracts the contents of location 'CNT'
12560 *                               from the contents of location 'START'.
12570 *
12580 * Register Usage:
12590 *   CC = reflects value decremented (N and Z-bits).
12600 *   All other registers preserved.
12610 *
12620 * Notes:
12630 * 1.<address> may be direct or extended!
12640 * 2.This macro essentially performs a "SUB n" function.
12650 *
12660 DEC.B     MACR
12670     IFEQ     NARG-1
12680     IFEQ     (\0)!. $FF00
12690     DEC     \0
12700     MEXIT
12710     ENDC
12720     STA     TEMPAS$
12730     LDA     \0
12740     SUB     #1
12750     STA     \0
12760     STA     TESTAS$
12770     LDA     TEMPAS$
12780     TST     TESTAS$
12790     MEXIT
12800     ENDC
12810     IFEQ     NARG-2
12820     STA     TEMPAS$
12830     LDA     \1
12840     SUB     \0
12850     STA     \1
12860     IFEQ     (\1)!. $FF00
12870     LDA     TEMPAS$
12880     TST     \1
12890     MEXIT
12900     ENDC

```



```

12910      IFNE      (\1)!. $FF00
12920      STA      TESTA$
12930      LDA      TEMPAS
12940      TST      TESTAS
12950      MEXIT
12960      ENDC
12970      ENDC
12980      IFEQ      NARG-3
12990      IFC      '\0', '#'
13000      STA      TEMPAS
13010      LDA      \2
13020      SUB      #\1
13030      STA      \2
13040      IFEQ      (\2)!. $FF00
13050      LDA      TEMPAS
13060      TST      \2
13070      MEXIT
13080      ENDC
13090      IFNE      (\2)!. $FF00
13100      STA      TESTA$
13110      LDA      TEMPAS
13120      TST      TESTAS
13130      MEXIT
13140      ENDC
13150      ENDC
13160      ENDC
13170      FAIL      Macro syntax error detected!
13180      ENDM
13190
13200      *****
13210      * DEC.W = decrement word
13220      *      DEC.W [[#,]<value>,<address>
13230      *
13240      * where:
13250      *      <value> = value to decrement the contents of the <address> and
13260      *                  <address>+1 locations by; immediate ("#," present) or
13270      *                  absolute addressing ("#," not present). If only
13280      *                  <address> is specified, a default immediate value of
13290      *                  one is used.
13300      *
13310      * Examples:
13320      *      1. "DEC.W START"          subtracts one from the contents of loca-
13330      *                                  tions 'START' and 'START'+1.
13340      *      2. "DEC.W CNT, START"    subtracts the contents of locations 'CNT'
13350      *                                  and 'CNT'+1 from the contents of locations
13360      *                                  'START' and 'START'+1.
13370      *      3. "DEC.W #, 5, START"   subtracts five from the contents of loca-
13380      *                                  tions 'START' and 'START'+1.
13390      *
13400      * Register Usage:
13410      *      CC = reflects value incremented (Z-bit only).
13420      *      All other registers preserved.
13430      *
13440      * Notes:
13450      *      1. <address> may be direct or extended!
13460      *      2. This macro essentially performs a "SUB n" function.
13470      *

```

```

13480 DEC.W MACR
13490     IFEQ  NARG-1
13500         STA  TEMPAS
13510         LDA  (\0)+1
13520         SUB  #1
13530         STA  (\0)+1
13540         LDA  \0
13550         SBC  #0
13560         STA  \0
13570         ORA  (\0)+1
13580         STA  TESTAS
13590         LDA  TEMPAS
13600         TST  TESTAS
13610         MEXIT
13620     ENDC
13630     IFEQ  NARG-2
13640         STA  TEMPAS
13650         LDA  (\1)+1
13660         SUB  (\0)+1
13670         STA  (\1)+1
13680         LDA  \1
13690         SBC  \0
13700         STA  \1
13710         ORA  (\1)+1
13720         STA  TESTAS
13730         LDA  TEMPAS
13740         TST  TESTAS
13750         MEXIT
13760     ENDC
13770     IFEQ  NARG-3
13780         IFC  '\0', '#'
13790         STA  TEMPAS
13800         LDA  (\2)+1
13810         SUB  #(\1)!.$FF
13820         STA  (\2)+1
13830         LDA  \2
13840         SBC  #(\1)!>8
13850         STA  \2
13860         ORA  (\2)+1
13870         STA  TESTAS
13880         LDA  TEMPAS
13890         TST  TESTAS
13900         MEXIT
13910     ENDC
13920     ENDC
13930         FAIL  Macro syntax error detected!
13940     ENDM
13950

```

```

13960 *****
13970 * INC.B = increment byte
13980 *   INC.B [[#,<value>,<address>]
13990 *
14000 * where:
14010 *   <value> = value to decrement the contents of the <address>
14020 *             location by; immediate ("#", present) or absolute
14030 *             addressing ("#", not present). If only <address> is
14040 *             specified, a default immediate value of one is used.
14050 *
14060 * Examples:
14070 *   1. "INC.B START"           adds one to the contents of location
14080 *                               'START'.
14090 *   2. "INC.B #,5,START"      adds five to the contents of location
14100 *                               'START'.
14110 *   3. "INC.B CNT,START"      adds the contents of location 'CNT' to
14120 *                               the contents of location 'START'.
14130 *
14140 * Register Usage:
14150 *   CC= reflects value incremented (N and Z-bits).
14160 *   All other registers preserved.
14170 *
14180 * Notes:
14190 *   1. <address> may be direct or extended!
14200 *   2. This macro essentially performs an "ADD n" function.
14210 *
14220 INC.B MACR
14230   IFEQ NARG-1
14240     IFEQ (\0)!. $FF00
14250     INC \0
14260     MEXIT
14270   ENDC
14280     STA TEMPAS
14290     LDA \0
14300     ADD #1
14310     STA \0
14320     STA TESTAS
14330     LDA TEMPAS
14340     TST TESTAS
14350     MEXIT
14360   ENDC
14370   IFEQ NARG-2
14380     STA TEMPAS
14390     LDA \1
14400     ADD \0
14410     STA \1
14420     IFEQ (\1)!. $FF00
14430     LDA TEMPAS
14440     TST \1
14450     MEXIT
14460   ENDC
14470   IFNE (\1)!. $FF00
14480     STA TESTAS
14490     LDA TEMPAS
14500     TST TESTAS
14510     MEXIT
14520   ENDC
14530 ENDC

```

```

14540 IFEQ NARG-3
14550 IFC '\0', '#'
14560 STA TEMPAS
14570 LDA \2
14580 ADD #1
14590 STA \2
14600 IFEQ (\2)!.$FF00
14610 LDA TEMPAS
14620 TST \2
14630 MEXIT
14640 ENDC
14650 IFNE (\2)!.$FF00
14660 STA TESTAS
14670 LDA TEMPAS
14680 TST TESTAS
14690 MEXIT
14700 ENDC
14710 ENDC
14720 ENDC
14730 FAIL Macro syntax error detected!
14740 ENDM
14750
14760 *****
14770 * INC.W = increment word
14780 * INC.W [[#,]<value>,<address>
14790 *
14800 * where:
14810 * <value> = value to increment the contents of the <address> and
14820 * <address>+1 locations by; immediate ("#," present)
14830 * or absolute addressing ("#," not present). If only
14840 * <address> is specified, a default immediate value of
14850 * one is used.
14860 *
14870 * Examples:
14880 * 1. "INC.W START" adds one to the contents of locations
14890 * 'START' and 'START'+1.
14900 * 2. "INC.W CNT,START" adds the value of 'CNT' to the contents
14910 * of locations 'START' and 'START'+1.
14920 * 3. "INC.W #,5,START" adds five to the contents of locations
14930 * 'START' and 'START'+1.
14940 *
14950 * Register Usage:
14960 * CC = reflects value incremented (Z-bit only).
14970 * All other registers preserved.
14980 *
14990 * Notes:
15000 * 1. <address> may be direct or extended!
15010 * 2. This macro essentially performs an "ADD n" function.
15020 *
15030 INC.W MACR
15040 IFEQ NARG-1
15050 IFEQ (\0)!.$FF00
15060 INC (\0)+1
15070 BNE \.0
15080 INC \0
15090 \.0 EQU *
15100 MEXIT
15110 ENDC

```

```

15120     IFNE      (\0)!.$FF00
15130         STA      TEMPA$
15140         LDA      (\0)+1
15150         ADD      #1
15160         STA      (\0)+1
15170         LDA      \0
15180         ADC      #0
15190         STA      \0
15200         ORA      (\0)+1
15210         STA      TESTA$
15220         LDA      TEMPA$
15230         TST      TESTA$
15240         MEXIT
15250     ENDC
15260 ENDC
15270     IFEQ      NARG-2
15280         STA      TEMPA$
15290         LDA      (\1)+1
15300         ADD      (\0)+1
15310         STA      (\1)+1
15320         LDA      \1
15330         ADC      \0
15340         STA      \1
15350         ORA      (\1)+1
15360         STA      TESTA$
15370         LDA      TEMPA$
15380         TST      TESTA$
15390         MEXIT
15400 ENDC
15410     IFEQ      NARG-3
15420         IFC      '\0', '#'
15430         STA      TEMPA$
15440         LDA      (\2)+1
15450         ADD      #(\1)!.$FF
15460         STA      (\2)+1
15470         LDA      \2
15480         ADC      #(\1)!>8
15490         STA      \2
15500         ORA      (\2)+1
15510         STA      TESTA$
15520         LDA      TEMPA$
15530         TST      TESTA$
15540         MEXIT
15550 ENDC
15560 ENDC
15570     FAIL      Macro syntax error detected!
15580     ENDM
15590

```

```

15600 *****
15610 * MOV.B = move byte
15620 *     MOV.B    [# , ]<byte>, <address>
15630 *
15640 * where:
15650 *     <byte>   = byte value to move to the <address> location, using
15660 *                 immediate ("#", " present) or absolute addressing ("#", "
15670 *                 not present).
15680 *
15690 * Examples:
15700 *     1. "MOV.B CNT, TMP"      puts the contents of location 'CNT'
15710 *                               into location 'TMP'.
15720 *     2. "MOV.B #, 5, START"  puts 5 into location 'START'.
15730 *
15740 * Register Usage:
15750 *     CC = reflects value moved.
15760 *     All other registers preserved.
15770 *
15780 MOV.B MACR
15790     IFEQ     NARG-2
15800             STA     TEMPAS
15810             LDA     \0
15820             STA     \1
15830     IFEQ     (\1)!.$FF00
15840             LDA     TEMPAS
15850             TST     \1
15860             MEXIT
15870     ENDC
15880     IFNE     (\1)!.$FF00
15890             IFEQ     (\0)!.$FF00
15900             LDA     TEMPAS
15910             TST     \0
15920             MEXIT
15930     ENDC
15940     IFNE     (\0)!.$FF00
15950             STA     TESTAS
15960             LDA     TEMPAS
15970             TST     TESTAS
15980             MEXIT
15990     ENDC
16000     ENDC
16010     ENDC
16020     IFEQ     NARG-3
16030             IFC     '\0', '#'
16040             IFEQ     (\2)!.$FF00
16050             IFEQ     \1
16060             CLR     \2
16070             MEXIT
16080     ENDC
16090     ENDC
16100             STA     TEMPAS
16110             IFEQ     \1
16120             CLRA
16130     ENDC
16140             IFNE     \1
16150             LDA     #1
16160     ENDC
16170             STA     \2
16180             IFEQ     (\2)!.$FF00
16190             LDA     TEMPAS
16200             TST     \2
16210             MEXIT
16220     ENDC

```

```

16230         IFNE  (\2)!.$FF00
16240         STA   TESTA$
16250         LDA   TEMPAS$
16260         TST   TESTAS$
16270         MEXIT
16280         ENDC
16290         ENDC
16300         ENDC
16310         FAIL   Macro syntax error detected!
16320         ENDM
16330
16340 *****
16350 * MOV.W = move word
16360 *   MOV.W [#,<word>,<address>
16370 *
16380 * where:
16390 *   <word> = word (16-bit) value to move to the <address> and
16400 *           <address>+1 locations, using immediate ("#", " present)
16410 *           or absolute addressing ("#", " not present).
16420 *
16430 * Examples:
16440 *   1. "MOV.W #,5,START"   puts $0005 into location 'START' and
16450 *                          'START'+1.
16460 *   2. "MOV.W #,CNT,TMP"  puts the value of symbol 'CNT' into
16470 *                          locations 'TMP' and 'TMP'+1.
16480 *   3. "MOV.W CNT,TMP"    copies the contents of location 'CNT'
16490 *                          and 'CNT'+1 into locations 'TMP' and
16500 *                          'TMP'+1.
16510 *
16520 * Register Usage:
16530 *   CC = reflects MS half of value moved.
16540 *   All other registers preserved.
16550 *
16560 MOV.W      MACR
16570         IFEQ   NARG-2
16580         STA   TEMPAS$
16590         LDA   (\0)+1
16600         STA   (\1)+1
16610         LDA   \0
16620         STA   \1
16630         IFEQ   (\1)!.$FF00
16640         LDA   TEMPAS$
16650         TST   \1
16660         MEXIT
16670         ENDC
16680         IFNE   (\1)!.$FF00
16690         IFEQ   (\0)!.$FF00
16700         LDA   TEMPAS$
16710         TST   \0
16720         MEXIT
16730         ENDC
16740         IFNE   (\0)!.$FF00
16750         STA   TESTAS$
16760         LDA   TEMPAS$
16770         TST   TESTAS$
16780         MEXIT
16790         ENDC
16800         ENDC
16810         ENDC

```

```

16820  IFEQ  NARG-3
16830  IFC   '\0', '#'
16840  IFEQ  ((\2)+1)!.$FF00
16850  IFEQ  \1
16860  CLR   \2
16870  CLR   \2+1
16880  MEXIT
16890  ENDC
16900  ENDC
16910  STA   TEMPAS
16920  IFEQ  (\1)!.$00FF
16930  CLRA
16940  ENDC
16950  IFNE  (\1)!.$00FF
16960  LDA   #(\1)!.$00FF
16970  ENDC
16980  STA   (\2)+1
16990  IFEQ  (\1)!>8
17000  IFNE  (\1)!.$00FF
17010  CLRA
17020  ENDC
17030  ENDC
17040  IFNE  (\1)!>8
17050  LDA   #(\1)!>8
17060  ENDC
17070  STA   \2
17080  IFEQ  ((\2)!.$FF00
17090  LDA   TEMPAS
17100  TST   \2
17110  MEXIT
17120  ENDC
17130  IFNE  (\2)!.$FF00
17140  STA   TESTAS
17150  LDA   TEMPAS
17160  TST   TESTAS
17170  MEXIT
17180  ENDC
17190  ENDC
17200  ENDC
17210  FAIL  Macro syntax error detected!
17220  ENDM
17230

```



```

17240 *****
17250 * MOVE = move block of memory
17260 *   MOVE  [#],<source>,[#],<destination>,[#],<length>
17270 *
17280 * where:
17290 *   <source>      is the address of the source memory block.
17300 *   <destination> is the address of the destination block.
17310 *   <length>      is the length of the block to move, in bytes.
17320 *                 Maximum of 65,536 bytes can be moved.
17330 *   #             is optional character to denote immediate
17340 *                 addressing for the next parameter
17350 * Examples:
17360 *   1. "MOVE #,ROM,#,RAM,#,CNT moves the block of memory starting at
17370 *       location 'ROM' for 'CNT' bytes, to
17380 *       location 'RAM'.
17390 *   2. "MOVE #,ABC,#,XYZ,,CNT moves the block of memory starting at
17400 *       location 'ABC' for the number of bytes
17410 *       in locations 'CNT' and 'CNT'+1, to
17420 *       location 'XYZ'.
17430 *
17440 * Register Usage:
17450 *   CC = unknown.
17460 *   All other registers preserved.
17470 *
17480 * Subr. used:
17490 *   LDAXREG, STAXREG
17500 *
17510 * Macros used:
17520 *   None, because this macro was written to be as efficient as
17530 *   possible.
17540 *
17550 * Notes:
17560 *   1. If all immediate addressing operands (#) and the move count is
17570 *      <= 256, then a special 'short form' is generated which DOES NOT
17580 *      contain any subroutine calls!
17590 *   2. Depending on the exact parameters passed, not all registers,
17600 *      subroutines and/or macros may be used.
17610 *   3. This macro takes advantage of the fact that there are in fact
17620 *      two XREGs, one for LOAD (XREG1$) and one for STORE (XREG2$).
17630 *   4. The INCKR macro cannot be used here, because it assumes that
17640 *      XREG1$ = XREG2$.
17650 * -----
17660 MOVE  MACR
17670       IFNE  NARG-6
17680         FAIL ** 'move' macro requires six arguments!
17690       ENDC
17700       IFC   '\4','#'          ! If all immediate operands (#) and move
17710         IFC   '\2','#'          ! count <=256, use short form!
17720         IFC   '\0','#'
17730         IFLE 5-256            ! No subr. calls!
17740         STA  TEMPAS
17750         STX  TEMPX$
17760         LDX  #(\5)
17770 \.0   LDA  (\1)-1,x
17780         STA  (\3)-1,x
17790         DEX
17800         BNE  \.0
17810         LDA  TEMPAS
17820         LDX  TEMPX$
17830         MEXIT
17840       ENDC
17850     ENDC
17860   ENDC
17870 ENDC
17880

```

```

17890      STA     TEMPAS
17900      STX     TEMPIX$
17910      LDA     XREG1$
17920      STA     TEMPXR$
17930      LDA     XREG1$+1
17940      STA     TEMPXR$+1
17950      IFC     '\0', '#'           ! immediate type 'from' address?
17960      LDA     #(\1)!. $FF
17970      STA     XREG1$+1           ! Set XREG1$ = 'from' address
17980      LDA     #(\1)!>8
17990      STA     XREG1$
18000      ENDC
18010      IFNC     '\0', '#'           ! not immediate type 'from' address?
18020      LDA     (\1)+1
18030      STA     XREG1$+1           ! Set XREG1$ = 'from' address
18040      LDA     (\1)
18050      STA     XREG1$
18060      ENDC
18070      IFC     '\2', '#'           ! immediate type 'to' address?
18080      LDA     #(\3)!. $FF
18090      STA     XREG2$+1           ! Set XREG2$ = 'to' address
18100      LDA     #(\3)!>8
18110      STA     XREG2$
18120      ENDC
18130      IFNC     '\2', '#'           ! not immediate type 'to' address?
18140      LDA     (\3)+1
18150      STA     XREG2$+1           ! Set XREG2$ = 'to' address
18160      LDA     (\3)
18170      STA     XREG2$
18180      ENDC
18190
18200      IFC     '\4', '#'           ! immediate type length?
18210      IFLE     \5-256             ! yes: 8-bit size= use X reg.
18220      LDX     #(\5)
18230      JSR     LDAXREG
18240      JSR     STAXREG
18250      IFEQ     XREG1$!. $FF00
18260      INC     XREG1$+1
18270      BNE     \.1
18280      INC     XREG1$
18290      INC     XREG2$+1
18300      BNE     \.2
18310      INC     XREG2$
18320      EQU     *
18330      ENDC
18340      IFNE     XREG1$!. $FF00
18350      LDA     XREG1$+1
18360      ADD     #1
18370      STA     XREG1$+1
18380      LDA     XREG1$
18390      ADC     #0
18400      STA     XREG1$
18410      LDA     XREG2$+1
18420      ADD     #1
18430      STA     XREG2$+1
18440      LDA     XREG2$
18450      ADC     #0
18460      STA     XREG2$
18470      ENDC
18480      DEX
18490      BNE     \.0
18500      LDA     TEMPXR$
18510      STA     XREG1$
18520      STA     XREG2$
18530      LDA     TEMPXR$+1

```

```

18540      STA      XREG1$+1
18550      STA      XREG2$+1
18560      LDA      TEMPAS$
18570      LDX      TEMPX$
18580      MEXIT
18590      ENDC
18600      *
18610      IFGT     \5-256      !   no: 16-bit size= use 'length'
18620      LDA      #(\5)!.$00FF
18630      STA      LENGTH$+1
18640      LDA      #(\5)!>8
18650      STA      LENGTH$
18660      \.0      JSR      LDAXREG
18670      JSR      STAXREG
18680      IFEQ     XREG1$!.$FF00
18690      INC      XREG1$+1
18700      BNE      \.1
18710      INC      XREG1$
18720      \.1      INC      XREG2$+1
18730      BNE      \.2
18740      INC      XREG2$
18750      \.2      EQU      *
18760      ENDC
18770      IFNE     XREG1$!.$FF00
18780      LDA      XREG1$+1
18790      ADD      #1
18800      STA      XREG1$+1
18810      LDA      XREG1$
18820      ADC      #0
18830      STA      XREG1$
18840      LDA      XREG2$+1
18850      ADD      #1
18860      STA      XREG2$+1
18870      LDA      XREG2$
18880      ADC      #0
18890      STA      XREG2$
18900      ENDC
18910      LDA      LENGTH$+1
18920      SUB      #1
18930      STA      LENGTH$+1
18940      LDA      LENGTH$
18950      SBC      #0
18960      STA      LENGTH$
18970      ORA      LENGTH$+1
18980      BNE      \.0
18990      LDA      TEMPXR$
19000      STA      XREG1$
19010      STA      XREG2$
19020      LDA      TEMPXR$+1
19030      STA      XREG1$+1
19040      STA      XREG2$+1
19050      LDA      TEMPAS$
19060      LDX      TEMPX$
19070      MEXIT
19080      ENDC
19090      ENDC
19100

```

```

19110 IFNC '\4', '#' ! nonimmediate type length
19120 LDA (\5)!.$00FF
19130 STA LENGTH$+1
19140 LDA (\5)!>8
19150 STA LENGTH$
19160 \.0 JSR LDAXREG
19170 JSR STAXREG
19180 IFEQ XREG1$!.$FF00
19190 INC XREG1$+1
19200 BNE \.1
19210 INC XREG1$
19220 \.1 INC XREG2$+1
19230 BNE \.2
19240 INC XREG2$
19250 \.2 EQU *
19260 ENDC
19270 IFNE XREG1$!.$FF00
19280 LDA XREG1$+1
19290 ADD #1
19300 STA XREG1$+1
19310 LDA XREG1$
19320 ADC #0
19330 STA XREG1$
19340 LDA XREG2$+1
19350 ADD #1
19360 STA XREG2$+1
19370 LDA XREG2$
19380 ADC #0
19390 STA XREG2$
19400 ENDC
19410 LDA LENGTH$+1
19420 SUB #1
19430 STA LENGTH$+1
19440 LDA LENGTH$
19450 SBC #0
19460 STA LENGTH$
19470 ORA LENGTH$+1
19480 BNE \.0
19490 LDA TEMPXR$
19500 STA XREG1$
19510 STA XREG2$
19520 LDA TEMPXR$+1
19530 STA XREG1$+1
19540 STA XREG2$+1
19550 LDA TEMPAS$
19560 LDX TEMPX$
19570 MEXIT
19580 ENDC
19590 FAIL Macro syntax error detected!
19600 ENDM
19610
19620

```

```

19630         OPT      L
19640 RAMSBR$ EQU      *           Start of RAM based subroutines!
19650 *****
19660 ** The following RAM subroutines MUST BE INITIALIZED from ROM upon          **
19670 ** startup (from 'RAMSBR$' for 'RAMSZ$' number of bytes). If changes        **
19680 ** are to be made to the RAM subroutines, make them here. Then copy         **
19690 ** the source below to the ROM area and insert a '.' in front of all        **
19700 ** the labels (leading '.' will be used to denote ROM). This has            **
19710 ** already been done for you in the RAMSBR.INI file. Just include            **
19720 ** this file into your ROM data area and add the following line in          **
19730 ** your RESET routine to initialize the RAM subroutines from the ROM.       **
19740 **                               MOVE #, .RAMSBR, #, RAMSBR, #,                **
19750 ** It is more efficient if the RAM subroutines are placed in DIRECT         **
19760 ** addressing memory, i.e., $0000-$00FF, but it is not required.           **
19770 *****
19780
19790 *-- start of RAM subroutines -----*
19800 *****
19810 * LDAXREG = load A via XREG subr.
19820 *
19830 * Register Usage:
19840 *   CC = reflects value loaded.
19850 *   All other registers preserved.
19860 *
19870 * NOTE:
19880 *   1. Instruction modified code here must be located in RAM!
19890 *
19900 LDAXREG EQU      *
19910         LDA      0-0+$FFFF
19920 XREG1$ EQU      *-2           Pseudo XREG #1
19930         RTS
19940
19950 *****
19960 * STAXREG = store A via XREG subr.
19970 *
19980 * Register Usage:
19990 *   CC = reflects value stored.
20000 *   All other registers preserved.
20010 *
20020 * NOTE:
20030 *   1. Instruction modified code here must be located in RAM!
20040 *
20050 STAXREG EQU      *
20060         STA      0-0+$FFFF
20070 XREG2$ EQU      *-2           Pseudo XREG #2
20080         RTS
20090
20100 *****
20110 * LDAYREG = load A via YREG subr.
20120 *
20130 * Register Usage:
20140 *   CC = reflects value loaded.
20150 *   All other registers preserved.
20160 *
20170 * NOTE:
20180 *   1. Instruction modified code here must be located in RAM!
20190 *
20200 LDAYREG EQU      *
20210         LDA      0-0+$FFFF
20220 YREG1$ EQU      *-2           Pseudo YREG #1
20230         RTS
20240

```

```

20250 *****
20260 * STAYREG = store A via YREG subr.
20270 *
20280 * Register Usage:
20290 *   CC = reflects value stored.
20300 *   All other registers preserved.
20310 *
20320 * NOTE:
20330 *   1. Instruction modified code here must be located in RAM!
20340 *
20350 STAYREG EQU *
20360 STA 0-0+$FFFF
20370 YREG2$ EQU *-2 Pseudo YREG #2
20380 RTS
20390 *-- end of RAM subroutines ----- *
20400
20410 RAMSZ$ EQU *-RAMSBR$ Size of ram subroutines (in bytes).
20420
20430 ORG LO$MEM
20440 * NOTE: TEMPAS$ and TESTAS$ must always be in low memory $0000-00FF.
20450 TEMPAS$ RMB 1 Temporary storage for A accumulator.
20460 TEMPX$ RMB 1 Temporary storage for X register.
20470 TEMPXR$ RMB 2 Temporary storage for XREG register.
20480 TESTAS$ RMB 1 Temporary operand storage for setting CC bits.
20490 LENGTH$ RMB 2 Temporary operand length.
20500
20510 *****

```

Listing 2 — RAMSBR.INI File

```

00010
00020 *****
00030 * ramsbr.ini 1.0
00040 * -----
00050 * Module Name: RAMSBR - RAM Subroutine Initialization
00060 * -----
00070 *
00080 * Description:
00090 * This file contains the initialization code for the RAM subroutine
00100 * area needed to support the MACROS05.MAC file. It MUST be placed in
00110 * the ROM data area and then copied to RAM for proper operation.
00120 * Consult the MACROS05.MAC file for more details.
00130 *
00140 *****
00150 *
00160 * Notes:
00170 * 1. Motorola reserves the right to make changes to this file.
00180 * Although this file has been carefully reviewed and is
00190 * believed to be reliable, Motorola does not assume any
00200 * liability arising out of its use. This code may be freely
00210 * used and/or modified at no cost or obligation by the user.
00220 * 2. The latest version of this file is maintained on the Motorola
00230 * FREEMWARE Bulletin Board, 512/891-FREE (512/891-3733). It operates
00240 * continuously (except for maintenance) at 1200-2400 baud, 8 bits,
00250 * no parity. Sample test files for PASM05 are also included.
00260 * Download the archive file, MACROS05.ARC, to get all the files.
00270 *
00280 *****
00290 * REVISION HISTORY (add new changes to top):
00300 *
00310 * 05/16/90 P.S. Gilmour
00320 * 1. Original entry generated from MACROS05.MAC version 1.0.
00330 *****
00340
00350 .RAMSBR$ EQU * Start of RAM based subroutines!
00360 *****
00370 ** The following RAM subroutines MUST BE INITIALIZED from ROM upon **
00380 ** startup (from 'RAMSBR$' for 'RAMSZ$' number of bytes). If changes **
00390 ** are to be made to the RAM subroutines, make them in the MACROS05.MAC **
00400 ** file and then copy the source here (ROM area) and insert a '.' in **
00410 ** front of all the labels (leading '.' will be used to denote ROM). **
00420 *****
00430
00440 *-- start of RAM subroutines ----- *
00450 *****
00460 * LDAXREG = load A via XREG subr.
00470 *
00480 * Register Usage:
00490 * CC = reflects value loaded.
00500 * All other registers preserved.
00510 *
00520 * NOTE:
00530 * 1. Instruction modified code here must be located in RAM!
00540 *
00550 .LDAXREG EQU *
00560 LDA 0-0+$FFFF
00570 .XREG1$ EQU *-2 Pseudo XREG #1
00580 RTS
00590

```

```

00600 *****
00610 * STA$X = store A via XREG subr.
00620 *
00630 * Register Usage:
00640 *   CC = reflects value stored.
00650 *   All other registers preserved.
00660 *
00670 * NOTE:
00680 *   1. Instruction modified code here must be located in RAM!
00690 *
00700 .STA$X EQU *
00710     STA 0-0+$FFFF
00720 .XREG2$ EQU *-2      Pseudo XREG #2
00730     RTS
00740
00750 *****
00760 * LDAYREG = load A via YREG subr.
00770 *
00780 * Register Usage:
00790 *   CC = reflects value loaded.
00800 *   All other registers preserved.
00810 *
00820 * NOTE:
00830 *   1. Instruction modified code here must be located in RAM!
00840 *
00850 .LDAYREG EQU *
00860     LDA 0-0+$FFFF
00870 .YREG1$ EQU *-2      Pseudo YREG #1
00880     RTS
00890
00900 *****
00910 * STA$Y = store A via YREG subr.
00920 *
00930 * Register Usage:
00940 *   CC = reflects value stored.
00950 *   All other registers preserved.
00960 *
00970 * NOTE:
00980 *   1. Instruction modified code here must be located in RAM!
00990 *
01000 .STA$Y EQU *
01010     STA 0-0+$FFFF
01020 .YREG2$ EQU *-2      Pseudo YREG #2
01030     RTS
01040 *-- end of RAM subroutines ----- *
01050
01060 .RAMSZ$ EQU *-.RAMSBR$ .Size of ram subroutines (in bytes).
01070     FNE RAMSZ$-.RAMSZ$
01080     FAIL Size mismatch between RAM/ROM subroutine areas!
01090     ENDC

```


Selecting the Right Microcontroller Unit

INTRODUCTION

Selecting the proper microcontroller unit (MCU) for your application is one of the critical decisions which control the success or failure of your project. There are numerous criteria to consider when choosing an MCU and this Application Note will enumerate most of them and presents an outline of the thought process guiding this decision. The reader must attach their own grading scale to the selection criteria presented and then evaluate the total to make the correct decision.

PURPOSE

The main goal is to select the least expensive MCU that minimizes the overall cost of the system while still fulfilling the system specification, i.e., performance, reliability, environmental, etc. The overall cost of the system includes everything, such as Engineering Research and Development (R&D), manufacturing (parts and labor), warranty repairs, updates, field service, upward compatibility, ease of use, etc.

SELECTION PROCESS

To start the selection process, the designer must first ask the question, "What does the MCU need to do in my system?" The answer to this one simple question dictates the required MCU features for the system and thus is the controlling agent in the selection process.

The second step is to conduct a search for MCUs which meet all of the system requirements. This usually involves searching the literature, primarily data books, data sheets, and technical trade journals, but also includes peer consultations. These days, recent trade journals seem to contain the most up to date information for the newer MCUs. If the fit is good enough, a single-chip MCU solution has been found, otherwise a second search must be conducted to find an MCU which best fits the requirements with a minimum of extra circuitry, including considerations of cost and board space. Obviously, a single-chip solution is preferred for cost as well as reliability reasons. Of course, if there is a company policy dictating which MCU manufacturer to use, this will narrow your search considerably.

The last step has several parts, all of which attempt to reduce the list of acceptable MCUs to a single choice. These parts include pricing, availability, development tools, manufacturer support, stability, and sole-sourcing. The whole process may need to be iterated several times to arrive at the optimum decision.

SELECTION CRITERIA

The general outline of the main criteria in selecting a microcontroller is listed below, in the order of importance. Each criteria is explained in greater detail later on.

1. Suitability for the application system, i.e., can it be done with a single-chip MCU or at most a few additional chips?
 - A. Does it have the required number of I/O pins/ports, i.e., too few = can't do the job and too many = excessive cost?
 - B. Does it have all the other required peripherals, such as serial I/O, RAM, ROM, A/D, D/A, etc.?
 - C. Does it have other peripherals that are not needed?
 - D. Does the CPU core have the correct throughput, i.e., computing power, to handle the system requirements over the life of the system for the chosen implementation language? Too much is wasteful and too little will never work.
 - E. Is the MCU affordable, i.e., does the project budget allocate enough funds to permit using this MCU? A budgeting quotation from the manufacturer is usually required to answer this question. If the MCU is not affordable for the project, all the other questions become irrelevant and you must start looking for another MCU.
2. Availability?
 - A. Is the device available in sufficient quantities?
 - B. Is the device in production today?
 - C. What about the future?
3. Development support available?
 - A. Assemblers.
 - B. Compilers.
 - C. Debugging tools.
 1. Evaluation Module (EVM).
 2. In-circuit emulators.
 3. Logic analyzer pods
 4. Debug monitors
 5. Source level debug monitors.
 - D. On-line bulletin board service (BBS).
 1. Real time executives.
 2. Application examples.
 3. Bug reports.
 4. Utility software, including "free" assemblers.
 5. Sample source code.

- E. Applications support.
 1. Specific group who does nothing but applications support?
 2. Application engineers, technicians, or marketers?
 3. How knowledgeable are the support personnel? Are they truly interested in helping you with *your* problem?
 4. Telephone and/or FAX support?
- 4. Manufacturer's history, i.e., "track record."
 - A. Demonstrated competence in design.
 - B. Reliability of silicon, i.e., manufacturing excellence.
 - C. On-time delivery performance.
 - D. Years in business.
 - E. Financial report.

SYSTEM REQUIREMENTS

Applying system analysis to the current project will determine the MCU requirements for the system. What peripheral devices are required? Is the application to be bit manipulating or number crunching? Once data is received, how much manipulation is required? Is the system to be driven by interrupt, polled, or human-responses? How many devices/bits (I/O pins) need to be controlled? Among the many possible types of I/O devices to be controlled/monitored are RS-232C terminals, switches, relays, keypads, sensors (temperature, pressure, light, voltage, etc.), audible alarms, visual indicators (LCD displays, LEDs), analog to digital (A/D), and digital to analog (D/A). Is a single or multiple voltage power supply required for the system? What is the power supply tolerance? Is the device characterized for operation at your system supply voltage? Are the voltages to be held to a small fixed percent variation or are they to operate over a wider range? What is the operating current? Is the product to be ac or battery operated? If battery operated, should rechargables be used, and if so, what is the operational time required before recharging and the required time for recharging?

Are there size and weight restrictions or aesthetic considerations such as shape and/or color? Is there anything special about the operating environment, such as military specification, temperature, humidity, atmosphere (explosive, corrosive, particulates, etc.), pressure/altitude? Is the application to be disk-based or ROM-based? Is it a real time application, and if so, are you going to build or purchase a real time kernel program or maybe a public domain version will suffice? Does your schedule contain enough time and personnel to develop your own? What about royalty payments and bug support? Much more investigation is required for real time applications in order to evaluate their special requirements.

GENERAL MCU ATTRIBUTES

MCUs can generally be classified into 8-bit, 16-bit, and 32-bit groups based upon the size of their arithmetic and index register(s), although some designers argue that bus access size determines the 8/16/32-bit architecture. Is a lower-cost 8-bit MCU able to handle the requirements of the system, or is a higher-cost 16-bit or 32-bit MCU required? Can 8-bit software simulation of features found on the 16-bit or 32-bit MCUs permit using the lower-cost 8-bit MCU by sacrificing some code size and speed? For example, can an 8-bit MCU be used with software macros to implement 16-bit accumulator

and indexing operations? The choice of implementation language (high level vs assembler) can greatly affect system throughput, which can then dictate the choice of 8/16/32-bit architectures, but system cost restraints may override this.

Clock speed, or more accurately, bus speed, determines how much processing can be accomplished in a given amount of time by the MCU. Some MCUs have a narrow clock speed range, whereas others can operate down to zero. Sometimes a specific clock frequency is chosen in order to generate another clock required in the system, e.g., for serial baud rates. In general, computational power, power consumption *and* system cost increase with higher clock frequencies. System costs increase with frequency because not only does the MCU cost more, but so do all the support chips required, such as RAMs, ROMs, PLDs, and bus drivers.

Consider also the processing technology of the MPU; N-channel metal-oxide semiconductor (NMOS) vs high-density complementary metal-oxide semiconductor (HCMOS). In HCMOS, signals drive from rail-to-rail, unlike earlier NMOS processors. Since these criteria can significantly affect noise issues in system design, HCMOS processors are usually preferred. Also, HCMOS uses less power and thus generates less heat. The design geometries in HCMOS are smaller which permit denser designs for a given size, and thus allow higher bus speeds. The denser designs also allow lower cost, for more units can be processed on the same sized silicon wafer. For these reasons, most MCUs today are produced using HCMOS technology.

MCU RESOURCES

By definition, all MCUs have on-chip resources to achieve a higher level of integration and reliability at a lower cost. An on-chip resource is a block of circuitry built into the MCU which performs some useful function under control of the MCU. Built-in resources increase reliability because they do not require any external circuitry to be working for the resource to function. They are pre-tested by the manufacturer and conserve board space by integrating the circuitry into the MCU. Some of the more popular on-chip resources are memory devices, timers, system clock/oscillator, and I/O. Memory devices include read/write memory (RAM), read-only memory (ROM), erasable programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM) and electrically erasable memory (EEM). The term EEM actually refers to an engineering development version of an MCU where EEPROM is substituted for the ROM in order to reduce development time. Timers include both real time clocks and periodic interrupt timers. Be sure to consider the range and resolution of the timer as well as any subfunctions, such as timer compare and/or input capture lines. I/O includes serial communication ports, parallel ports (I/O lines), analog-to-digital (A/D) converters, digital-to-analog (D/A) converters, liquid crystal display drivers (LCD), and vacuum fluorescent display drivers (VFD).

Other less common built-in resources are internal/external bus capability, computer operating properly (COP) watchdog system, clock operating properly detection, selectable memory configurations, and system integration module (SIM). The SIM replaces the external "glue" logic usually required to interface to external devices via chip select pins.

On most MCUs with on-chip resources, a configuration register block is included to control these resources. Sometimes the configuration register block itself can be set up to appear at a different location in the memory map. Sometimes a user

and/or factory test register is present, which indicates concern for quality by the manufacturer. With configuration registers also comes the possibility of errant code altering the desired configuration, so check for "lock-out" mechanisms, i.e., before a register can be changed, a bit in another register must first be altered in a certain sequence. Although configuration registers can at first be very confusing and intimidating because of their complexity, they are extremely valuable because of the flexibility they offer at a low cost so that a single MCU can serve many applications.

MCU INSTRUCTION SET

The instruction set and registers of each MCU should be carefully considered, as they play critical roles in the capability of the system. Have software engineers study the indexed addressing modes versus the anticipated needs of your system. Are there any specialty instructions available which could be used in your system, such as multiply, divide, and table lookup/interpolate? Are there any low power modes for battery conservation, such as stop, low power stop, and/or wait? Are there any bit manipulation instructions (bit set, bit clear, bit test, bit change, branch on bit set, branch on bit clear) to allow easier implementation of controller applications? How about bit field instructions?

Be dubious of fancy instructions which seem to do a lot in one instruction. The real measure of performance is how many clock cycles it takes to accomplish the task at hand, not how many instructions were executed. A fair comparison is to code the same routine and compare the *total* number of clock cycles executed and bytes used. Are there any unimplemented instructions in the opcode map and what happens if they are accidentally executed? Does the system handle this gracefully with an exception handler or does the system crash?

MCU INTERRUPTS

Examining the interrupt structure is a necessity when constructing a real time system. How many interrupt lines or levels are there versus how many does your system require? Is there an interrupt level mask? Once an interrupt level is acknowledged, are there individual vectors to the interrupt handler routines or must each possible interrupt source be polled to determine the source of the interrupt? In speed critical applications, such as controlling a printer, the interrupt response time, i.e., the time from the start of the interrupt (worst case phasing relative to the MCU clock) until the first instruction in the appropriate interrupt handler is executed, can be *the* selection criterion in determining the right MCU.

COMPANY ATTRIBUTES

Examine the assets of your own company with a little truthful introspection. Does your company have a significant investment in knowledge/training of existing personnel with a particular MCU manufacturer and in the development tools for those MCUs? Does your company own enough development tools or will you have to buy or rent more? If a new MCU is under consideration, are there development tools available, such as high level language compilers, assembler/linkers, evaluation modules, and debuggers/emulators? Are your present development tools easily expandable for new MCUs? Will additional personnel have to be hired and trained for this project? Can you hire an expert to train/lead the rest of your team? Does your budget permit hiring additional permanent staff and/or

contractors? Is your company satisfied with your current MCU manufacturer's product line and services?

SUPPLIER ATTRIBUTES

The third step is pruning the list of technically acceptable MCUs by examining the MCU manufacturer and supplier, i.e., the companies with which you plan to enter into a long-term relationship for mutual benefit. A supplier can either be the MCU manufacturer itself, or it can be a full-service dealer who is the authorized representative for several manufacturers. A supplier with a broader range of products and a reputation for quality, reliability, service, and on-time deliverability at a fair price can best serve your needs. Additionally, the more products you purchase from one supplier, the more leverage you obtain for pricing, service, and support. Always keep in mind that although your dollar volume may seem high to you, it is always a relative amount to the total business of the supplier. Suppliers who can furnish not only MCUs, but memories (RAM and ROM), discrete devices (transistors, diodes, etc.), standard digital logic devices (7400, 74HC00, etc.), specialty chips, customer specific devices (CSIC), application specific devices (ASIC), and programmable logic devices (PLDs), will be better suited to serve your growing needs. Has the manufacturer and/or supplier won any awards for quality, reliability, service, and/or deliverability? Be suspicious of self-bestowed awards.

MANUFACTURER ATTRIBUTES

Other criteria to consider in selecting the MCU manufacturer/supplier are stability, sole-supplier status, literature, and support. Stability can best be ascertained by considering the number of years in business and obtaining a Dunn & Bradstreet rating plus copies of past Annual and Quarterly Financial Reports. Your company's Purchasing and Credit Departments can greatly assist you in these areas. Listing on a major stock exchange is another sign of stability. A local stock broker can assist you in obtaining up-to-date information for those manufacturers listed on stock exchanges, or you can visit your local library to check the Periodical Guide for pertinent information. The Wall Street Journal is another excellent source of up-to-date financial news. Sole-supplier status is unfortunately usually the norm, as most MCU manufacturers do not often cross license their products to other manufacturers. If the manufacturer has a good track record for supply, delivery, and pricing, sole-supplier status should not be a problem.

MANUFACTURER SUPPORT

Direct manufacturer support includes Marketing/Sales, Field Application Engineers (FAEs), and Application Engineering. Are the FAEs near your site? When telephoning for support, can you reach the support person directly or do you play "telephone tag"? Are calls returned promptly? Is there a toll-free 800 number? Is there a FAX number? How many phone lines are available? Are the phone lines always busy? Do they have an individual Voice Mail answering system or does a secretary in another office take "While You Were Out" messages which must be physically relayed to the support person? Voice Mail is a state of the art computer controlled answering system whereby each user effectively has their own password protected answering machine with enhanced capabilities, such as message forwarding. What hours do the support personnel work? Do they have other duties and/or responsibilities besides support? How many support personnel are

there? Are factory personnel, such as Product Engineers, Manufacturing Engineers, Quality Engineers, Hardware Engineers, and Software Engineers, readily available to assist the support personnel? Are the factory people on-site with the support personnel? Are the support personnel knowledgeable, have a helpful attitude, and do they "follow through" in a timely manner when they promise to do something, such as research your problem or send you something? Does it come via regular mail, UPS, or Overnight Express? Were you charged for fast delivery?

Does the Manufacturer have an Electronic Bulletin Board Service (BBS) where information such as application programs, product news, software updates, source code, bug lists, electronic mail, and conferencing are available? What baud rates are supported? How many phone lines are available? What are the hours of operation? Do you need any special brand of computer and/or modem to access it? Is there a system operator (sysop) assigned to manage it?

LITERATURE SUPPORT

Literature covers a wide selection of printed material which can assist you in the selection process. This includes items from the Manufacturer, such as Data Sheets, Data Books, and Application Notes, as well as items available at the local bookstore and/or library. Items from the local bookstore and/or library indicate not only the popularity of the manufacturers/MCUs under consideration, but also offer unbiased opinions when written by non-manufacturer related authors.

FINALIZING THE SELECTION

As a final step to help in the selection process, build a table listing each MCU under consideration on one axis and the important attributes on the other axis. Then fill in the blanks from the manufacturer's data sheets in order to obtain a fair side-by-side comparison. Some manufacturers have pre-made comparison sheets of their MCU product line which makes this task much easier, but as with all data sheets, be sure they are up to date with current production units. Among the possible attributes are price (for the anticipated production volume, including predictions of future pricing, i.e., will the price be decreasing as you move into production?), RAM, ROM, EPROM, EEPROM, timer(s), A/D, D/A, serial ports, parallel ports (I/O control lines), bus speed (minimum/maximum), special instructions (multiply, divide, etc.), number of available interrupts, interrupt response time (time from start of interrupt to execution of the first interrupt handler instruction), package size/type (ceramic DIP or LCC, plastic 0.3" DIP or 0.6" DIP, shrink DIP (.071" pin spacing), PLCC, PQFP, EIAJ QFP, SOIC; some involve surface mount technology), power supply requirements, and any other items important to your system design. The tables at the end of this application note detail the attributes of Motorola's MCU product line.

If after all this, you still have more than one MCU on your list, consider expandability and value. What expansions in the

system requirements can you predict that will be needed in possible future iterations of this product? And lastly, consider value, for if two MCUs cost the same but one offers a few more features which are not required today but would make future expansion easier for no additional cost, chose that MCU.

CUSTOM MCU SOLUTION

If there is no commercially available single-chip MCU that meets your system requirements and your anticipated production volume is high enough, you should consider using a custom CSIC MCU. In a custom CSIC MCU, you choose the core processor type and the exact peripherals needed for your system from a list of standard cells available. This gives you the benefits of a single-chip solution for slightly more cost, so the production volume must be high enough to justify it. Additionally, some manufacturers will not even start production unless the order volume is around one million units. However, if your production volume can be combined with others to reach the one million level, production could be started. Or, if the desired unit is judged to be have a broad enough market appeal, the manufacturer may proceed with production anyway because they plan to offer it as a standard product. As the design initiator you may be able to obtain an exclusivity clause whereby you have sole rights to the CSIC MCU for a specific period of time. Then the manufacturer can start marketing it to everyone.

TEAMWORK

Finally, as project leader you can do all this investigative work yourself, or you can start involving your team by assigning investigative tasks to them, such as having the software engineers evaluate the instruction sets of each MPU under consideration. By involving your team early in the decision process, you not only build team spirit, but gain individual commitment to the project via active participation. This approach undoubtedly generates some conflict, as everyone has their own opinion, but your job as project leader is to be a mediator. After listening to all opinions, it is still your choice as project leader. As in political elections, once the winner of the primary has emerged, all party members are expected to fully support the winner, and so should all project team members support the decisions of the project leader in order to ensure a successful project.

CONCLUSION

In conclusion, selecting the right MCU for your project is not an easy decision, as MCUs have become more complex devices since on-chip resources were added. And since the trend is towards more on-chip integration of off-chip resources to reduce system costs, the decision will become increasingly complex with time. This application note is not intended to make the choice for the designer, but to serve as a thought provoking guideline as to all the possible selection criteria that should be considered in this important decision process.

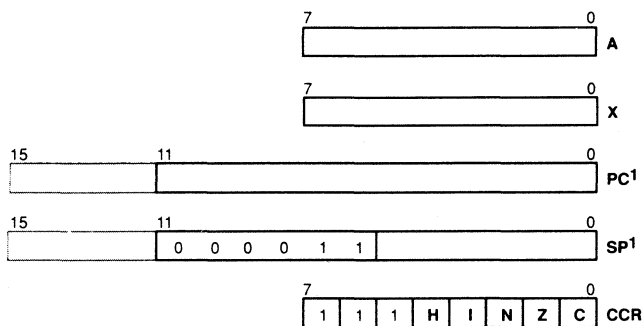
MOTOROLA MICROCONTROLLER COMPARISON GUIDE

Motorola maintains a comparison guide on our most popular M68HC05, M68HC11, and M68300 Families of microcontrollers (MCUs). For a copy, please contact the Motorola Literature Distribution Center nearest you and request the current *MICROCONTROLLER (MCU) QUARTERLY UPDATE FOLDER*, Motorola part number SG148/D.

MOTOROLA 8-BIT M68HC05 MICROCONTROLLER FAMILY

Programming Model

The M68HC05 Family of MCUs has five central processing unit (CPU) registers available to the programmer as shown in Figure 1 below.



1. Length depends on the address/memory size of the individual M68HC05, e.g., a 2K memory map requires an 11-bit program counter and stack pointer as shown. Current length ranges from 11 to 14 bits (2K to 16K).

Figure 1. M68HC05 Programming Model

MOTOROLA 8-BIT M68HC11 MICROCONTROLLER FAMILY

Programming Model

The M68HC11 Family of MCUs has eight central processing unit (CPU) registers available to the programmer as shown in Figure 2 below.

MOTOROLA 16-/32-BIT M68300 MICROCONTROLLER FAMILY

Programming Model

The M68300 Family of MCUs has 23 central processing unit (CPU) registers available to the programmer as shown in Figures 3 and 4 below, organized into User and Supervisor models.

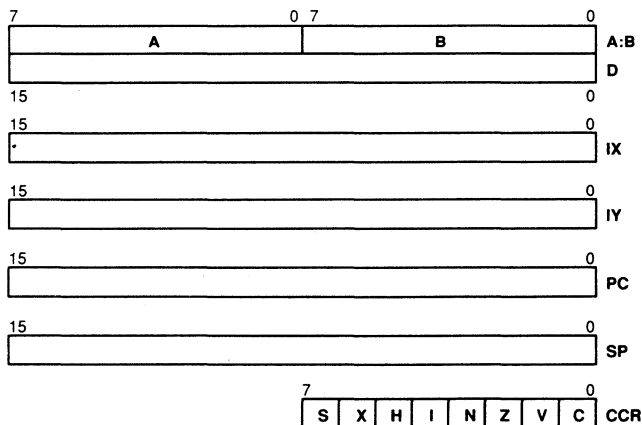


Figure 2. M68HC11 Programming Model

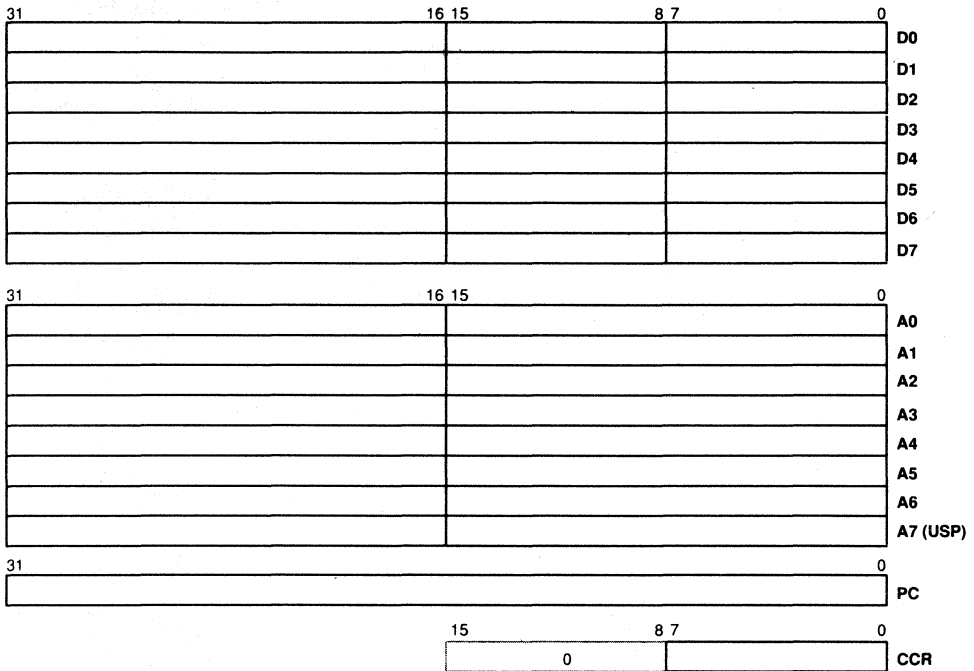
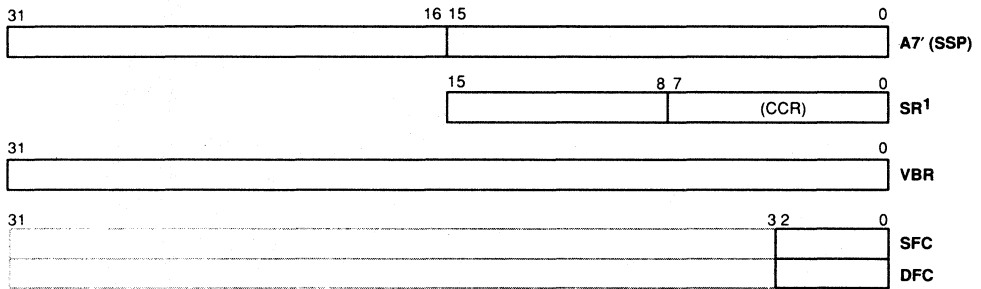


Figure 3. M68300 User Programming Model



1. The Status Register (SR) consists of two halves as shown below.

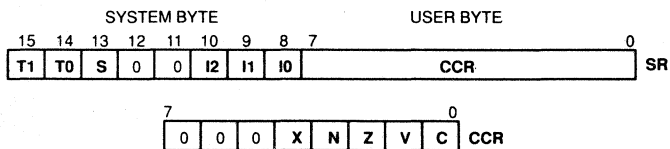


Figure 4. M68300 Supervisor Programming Model Supplement

DEVELOPMENT SUPPORT

Development support for Motorola MCUs is available from Motorola in the form of low cost Educational Computer Boards (ECBs), Evaluation Boards (EVBs), and Evaluation Modules (EVMs), and higher priced emulator systems with bus analysis (CDS Jewelbox and HDS-300 systems). EVBs and EVMs are on the order of hundreds of dollars, whereas emulators are on the order of thousands of dollars. Simple assemblers are available for no charge on the Freeware BBS, whereas more powerful assemblers with such features as macros, structured assembly, conditional assembly, and relocatable modules (for linking) are available for a few hundred dollars for popular PCs. Additionally, the Freeware BBS contains the most up to date information on Motorola MCUs and has a wealth of free software. Also available are literature and Application Notes on many subjects, including a title index, from Motorola's Literature Distribution Centers.

MOTOROLA FREWARE ELECTRONIC BULLETIN BOARD SYSTEM

"Freeware" is the name of the electronic bulletin board system (BBS) dedicated to support Motorola Microprocessor Units (MPUs) and Microcontroller Units (MCUs). The Freeware BBS contains the most up to date information, including support software for EVMs, PCs, and MACs, development software for MCUs and MPUs, confidential electronic mail service, file downloads/uploads, distributor directory and sales offices by state, press news, development support, literature, mask set erratas, devices/packages being phased out, ECB/EVB/EVM product literature, and contest/promotion/seminar information.

Freeware is on-line 24 hours a day, everyday except for maintenance. To use the BBS, you need a 300-2400 baud modem, and a terminal or personal computer (PC) with communications software (e.g., Kermit, ProComm, etc.). Set your character format to 8-bit, no parity, 1 stop bit and dial the Freeware number (512) 891-FREE (891-3733). Press RETURN and then enter the requested information to log on. You are now a registered user. Follow the menus for the desired functions (e.g., download, upload, mail, conferences, etc.). On-line help is available.

For most up to date information, please contact the Motorola Literature Distribution Center nearest you and request the *MCU FREWARE* brochure, Motorola part number BR568/D.

APPLICATIONS SUPPORT

Application support for Motorola products is provided by the same local salesperson and Field Application Engineer (FAE) that made the sale. If a technical question can not be answered locally by the salesman or FAE, the FAE will contact the

Factory Applications Support Group which has a highly trained staff of engineers to relentlessly pursue the answers in both hardware and software. Additionally, the Aps Group has the backup support of the entire on-site Factory staff to ensure answers to your questions.

MOTOROLA LITERATURE DISTRIBUTION CENTERS

Motorola has several Literature Distribution Centers (LDCs) worldwide with the main one located in Phoenix, Arizona, U.S.A. They carry all Motorola literature, including Data Books, Data Reference Manuals, User Manuals, Application Notes, brochures, books on Motorola products, etc. Contact your local Motorola representative or the LDC office nearest you and they will be most happy to serve you. The LDC addresses are listed on the back cover of this application note. The phone numbers are listed below or consult your local telephone directory book.

Motorola Literature Distribution Center
Phone: (602) 994-6561

Motorola Semiconductors H.K. Ltd.
Phone: 480-8333

Motorola Ltd.; European Literature Center
Phone: (908) 61 46 14

Nippon Motorola Ltd.
Phone: 03-440-3311

ABOUT MOTOROLA

Motorola was founded in 1928 by Paul V. Galvin and has continuously been in the electronics business since then. Motorola moved up to number 48 on the 1989 Fortune 500 list of the largest U.S. industrial corporations with \$9.62 billion in sales and is the #3 in electronics in the U.S., behind General Electric and Westinghouse Electric. In terms of semiconductor market share, Motorola is #1 in North America and #4 in the world. Motorola's Microcontroller Division is the world's #1 supplier of 8-bit MCUs, with more than 500 million in use around the globe. Motorola was ranked #3 by Fortune Magazine's poll of *America's Most Admired Corporations*.

Motorola has consistently demonstrated product and technology leadership along with the global capability and teamwork to get the job done for the lowest possible price. Motorola is driven by Total Customer Satisfaction, which means global design and manufacturing facilities, on-time delivery, and above all, a dedication to quality that is unsurpassed in the industry. Motorola's relentless pursuit of perfection is the key in achieving Six Sigma Quality by 1992. This translates to virtual perfection (3.4 defects per million parts). There is only one goal: zero defects in everything Motorola does.

RECENT AWARDS PRESENTED TO MOTOROLA

1990 *GM Mark of Excellence Award*

This award was presented by Delco Electronics for outstanding performance as a supplier in all five areas covered by GM's supplier monitoring program — quality, cost, delivery, technology and management. Currently, less than 30 suppliers have earned the GM Mark of Excellence award.

1990 *Computer Design's Vendor Preference Survey* reported Motorola Microcontrollers as #1 in the three measured categories of Response Time, Documentation, and Applications Support.

1989 Texas Instruments Information Technology Group *Supplier Excellence Award*

Presented to less than one percent of TI's suppliers worldwide for quality and on-time delivery performance.

1989 Chrysler Motor Company *Pentastar Award*

Presented to select group of Chrysler's suppliers worldwide for quality, price, delivery, and technology.

1989 Ford Motor Company *Q1 Preferred Quality Award*

A total quality supplier award; requires consistent high quality coupled with excellent pricing and delivery.

1988 *Malcolm Baldrige National Quality Award*

This award was created by Congress in 1987, named after the late Malcolm Baldrige, Secretary of Commerce during

the Reagan Administration. It is awarded to companies that demonstrate superior company-wide management of quality processes. A panel of judges examines the quality standards in eight critical business areas for each company applying for the award. Six prizes are offered each year — two each for manufacturing companies and service companies, and two for small businesses. In the two years the award has been offered, only five companies out of a potential 12 winners have met the rigid standards required to capture the prize. In 1988, Motorola was one of only three Baldrige winners out of 66 applicants. Winners of the Malcolm Baldrige National Quality Award must wait five years before they can apply again. Therefore, Motorola will be eligible to compete again in 1993.

1988 Dataquest *Supplier of the Year*

1988 Texas Instruments Data Systems Group *Supplier Excellence Award*

Presented for demonstrated excellence in meeting Texas Instrument's requirements.

1987/1988 Bosch Group *Recognition Award*

Presented for quality and special performance as a supplier to the Bosch Group.

1987 Delco Electronics *Award of Excellence*

Presented for superior quality and delivery performance.

Reducing A/D Errors in Microcontroller Applications

Many significant benefits can be realized in an electronic product by converting analog signals into the digital domain. From drift-free signal filtering to extremely reliable signal detection, the digital domain offers a level of performance many times only approximated by its analog circuit-based counterpart. Once cost prohibitive, converting analog signals into the digital domain has become more cost effective. These decreasing costs, increasing digital semiconductor speeds, and the benefits of digital processing have contributed significantly to the increasing popularity of digital systems and to the rise of the digital system with built-in analog interfaces. One such popular system on silicon is the single-chip microcontroller unit (MCU). Now available from many manufacturers and in many forms, MCUs with resident analog interfaces like analog-to-digital converters (ADCs) and other on-chip peripherals can provide unsurpassed cost effectiveness to a product's design. The MCU with integral ADC may easily be used to convert analog signals in the digital domain with the convenience of an already defined on-chip ADC-to-CPU interface. In addition, the MCU offers the flexibility afforded to all software-based systems.

MCUs have liberated many board-level designers from selecting, designing, and debugging microprocessor peripherals in multichip assemblies. This type of highly integrated solution is becoming more popular than the multichip solution. Consequently, it is reasonable to expect that the practicing design engineer will eventually work with an MCU-based system. Yet, despite the advantages of the MCU system, some integrated peripherals such as the ADC offer new challenges to the designer. By incorporating a wide bandwidth linear system, such as an ADC, on the same die with a high-speed digital central processor unit (CPU), ADC performance can be adversely affected. Noisy ADC readings functionally manifest themselves in a range from merely annoying and relatively benign glitches to more catastrophic hard failures. In any case, an MCU-based system does not have to be at the mercy of poor MCU/ADC performance. Fortunately, by following some fairly rudimentary systems-level guidelines in the design phase of the MCU-based product, potential ADC performance problems can be avoided.

To resolve ADC performance issues, it is necessary to understand a little about the nature of the MCU and the various areas of susceptibility of several ADC types. Although much information presented in this application note assumes that the ADC is resident on-chip with the CPU, other converter types not typically found on-chip with MCUs are discussed for those instances in which a multichip combination is encountered. The following paragraphs also apply to these less frequent hardware combinations.

ADC TYPES

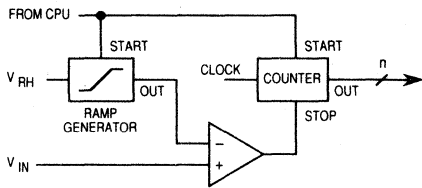
Even when the ADC is available on-chip with an MCU where the unpleasant task of interfacing and debugging the ADC-to-CPU interface is done, obtaining maximum performance from the ADC requires attention to application details of the given MCU/ADC combination. The type of design precautions and applications details needed to avoid problems varies as a function of the type of ADC used. Understanding the mechanics of the given ADC is crucial to improving performance.

ADCs may be categorized into five main categories: integrating, servo, flash, successive approximation, and hybrid. Although each type has unique capabilities and traits, each has surprisingly similar points of vulnerability.

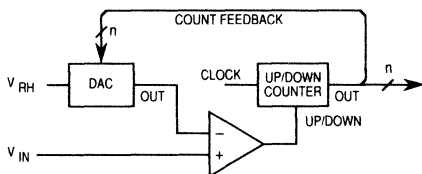
The integrating converter has appeal for applications requiring high resolution (16-bit or higher) and low cost. Because the basic converter is simply implemented (see Figure 1(a)), hardware is minimized while high resolution is obtained. In addition, the integrating ADC may provide some noise immunity that is not feasible with higher speed designs. Although it is possible to build the integrating converter onto MCU chips (there is nothing technologically impeding such a construct), its lower speed has apparently been discouraged by MCU designers since it is currently not offered on an MCU by a major manufacturer.

Whereas the integrating converter tends to be the slowest of ADC types, the servo converter tends to have the highest resolution and fastest conversion times in its most recent advancement—the sigma-delta converter. The more traditional servo converter tries to balance the charge or voltage on an input comparator by using a feedback configuration (see Figure 1(b)) to force slewing from a previous charge or voltage to the current input signal level applied to the other input of the same comparator. This process is followed by appropriately changing a digital counter up or down (in this form, the converter is often called a tracking converter). Before the sigma-delta variation, the servo converter was less popular than other converter types primarily due to its slew-rate limitation. Nothing about the servo converter would prevent its inclusion onto an MCU die; once BiCMOS processes improve, this converter type will probably become a popular feature of future MCUs (particularly the sigma-delta variation, most of which is linear circuitry).

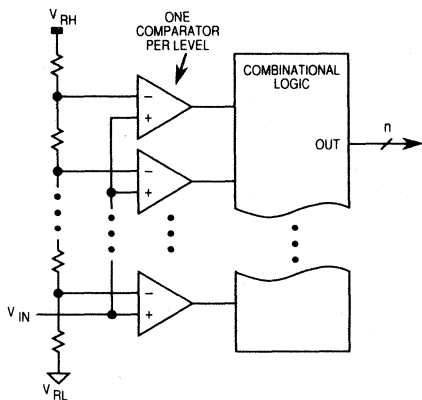
Although the sigma-delta variation of the servo converter provides high-resolution conversions and maintains a relatively high throughput rate, the fastest type is the flash converter. By stringing together several voltage comparators (one per desired level to be detected), conversion bandwidths in excess of 100 MHz are now quite commonplace (see Figure 1(c)), albeit at lower resolutions (4–6 bits are common). As the input voltage is applied to one input of all the comparators, a set of



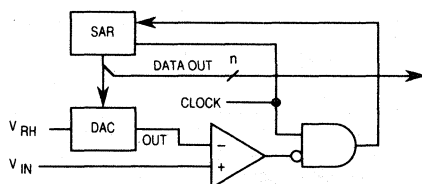
(a) Integrating ADC



(b) Tracking/Servo ADC



(c) Flash ADC



(d) Successive Approximation ADC

Figure 1. ADC Types

reference voltages are applied to the other comparator inputs. After a period of time has elapsed, determined primarily by the propagation delays through the comparators, the discrete-level representation of the input voltage is available at the comparator outputs. Flash conversion, although incredibly fast, requires a tremendous number of devices to implement even modest-resolution converters. In addition to the number of transistors necessary to implement each comparator, the outputs of each comparator are typically input to a combinational logic array to form a desired output code. Consequently, this converter, which consumes much silicon area when compared to other converter types, has not been widely accepted by MCU designers and users.

The fourth ADC type is the successive-approximation converter (SAC). Of all current converter types, SAC is the most popular (see Figure 1(d)). This popularity is primarily due to its applicability to smaller circuit requirements, medium to fast conversion speeds, and medium- to high-resolution applications (8–16 bits). Like other converter types, the SAC uses a differential voltage comparator to compare the input signal with a reference voltage. By performing a binary search, conversion rates of one bit per clock are possible. Because only one comparator is typically required and the output code is inherent in the conversion process, circuitry and silicon surface area are reduced when compared with other conversion methods. Although the exact implementation varies from silicon-chromium-based to charge-redistribution, this ADC is currently the most prevalent type found on MCUs.

The fifth ADC category is the hybrid converter. In this case, the term "hybrid" is not used to reference a specific implementation approach, but rather implies combining one or more ADC types to form an ADC with different performance characteristics. For example, some of the faster and higher resolution ADCs now employ a hybridized technique which utilizes flash-conversion prescaling followed by an SAC. In this case, almost instantaneous prescaling is accomplished and easily interfaced to an existing SAC design. Hybrid converters are a very viable alternative as an MCU peripheral and may find eventual popularity in MCU designs when higher resolution converters are needed.

ADC NOISE SUSCEPTIBILITY

The comparator is the cornerstone of the A/D conversion process. The ability of the comparator to announce the presence of small voltage/current differentials with large changes in its output voltage make the comparator invaluable to the A/D conversion process. Yet, this same feature also accounts for the largest potential source of ADC malfunction. Of course, degradation of the comparator's desired action, and hence the ADC, is most usually caused by unwanted noise. Two basic characteristics of the comparator affect noise susceptibility: bandwidth and power supply connections.

Wide bandwidth comparators easily respond to noise as well as to signals. Even in the low-speed integrating converter, the accuracy of measurement is heavily contingent upon the comparator's speed of operation. To illustrate, imagine that a

very slowly varying input signal has been applied to an input of such a comparator. For a single-slope integrating converter, the other comparator input will have a linearly increasing voltage (or other convenient shape) applied to it. As this voltage ramp increases, an independent digital counter (started at the same time the voltage ramp began) will count clock pulses provided by some timebase. When the voltage ramp finally exceeds the input voltage, the comparator will change state. If the comparator fails to respond to the voltage ramp in a timely fashion, the digital counter will register an incorrect count when compared to the results obtained by a perfectly fast comparator, implying that the response time (characterized by bandwidth) must be reasonably fast even in the slowest ADC types. Consequently, a wide bandwidth comparator will appropriately respond not only to input/reference signals but also to any other signal present at the comparator input terminals (including noise components superimposed upon the signals of interest).

The typical comparator uses some form of differential front end. The operation of the differential front end is dependent upon biasing networks that are ultimately connected to the supply terminals of the comparator. Therefore, the comparator should be considered as a five-terminal device — two differential inputs, one output, and two inputs to the biasing networks — for the purposes of designing with the ADC. The implication is that signals present at the supply terminals of an ADC, particularly the high-frequency signals typically superimposed on the power supply in digital systems, can affect comparator and ADC operation.

Due to the high bandwidth of the comparator found in ADCs, the designer of a given system should be extremely careful about the type and amount of signals allowed to reach the comparator stage of the ADC, particularly the power supply terminals. For this reason, some of the more mundane and overlooked aspects of electrical product design, such as printed circuit design and circuit interconnection, become increasingly critical to the success of the MCU/ADC system.

APPROPRIATE DESIGN TECHNIQUES

Most of the MCU is digital. As seen in Figure 2, a major portion of the MC68HC11E9, a representative MCU, is digital circuitry. Thus, it is reasonable to assume that digital design practices will generally be employed when designing with the MCU. With an analog-based subsystem, such as the M68HC11 ADC, normally accepted digital design practices may not be sufficient to ensure satisfactory performance of the converter. As an illustration, consider noise levels normally found on the power supply of a typical high-speed HCMOS digital system. It is not unusual to find 100 mV_{pp} broadband noise riding on top of the positive voltage rail. With a nominal 5 V HCMOS system, the resulting voltage drop, down to 4.9 V, is above the V_{OH} for HCMOS. Thus, the 100 mV signal will probably not upset circuit operation. When present in such a robust digital system (HCMOS), this 100 mV noise signal is a mere visual nuisance on the oscilloscope. Because of the theoretically infinite signal-to-noise ratio of digital gates, the presence of the 100 mV noise poses no practical threat.

However, when such a noise signal is inserted into an ADC system, the results can be much more dramatic. In an 8-bit ADC system with a nominal 5 V reference, this same 100 mV noise can result in a greater than 5-bit error in the ADC reading. Thus, an MCU system utilizing an ADC assumes a different electrical character that requires application of design practices not traditionally used in the design of digital systems.

What design practices should be used? To correct or avoid a noisy ADC/MCU design, separate the noisy signals from the sensitive ones. The challenge is to design a system in which this separation is practically realized. The closer to the ideal of completely separating the noisy signals from the sensitive ones, the better. For situations where the noisy and sensitive circuits cannot be completely separated, reduce the noise coupling as much as possible. Since it is difficult to axiomatically specify how to implement both concepts in all cases, an illustration will aid understanding and provide an analogy by which individual situations may be gauged.

Motorola tests 100% of the ADCs found on their MCUs. Before any M68HC11 ADCs leave the factory, they have been tested and verified for specified ADC performance. Even so, it is possible to operate the M68HC11 in an environment that causes the M68HC11 ADC to subsequently malfunction. These two scenarios in the life of such an MCU indicate not, strictly speaking, a parts-related anomaly, but rather a significant interaction of the part's characteristics with the electrical environment.

Typically, a large contributor to malfunction is the printed circuit board (PCB) layout. Since the PCB can influence many of the circuit parametrics (reactance, voltage, etc.), the PCB layout can help or hinder ADC performance. Yet, the PCB layout is not typically done by the circuit's designer. More importantly, laying out the PCB artwork, up to and including the width and placement of traces, is often performed by people without a detailed knowledge of correct electrical circuit design practices. Many PCB designers are only concerned with ensuring that they have connected all the points connected in the schematic. Although this has its economic advantages, this can be a dangerous proposition with regard to ADC performance. Figure 3 shows an example of such a PCB layout which, although it manages to distribute the power to all of the devices, provides several potential sources of ADC/MCU performance problems.

First, the MCU/ADC is placed farthest from the power terminal, meaning that the MCU return currents will be mixed with the digital circuit currents between the MCU and the power terminal. Although the MCU may not produce large return currents in the power return, high-speed digital circuits typically do. The inductance of PCB traces at high frequencies can be significant enough to produce large noise spikes when measured between the ground pin of the MCU and the ground terminal of the board.

Second, the opamp, which buffers the signals to the ADC inputs on the MCU, is physically located close to the MCU but is electrically located in a very poor place. As with the MCU, the opamp power supply return will be corrupted with high-frequency spikes. However, the voltage drops measured

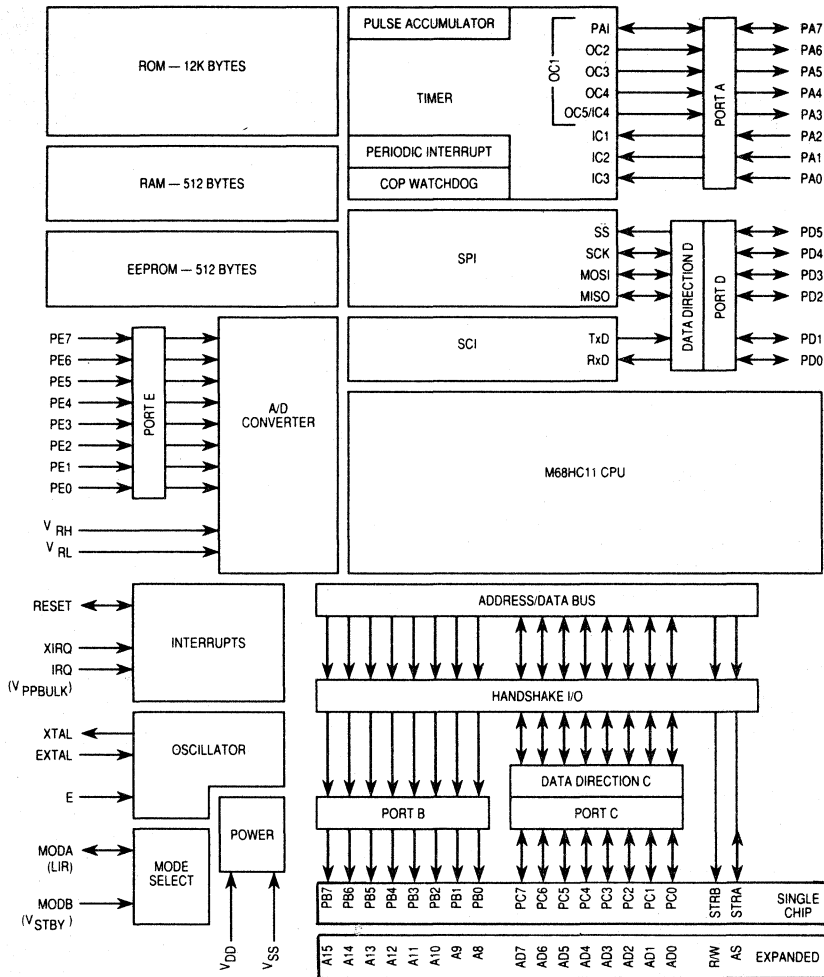


Figure 2. MC68HC11E9 Block Diagram

between the opamp and the MCU will be even worse than those measured between the MCU and power terminal. When deciding which parts are to be jointly located on the PCB, the electrical impact of conductor distance and tolerance to any induced noise must be considered.

Third, the bypass capacitors, as shown, are ineffectual in reducing high-frequency noise on V_{DD} . To perform the decoupling function properly, bypass capacitors should be attached

as close as possible to the IC power pins they are intended to bypass. In addition, PCB trace inductance should be minimized between the leads of the capacitor and the power pins.

Figure 3 illustrates a few of the PCB-related errors that can degrade ADC performance. Specific PCB designs involving MCU/ADCs should be carefully engineered. A better PCB layout is depicted in Figure 4, which corrects the defects shown in Figure 3.

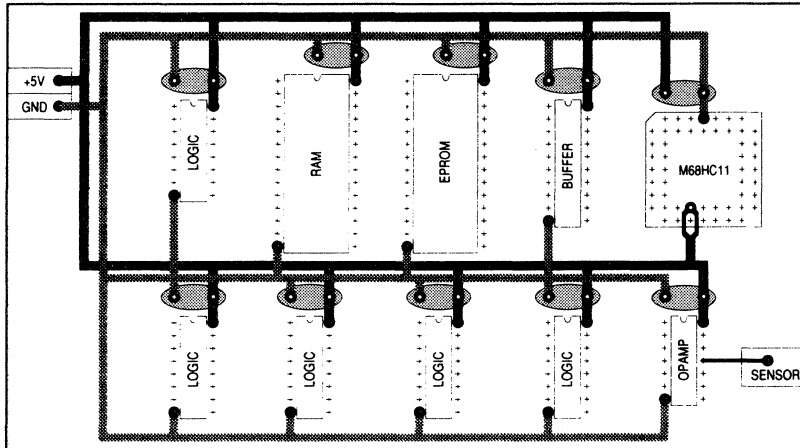


Figure 3. Poor Layout Design

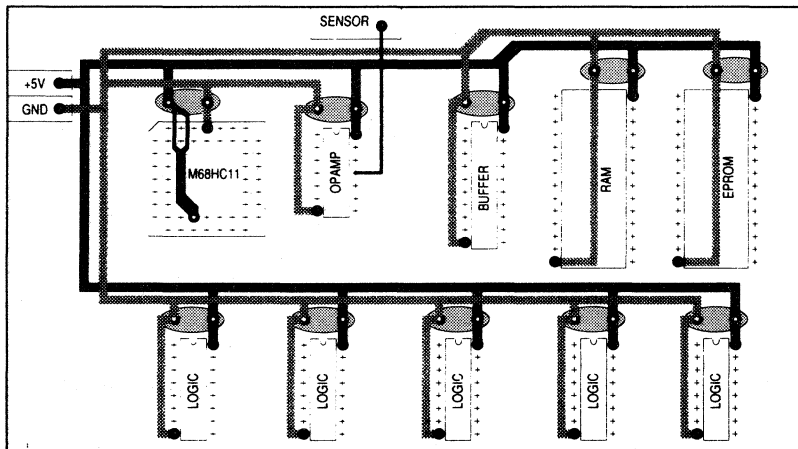


Figure 4. Improved Layout Design

A SPECIFIC MCU WITH ADC

Other factors involving a more specific ADC system contribute to reduced ADC performance. Thus, this discussion will focus on the ADC system found on the Motorola M68HC11 Family of MCUs.

A unique implementation of an SAC, the standard M68HC11 (2 MHz bus) ADC provides a $16\ \mu\text{s}$ 8-bit A/D conversion with the convenience of an on-chip MCU peripheral. The ADC is a charge-redistribution SAC. The digital-to-analog converter (DAC) is implemented with capacitors rather than the usual R-2R silicon-chromium (SiCr) thin-film resistors. Although the SiCr resistor has the advantage over the commonly used diffused resistor in improved temperature stability and tracking, laser trimming is necessary to obtain ADC accuracies compatible with even medium-resolution converters. Processing this R-2R ladder presents a challenge since trimming one resistor in the network will change the current in the previously trimmed bit, requiring an iterative trimming process. Furthermore, the R-2R ladder requires careful control of the ON resistance in the MOS switches because the switches also determine the current flow through the R-2R network. The M68HC11 capacitive DAC avoids these shortcomings. The charge-redistribution method is easily fabricated using poly-poly capacitors. No trimming of the poly capacitors or MOS switches is required to obtain medium-resolution accuracies. As an added benefit, a sample-hold function, which extends the effective conversion bandwidth of the ADC, is an inherent byproduct of the redistribution technique.

The internal operations of the M68HC11 converter are relevant to preventing or reducing ADC errors. For converters using SiCr R-2R ladders, the impact of parametric phenomena may be different than for the M68HC11. It is necessary to understand the nature and implementation of the ADC to realize the highest performance from it. To understand the M68HC11 conversion process, a 2-bit example is presented (see Figure 5). A conversion is accomplished by a sequence of three operations. In the sample mode (see Figure 5(a)), the top plate is connected to V_L (0 V), and the bottom plates are connected to the input voltage, V_X , resulting in a stored charge on the top plate that is proportional to the input voltage. In the hold mode (see Figure 5(b)), the top switch is then opened, and the bottom plates are connected to V_L . Since the charge on the top plate is conserved, its potential goes to $-V_X$, which is the initial voltage at the input of the comparator. The approximation mode (see Figure 5(c)), begins by testing the value of the most significant bit by raising the bottom plate of the largest capacitor to the reference voltage, V_H . The equivalent circuit is now actually a voltage divider between two equal capacitances. The output of the comparator, after each capacitor is switched, determines whether the bottom plate of that capacitor will remain at V_H or be returned to V_L before the next capacitor is switched. Conversion proceeds in this manner until all bits have been determined and the result is stored in the successive-approximation register (SAR).

The following major sources of M68HC11 ADC errors controllable by external circuit parameters are discussed in the following paragraphs.

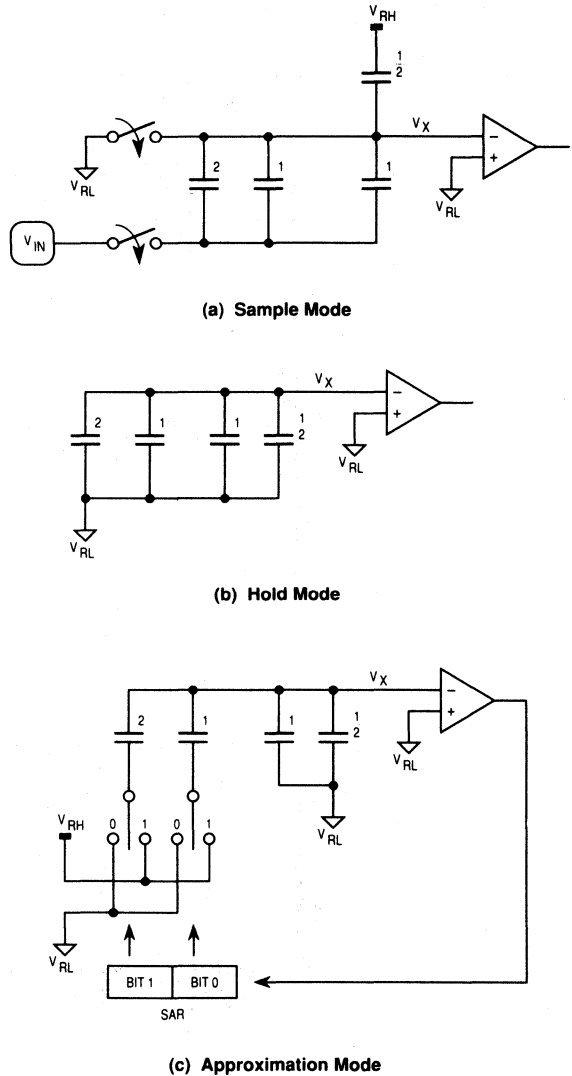


Figure 5. ADC Conversion Modes

Leakage Current on ADC Input Pin

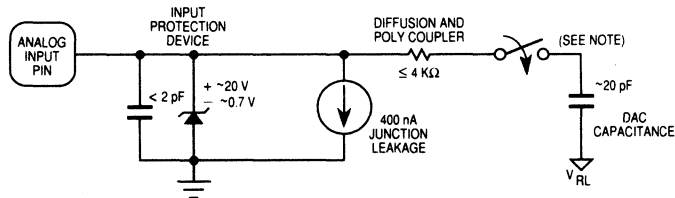
The electrical model of an M68HC11 ADC input pin is shown in Figure 6. The problem is caused by n-channel device junction leakages at this node (there are no p-channel devices used here), which are worse at high temperatures. Consequently, the leakage current is (1) unidirectional and (2) bound by the maximum specification of 400 nA. This leakage-induced error would tend to only cause a static lowering of ADC results. To avoid leakage effects, the external circuit network feeding the ADC pin(s) should maintain impedances, which, in the presence of maximum leakage, would guarantee a maximum desired error. For example, if the maximum error (due to leakage) is desired to be ≤ 1 LSB with a 5 V reference voltage, then the maximum source impedance (resistance) feeding this pin should be $50 \text{ k}\Omega = (19.5 \text{ mV}/400 \text{ nA})$.

Charge Time on Sample Capacitor

By lengthening the resistance-capacitance (RC) time constant, comprised of the source resistance feeding the ADC pin and the DAC capacitance evidenced at the pin during the sample mode, errors may result. However, given the size of the DAC input capacitance, the size of the source resistance necessary to induce these RC time-constant errors will probably be inundated by the effects of pin leakage described previously.

V_{DD}/V_{SS} and Input Terminal Noise

The differential comparator used in the M68HC11 ADC derives its power from V_{DD} and V_{SS}, the power pins that supply the rest of the M68HC11 (see Figure 7). The M68HC11,



NOTE: This analog switch is closed only during the 12-cycle sample time.

Figure 6. Electrical Model of an M68HC11 ADC Input Pin

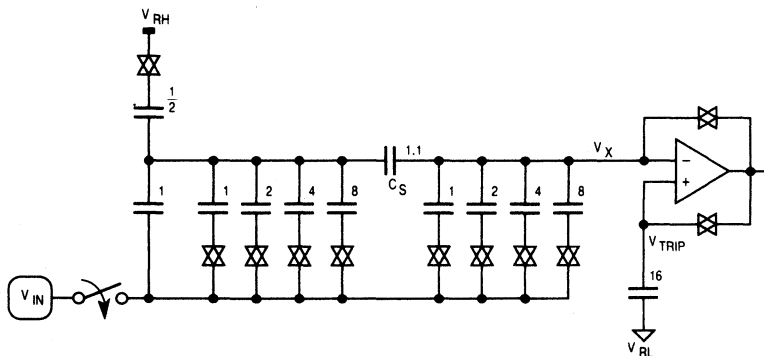


Figure 7. MC68HC11 ADC in Sample Mode

when considered with respect to ADC performance, is a source of noise, partially due to the waveshape and harmonics associated with square waves. In addition, the complex relationship between the primary M68HC11 clock and related noise voltages are further complicated by dependance of the M68HC11 upon many software combinations, each sufficiently changing the noise characteristics emanating from the M68HC11. Therefore, ADC performance degradation, which is linked to noise generated on V_{DD}/V_{SS} by the M68HC11, can often appear related to execution of specific software combinations. As established earlier, this is due to the ADC wide bandwidth comparator.

NOTE

Because the M68HC11 ADC uses a very wide bandwidth comparator capable of responding to noise components in excess of 20 MHz, it must be guarded against unwanted noise at its input terminals and V_{DD}/V_{SS} pins.

The reference to input terminal noise must be distinguished between noise externally superimposed on the input signal lines that is measured between a system reference and a given input signal (occurs from capacitive coupling between high-impedance ADC inputs and noisy signal sources or electro-magnetic interference) and voltage differentials experienced by different comparator inputs when referenced to each other. The importance of input terminal noise in this context is the presence of non-common-mode differential noise between the biasing networks in the comparator and the input lines. If, under noisy conditions, the same noise is presented to an input to the comparator and one of the supply (or other input) terminals, the common-mode rejection ratio (CMRR) capabilities of the comparator may prevent performance perturbations; whereas, noise presented to either terminal, with respect to system ground, may cause havoc (see **Wide Bandwidth Input Signals**). Efforts should be made to ensure that noise, if it cannot be reduced further, is also seen by the other comparator inputs to take advantage of the CMRR. Of the ADC error sources, this is one of the most challenging to control in a practical and effective manner.

Wide Bandwidth Input Signals

A certain way to disrupt ADC function is to give the wide bandwidth comparator something to respond to other than the input signal of interest. By designing the electronics feeding the ADC inputs to pass input signals having frequencies that range from DC to purple, ADC problems are usually guaranteed. Thus, this fourth area is a common source of ADC malfunction.

Other Error Sources

Although occurring less frequently and more subtly, other error sources can also impact ADC performance: rate of conversion requests to a particular channel and interchannel charge-sharing. These sources and an estimate of the impact on a given M68HC11 system are presented in detail in M68HC11RM/AD, *M68HC11 Reference Manual*.

REAL-WORLD EXAMPLE

When discussing the mechanics of noise phenomena in MCU/ADC systems, it is very difficult to understand how large the noise problem is, how well it is expected to respond to corrective action, and how closely the analysis matches the real world. To help resolve these problems, an actual troubleshooting session involving an M68HC11-based assembly is presented.

The subject assembly, an industrial controller, is a typical MCU/CPU installation utilizing the M68HC11 in expanded multiplexed (CPU) mode. The customer designed the program memory to expand to 32K x 8, RAM to 2K x 8, an external address decoder, some additional digital I/O lines, and analog buffers feeding the ADC inputs. Built on a six-layer PCB, the assembly had the benefit of separate ground and voltage planes, and was designed to be placed in a Faraday shield providing electromagnetic compatibility. This assembly was designed without the aid of any of the concepts presented in this application note. Understandably, the customer was having difficulty with ADC performance.

The Problem

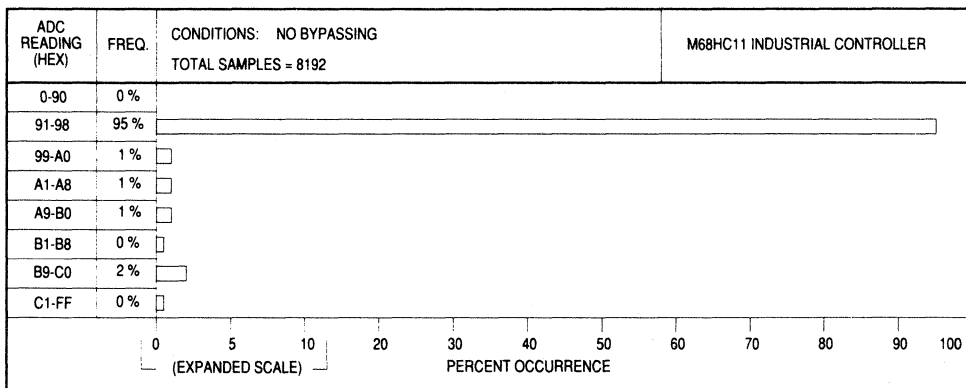
Functionally, the ADC noise problem manifested itself as an extreme shutdown condition in the final product. Since this assembly provides control to industrial equipment, conditions sensed by this controller could indicate dangerous conditions, which must be dealt with by severe and swift action, including functional shutdown of the controlled equipment. To achieve the safest response times and largest safety margins to such stimuli, the software designers of this system required 64 μ s continuous conversions (>15 kHz sampling frequency). Once they were run through part of the designer's algorithm, the conversion results could not deviate more than ± 2 counts from the actual system ADC measurements. The M68HC11 was selected for this application because of its high level of integration as well as the ± 1 LSB 8-bit ADC performance. Errors many times this specification were encountered in the application. Unfortunately, evaluation of the extent of the ADC errors concerned only functional operation of the assembly and manual inspection of ADC values read with an in-circuit emulator, making the problem more serious. An attempt by the hardware engineers to reduce the noise by changing the bypassing scheme yielded no apparent change in the pattern of product shutdown. In this case, the lack of quantitative data convinced the engineers that they had no control over the problem, diverting attention from the actual cause. When dealing with these types of problems, always instrument the problem correctly — that is, ensure measuring techniques used to observe the malfunction follow these guidelines:

1. Quantify the A/D conversion process with regard to frequency of occurrence and magnitude of error.
2. Ensure that the measurements are with sufficient resolution so that minute improvements or degradations in performance may be monitored and evaluated.
3. Ensure that the number of observed conversions are similar to product usage or are statistically significant to allow inference from the measured sample to actual product operation.

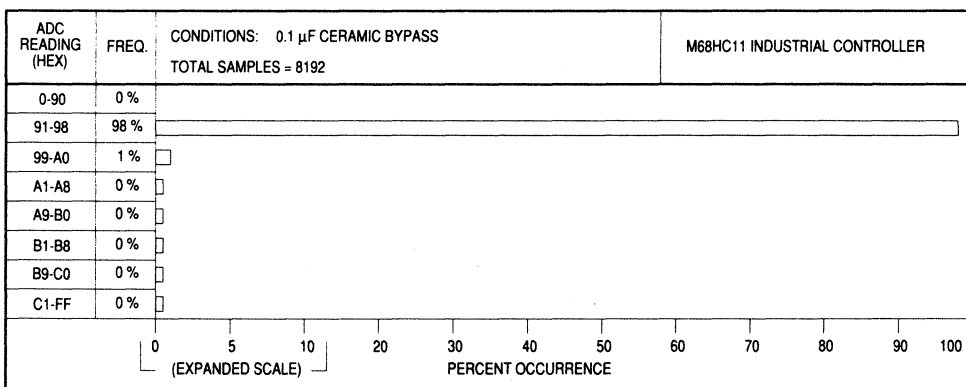
Had the assembly been properly monitored, an improvement in ADC performance with the different bypassing scheme would have been evident (see Figure 8). These two histograms display ranges of A/D conversion values on the vertical axis and the hit rate (percent of total readings landing within the boundaries of the selected ADC reading range) on the horizontal axis. As shown, there was approximately a 3% improvement in the number of correct A/D conversions with new bypassing. To detect these changes, the EPROM on the controller PCB was probed with a fairly simple logic analyzer. The logic analyzer was then configured to trigger on accesses to a location in memory containing the results of A/D conversions. By utilizing the simple statistics options given by the analyzer, each quantitative improvement in ADC performance was observed.

The Perfect Circuit

After sufficiently instrumenting the offending assembly, the next step is to attempt to duplicate ideal operating conditions for the MCU/ADC. Since every M68HC11 is 100% tested for ADC performance before leaving the factory, in the absence of externally induced failure, the M68HC11 should maintain factory performance given identical operating conditions. By operating the M68HC11 in near perfect conditions, the engineer learns if the failure is or is not parts related. The motivating factor for this case, however, concerned 1 V_{pp} noise (spaced in time at approximately the E-clock rate of the M68HC11) found when measuring V_{DD} at the pins of the M68HC11. Given what is known about the ADC comparator, it was best for system performance to reduce this V_{DD} noise as much as possible. The noise was reduced by isolating the power bussing to the M68HC11 only. The PCB foil was cut to V_{DD}, V_{SS}, V_{RL}.



(a) No Bypassing



(b) Improved Bypassing

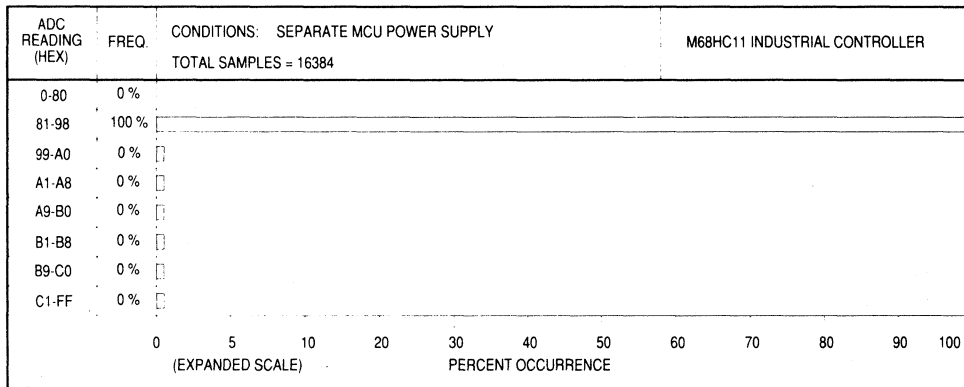
NOTE: Readings are rounded to nearest 1% value. Columns with 0% and a grey bar imply >0% and <0.5%.

Figure 8. Effect of Bypassing Only

and V_{RH} leading to the M68HC11. Discrete wires were then run directly to an external laboratory-grade power supply. With this configuration, measurements were taken as before. The results of these measurements are shown in Figure 9. As the graph shows, an improvement was made over the non-bypassed assembly. Instead of a 5% error in the ADC readings, less than 0.5% of the readings were outside of the expected range. Also evident in Figure 9 is the presence of full-scale errors as before. At this point, a bypass capacitor was soldered between the M68HC11 V_{DD} and V_{SS} pins. The resulting measurements, shown in Figure 10, are an apparent improvement over the previous non-bypassed assembly. However, due to the granularity of the measurement reported by the logic analyzer, it cannot be stated quantitatively how

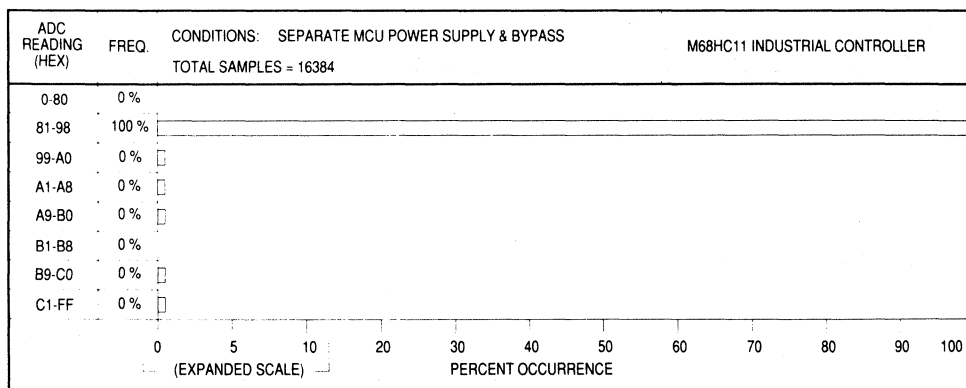
much the bypassing improved the condition. Further manipulation of the bypassing network failed to improve the readings in a discernible manner.

At this point, only power distribution busses had been manipulated to reduce ADC errors. Another part of the ADC circuit manipulated to yield some improvement was the linear portion interfacing the MCU to the various input signals. Consisting of 324-type opamps operated at unity gain, this linear buffer provided a low-impedance source for the ADC input multiplexer. Although not usually considered a wideband op-amp, it proved too wideband for this system. Most data coming from the devices feeding the 324 buffers were slowly varying DC or signals with frequencies below 500 Hz. Yet, the full bandwidth of signals allowed by the buffer passed unaltered to



NOTE: Readings are rounded to nearest 1% value. Columns with 0% and a grey bar imply >0% and <0.5%.

Figure 9. Separate MCU Power Supply



NOTE: Readings are rounded to nearest 1% value. Columns with 0% and a grey bar imply >0% and <0.5%.

Figure 10. Separate MCU Power Supply with 0.1 μ F Bypass

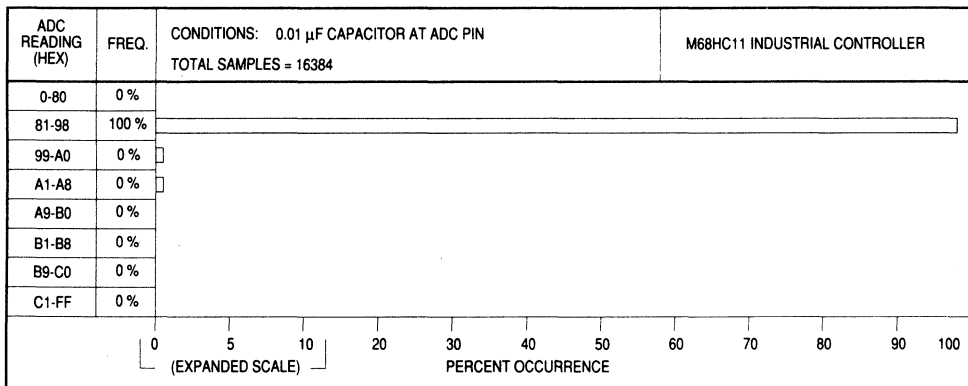
the ADC inputs. This manipulation violated a design guideline that urges the designer to tailor the bandwidth of each ADC channel to the bandwidth of the input signal. By properly filtering the input to the ADC, frequencies that may prove troublesome if left unfiltered will not be allowed to pass to the ADC input. To test the affect of this guideline on this specific industrial controller, 0.01 μF capacitors were soldered to the ADC input pins at the M68HC11. The measurements taken with this configuration (see Figure 11) showed significant improvement. As shown in Figure 11, there were still occasional occurrences of out-of-spec ADC readings.

In the absence of other guidelines, the only choice left to achieve specified ADC performance was to refine the implementation of the existing guidelines. The second guideline, duplicate ideal operating conditions, is usually the most likely candidate for improvement. One of the corollaries to duplicating ideal operating conditions is reducing unwanted interaction between adjacent circuit segments. In this case, V_{DD} noise had not been completely eradicated. Rather than inserting a local IC regulator for just the M68HC11, an alternative method

of V_{DD} isolation was attempted: a series diode with V_{DD} forming a peak detector with the bypass cap. Out-of-spec ADC errors were totally eliminated (see Figure 12(a)). To check the thoroughness of this last circuit fix, the range of sensitivity for the ADC result range of interest was changed on the logic analyzer. By changing the range to show values between $(94)_{16}$ and $(96)_{16}$, inclusively, the ± 1 LSB spec could be observed directly. The results of this measurement run are shown in Figure 12(b).

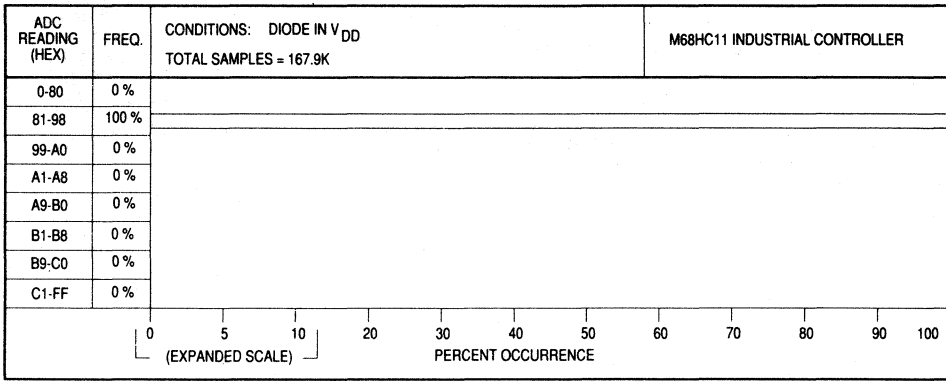
SUMMARY

The highly integrated MCU can be a cost-effective design tool. With the breadth of MCU choices available to the circuit designer these days, analog circuit functions may now often be implemented by MCUs with integral ADCs. By following the practical guidelines presented in this application note during the design phase, the MCU-based product design using the on-chip ADC can achieve its full cost-effective potential.

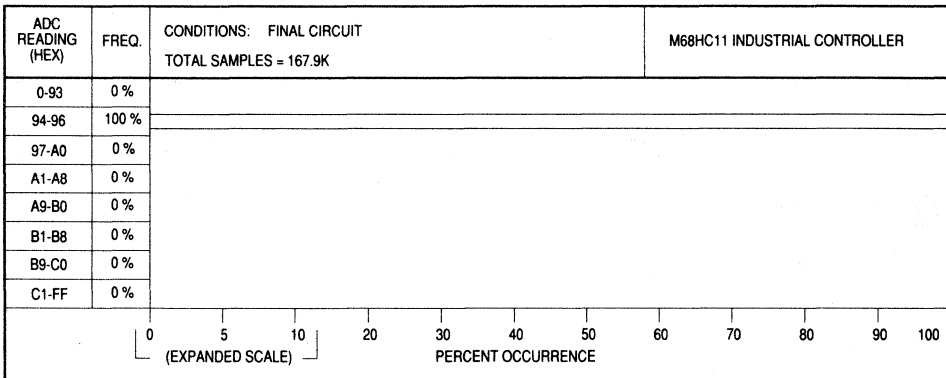


NOTE: Readings are rounded to nearest 1% value. Columns with 0% and a grey bar imply $>0\%$ and $<0.5\%$.

Figure 11. Capacitor on ADC Pin



(a) Diode in V_{DD}



(b) Tightened ADC Range

NOTE: Readings are rounded to nearest 1% value. Columns with 0% and a grey bar imply >0% and <0.5%.

Figure 12. V_{DD} Diode and Tightened ADC Range

MC68HC11 Bootstrap Mode

Prepared by: Jim Sibigroth
Mike Rhoades
John Langan

INTRODUCTION

M68HC11 MCUs have a bootstrap mode that allows a user-defined program to be loaded into the internal random access memory (RAM) by way of the serial communications interface (SCI); the M68HC11 then executes this loaded program. The loaded program can do anything a normal user program can do as well as anything a factory test program can do because protected control bits are accessible in bootstrap mode. Although the bootstrap mode is a single-chip mode of operation, expanded mode resources are accessible because the mode control bits can be changed while operating in the bootstrap mode.

This application note explains the operation and application of the M68HC11 bootstrap mode. Although the basic concepts associated with this mode are quite simple, the more subtle implications of these functions require careful consideration. Useful applications of this mode are overlooked due to an incomplete understanding of the bootstrap mode. Also, common problems associated with the bootstrap mode could be avoided by a more complete understanding of its operation and implications.

Topics included in this application note are as follows:

- Basic operation of the M68HC11 bootstrap mode
- General discussion of bootstrap mode uses
- Detailed explanation of on-chip bootstrap logic
- Detailed explanation of bootstrap firmware
- Bootstrap firmware vs. EEPROM security
- Incorporating the bootstrap mode into a system
- Driving bootstrap mode from another M68HC11
- Driving bootstrap mode from a personal computer
- Common bootstrap mode problems
- Variations for specific versions of M68HC11
- Commented listings for selected M68HC11 bootstrap ROMs

BASIC BOOTSTRAP MODE

This section describes only basic functions of the bootstrap mode. Other functions of the bootstrap mode are described in detail in the remainder of this application note.

When an M68HC11 is reset in bootstrap mode, the reset vector is fetched from a small internal read-only memory (ROM) called the bootstrap ROM or (boot ROM). The firmware program in this boot ROM then controls the bootloading pro-

cess. First, the on-chip SCI is initialized. The first character received (\$FF) determines which of two possible baud rates should be used for the remaining characters in the download operation. Next, a binary program is received by the SCI system and is stored in RAM. Finally, a jump instruction is executed to pass control from the bootloader firmware to the user's loaded program. Bootstrap mode is useful both at the component level and after the MCU has been embedded into a finished user system.

At the component level, Motorola uses the bootstrap mode to control a monitored burn-in program for the on-chip electrically erasable programmable read-only memory (EEPROM). Units to be tested are loaded into special circuit boards that each hold fifty MCUs. These boards are then placed in burn-in ovens. Driver boards outside the ovens download an EEPROM exercise and diagnostic program to all fifty MCUs in parallel. The MCUs under test independently exercise their internal EEPROM and monitor programming and erase operations. This technique could be utilized by an end user to load program information into the EPROM or EEPROM of an M68HC11 before it is installed into an end product. As in the burn-in setup, many M68HC11s can be gang programmed in parallel. This technique can also be used to program the EPROM of finished products after final assembly.

Motorola also uses bootstrap mode for programming target devices on the M68HC11EVM Evaluation Modules. Because bootstrap mode is a privileged mode like special test, the EEPROM-based configuration register (CONFIG) can be programmed using bootstrap mode on the EVM.

The greatest benefits from bootstrap mode are realized by designing the finished system so that bootstrap mode can be used *after* final assembly. The finished system need not be a single-chip mode application for the bootstrap mode to be useful because the expansion bus can be enabled after resetting the MCU in bootstrap mode. Allowing this capability requires almost no hardware or design cost and the addition of this capability is invisible in the end product until it is needed.

The ability to control the embedded processor through downloaded programs is achieved without the disassembly and chip-swapping usually associated with such control. This mode provides an easy way to load non-volatile memories such as EEPROM with calibration tables or to program the application firmware into a one-time programmable (OTP) MCU after final assembly.

Another powerful use of bootstrap mode in a finished assembly is for final test. Short programs can be downloaded to check parts of the system, including components and circuitry external to the embedded MCU. If any problems appear during product development, diagnostic programs can be downloaded to find the problems, and corrected routines can be downloaded and checked before incorporating them into the main application program.

Bootstrap mode can also be used to interactively calibrate critical analog sensors. Since this calibration is done in the final assembled system, it can compensate for any errors in discrete interface circuitry and cabling between the sensor and the analog inputs to the MCU. Note that this calibration routine is a downloaded program that does not take up space in the normal application program.

BOOTSTRAP MODE LOGIC

In the MC68HC11 very little logic is dedicated to the bootstrap mode: Thus, this mode adds almost no extra cost to the MCU system. The biggest piece of circuitry for bootstrap mode is the small boot ROM. This ROM is 192 bytes in the original MC68HC11A8, but some of the newest members of the M68HC11 Family have as much as 448 bytes to accommodate added features. Normally, this boot ROM is present in the memory map only when the MCU is reset in the bootstrap mode to prevent interference with the user's normal memory space. The enable for this ROM is controlled by the read boot ROM (RBOOT) control bit in the highest priority interrupt (HPRIO) register. The RBOOT bit can be written by software whenever the MCU is in special test or special bootstrap modes; when the MCU is in normal modes, RBOOT reverts to zero and becomes a read-only bit. All other logic in the MCU would be present whether or not there was a bootstrap mode.

Figure 1 shows the composite memory map of the MC68HC711E9 in its four basic modes of operation, including bootstrap mode. The active mode is determined by the mode A (MDA) and special mode (SMOD) control bits in the HPRIO control register. These control bits are in turn controlled by the state of the mode A (MODA) and mode B (MODB) pins during reset. Table 1 shows the relationship between the state of these pins during reset, the selected mode, and the state of the MDA, SMOD, and RBOOT control bits. Refer to the composite memory map and Table 1 for the following discussion.

The MDA control bit is determined by the state of the MODA pin as the MCU leaves reset. MDA selects between single-chip and expanded operating modes. When MDA is zero, a single-chip mode is selected, either normal single chip or special bootstrap mode. When MDA is one, an expanded mode is selected, either normal expanded mode or special test mode.

The SMOD control bit is determined by the inverted state of the MODB pin as the MCU leaves reset. SMOD controls whether a normal mode or a special mode is selected. When SMOD is zero, one of the two normal modes is selected, either normal single-chip or normal expanded mode. When SMOD is one, one of the two special modes is selected, either special bootstrap mode or special test mode. When either special mode is in effect (SMOD = 1), certain privileges are in effect —

Table 1. Mode Selection Summary

Input Pins		Mode Selected	Control Bits in HPRIO		
MODB	MODA		RBOOT	SMOD	MDA
1	0	Normal Single Chip	0	0	0
1	1	Normal Expanded	0	0	1
0	0	Special Bootstrap	1	1	0
0	1	Special Test	0	1	1

i.e., the ability to write to the mode control bits and fetching the reset and interrupt vectors from \$BFxx rather than \$FFxx.

The alternate vector locations are achieved by simply driving address bit A14 low during all vector fetches if SMOD = 1. For special test mode, the alternate vector locations assure that the reset vector can be fetched from external memory space so the test system can control MCU operation. In special bootstrap mode, the small boot ROM is enabled in the memory map by RBOOT = 1 so the reset vector will be fetched from this ROM and the bootloader firmware will control MCU operation.

RBOOT is reset to one in bootstrap mode to enable the small boot ROM. In the other three modes, RBOOT is reset to zero to keep the boot ROM out of the memory map. While in special test mode, SMOD = 1; which allows the RBOOT control bit to be written to one by software to enable the boot ROM for testing purposes.

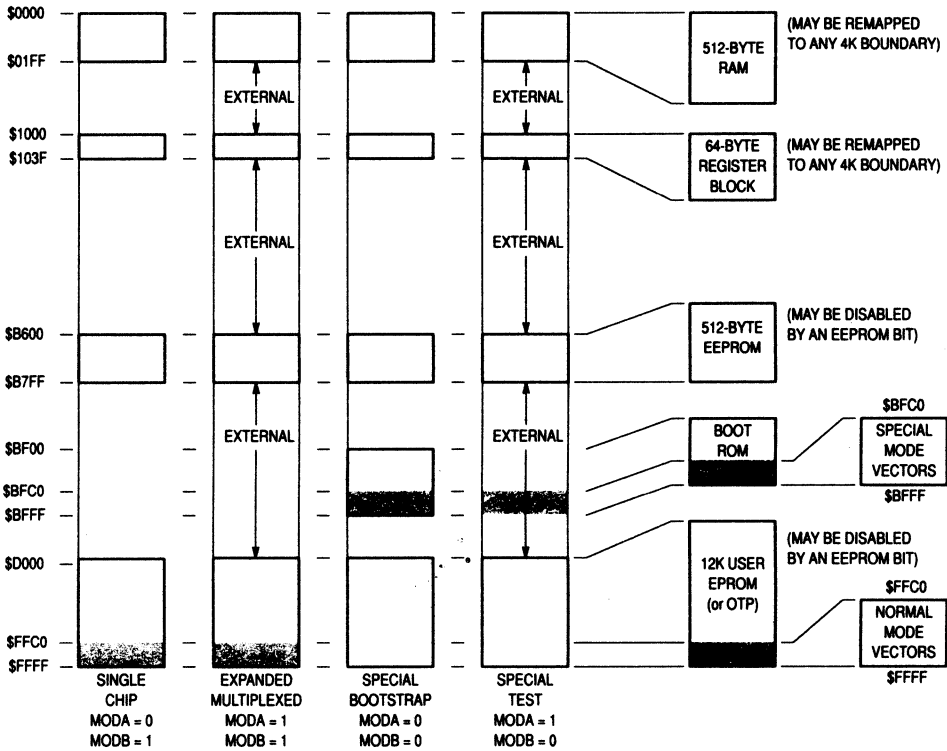
BOOT ROM FIRMWARE

The main program in the boot ROM is the bootloader, which is automatically executed as a result of resetting the MCU in bootstrap mode. Some newer versions of the M68HC11 Family have additional utility programs that can be called from a downloaded program. One utility is available to program EPROM or OTP versions of the M68HC11. A second utility allows the contents of memory locations to be uploaded to a host computer. In the MC68HC711K4 boot ROM, a section of code is used by Motorola for stress testing the on-chip EEPROM. These test and utility programs are similar to self-test ROM programs in other MCUs except that the boot ROM does not use valuable space in the normal memory map.

Bootstrap firmware is also involved in an optional EEPROM security function on some versions of the M68HC11. This EEPROM security feature prevents a software pirate from seeing what is in the on-chip EEPROM. The secured state is invoked by programming the no security (NOSEC) EEPROM bit in the CONFIG register. Once this NOSEC bit is programmed to zero, the MCU will ignore the mode A pin and always come out of reset in normal single-chip mode or special bootstrap mode, depending on the state of the mode B pin. Normal single-chip mode is the usual way a secured part would be used. Special bootstrap mode is used to disengage the security function (only after the contents of EEPROM and RAM have been erased). Refer to the M68HC11RM/AD, *M68HC11 Reference Manual* for additional information on the security mode and complete listings of the boot ROMs that support the EEPROM security functions.

AUTOMATIC SELECTION OF BAUD RATE

The bootloader program in the MC68HC711E9 accommodates either of two baud rates. The higher of these baud rates (7812 baud at a 2-MHz E-clock rate) is used in systems that operate from a binary frequency crystal such as 223 Hz (8.389 MHz). At this crystal frequency the baud rate is 8192 baud which was used extensively in automotive applications based on the MC6801 MCU. The second baud rate available to the M68HC11 bootloader is 1200 baud at a 2-MHz E-clock rate. Some of the newest versions of the M68HC11 accommodate other baud rates using the same differentiation technique explained here. Refer to the reference numbers in square brackets in Figure 2 during the following explanation.



NOTE: Software can change some aspects of the memory map after reset.

Figure 1. MC68HC711E9 Composite Memory Map

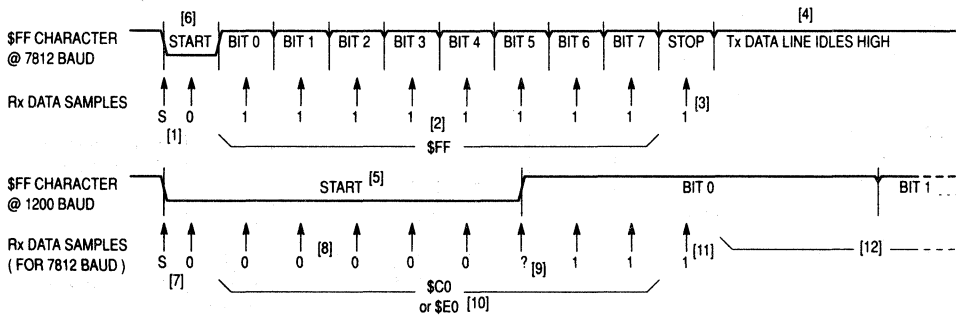


Figure 2. Automatic Detection of Baud Rate

Figure 2 shows how the bootloader program differentiates between the default baud rate (7812 baud at a 2-MHz E-clock rate) and the alternate baud rate (1200 baud at a 2-MHz E-clock rate). The host computer sends an initial \$FF character, which is used by the bootloader to determine the baud rate that will be used for the downloading operation. The top half of Figure 2 shows normal reception of \$FF. Receive data samples at [1] detect the falling edge of the start bit and then verify the start bit by taking a sample at the center of the start bit time. Samples are then taken at the middle of each bit time [2] to reconstruct the value of the received character (all ones in this case). A sample is then taken at the middle of the stop bit time as a framing check (a one is expected) [3]. Unless another character immediately follows this \$FF character, the receive data line will idle in the high state as shown at [4].

The bottom half of Figure 2 shows how the receiver will incorrectly receive the \$FF character that is sent from the host at 1200 baud. Because the receiver is set to 7812 baud, the receive data samples are taken at the same times as in the upper half of Figure 2. The start bit at 1200 baud [5] is 6.5 times as long as the start bit at 7812 baud [6].

Samples taken at [7] detect the falling edge of the start bit and verify it is a logic zero. Samples taken at the middle of what the receiver thinks are the first five bit times [8] detect logic zeros. The sample taken at the middle of what the receiver thinks is bit 5 [9] may detect either a zero or a one because the receive data has a rising transition at about this time. The samples for bits 6 and 7 detect ones, causing the receiver to think the received character was \$C0 or \$E0 [10] at 7812 baud instead of the \$FF which was sent at 1200 baud. The stop bit sample detects a one as expected [11], but this detection is actually in the middle of bit 0 of the 1200 baud \$FF character. The SCI receiver is not confused by the rest of the 1200 baud \$FF character because the receive data line is high [12] just as it would be for the idle condition. If a character other than \$FF is sent as the first character, an SCI receive error could result.

MAIN BOOTLOADER PROGRAM

Figure 3 is a flowchart of the main bootloader program in the MC68HC711E9. This bootloader demonstrates the most important features of the bootloaders used on all M68HC11 Family members. For complete listings of other M68HC11 versions refer to Listings 3–8 at the end of this application note, and

appendix B of the M68HC11RM/AD, *M68HC11 Reference Manual*.

The reset vector in the boot ROM points to the start [1] of this program. The initialization block [2] establishes starting conditions and sets up the SCI and port D. The stack pointer is set because there are push and pull instructions in the bootloader program. The X index register is pointed at the start of the register block (\$1000) so indexed addressing can be used. Indexed addressing takes one less byte of ROM space than extended instructions, and bit manipulation instructions are not available in extended addressing forms. The port D wire-OR mode (DWOM) bit in the serial peripheral interface control register (SPCR) is set to configure port D for wired-OR operation to minimize potential conflicts with external systems that use the PD1/TxD pin as an input. The baud rate for the SCI is initially set to 7812 baud at a 2-MHz E-clock rate but can automatically switch to 1200 baud based on the first character received. The SCI receiver and transmitter are enabled. The receiver is required by the bootloading process, and the transmitter is used to transmit data back to the host computer for optional verification. The last item in the initialization is to set an intercharacter delay constant used to terminate the download when the host computer stops sending data to the MC68HC711E9. This delay constant is stored in the timer output compare 1 (TOC1) register, but the on-chip timer is not used in the bootloader program. This example illustrates the extreme measures used in the bootloader firmware to minimize memory usage. However such measures are not usually considered good programming technique because they are misleading to someone trying to understand the program.

After initialization, a break character is transmitted [3] by the SCI. By connecting the TxD pin to the RxD pin (with a pullup because of port D wired-OR mode), this break will be received as a \$00 character and cause an immediate jump [4] to the start of the on-chip EEPROM (\$B600 in the MC68HC711E9). This feature is useful to pass control to a program in EEPROM essentially from reset. Refer to **COMMON BOOTSTRAP MODE PROBLEMS** before using this feature.

If the first character is received as \$FF, the baud rate is assumed to be the default rate (7812 baud at a 2-MHz E-clock rate). If \$FF was sent at 1200 baud by the host, the SCI will receive the character as \$E0 or \$C0 because of the baud rate mismatch, and the bootloader will switch to 1200 baud [5] for the rest of the download operation. When the baud rate is switched to 1200 baud, the delay constant used to monitor the

intercharacter delay must also be changed to reflect the new character time.

At [6], the Y index register is initialized to \$0000 to point to the start of on-chip RAM. The index register Y is used to keep track of where the next received data byte will be stored in RAM. The main loop for loading begins at [7].

The number of data bytes in the downloaded program can be any number between zero and 512 bytes (the size of on-chip RAM). This procedure is called 'variable-length download' and is accomplished by ending the download sequence when an idle time of at least four character times occurs after the last character to be downloaded. In M68HC11 Family members which have 256 bytes of RAM, the download length is fixed at exactly 256 bytes plus the leading \$FF character.

The intercharacter delay counter is started [8] by loading the delay constant from TOC1 into the X index register. The 19-E-cycle wait loop is executed repeatedly until either a character is received [9] or the allowed intercharacter delay time expires [10]. For 7812 baud, the delay constant is 10,241 E cycles (539 X 19 E cycles per loop). Four character times at 7812 baud is 10,240 E cycles (baud prescale of 4 X baud divider of 4 X 16 internal SCI clocks/bit time X 10 bit times/character X 4 character times). The delay from reset to the initial \$FF character is not critical since the delay counter is not started until after the first character (\$FF) is received.

To terminate the bootloading sequence and jump to the start of RAM without downloading any data to the on-chip RAM, simply send \$FF and nothing else. This feature is similar to the jump to EEPROM at [4] except the \$FF causes a jump to the start of RAM. This procedure requires that the RAM has been loaded with a valid program since it would make no sense to jump to a location in uninitialized memory.

After receiving a character, the downloaded byte is stored in RAM [11]. The data is transmitted back to the host [12] as an indication that the download is progressing normally. At [13], the RAM pointer is incremented to the next RAM address. If the RAM pointer has not passed the end of RAM, the main download loop (from [7] to [14]) is repeated.

When all data has been downloaded, the bootloader goes to [16] because of an intercharacter delay timeout [10] or because the entire 512-byte RAM has been filled [15]. At [16], the X and Y index registers are set up for calling the PROGRAM utility routine, which saves the user from having to do this in a downloaded program. The PROGRAM utility is fully explained in EPROM PROGRAMMING UTILITY. The final step of the bootloader program is to jump to the start of RAM [17], which starts the user's downloaded program.

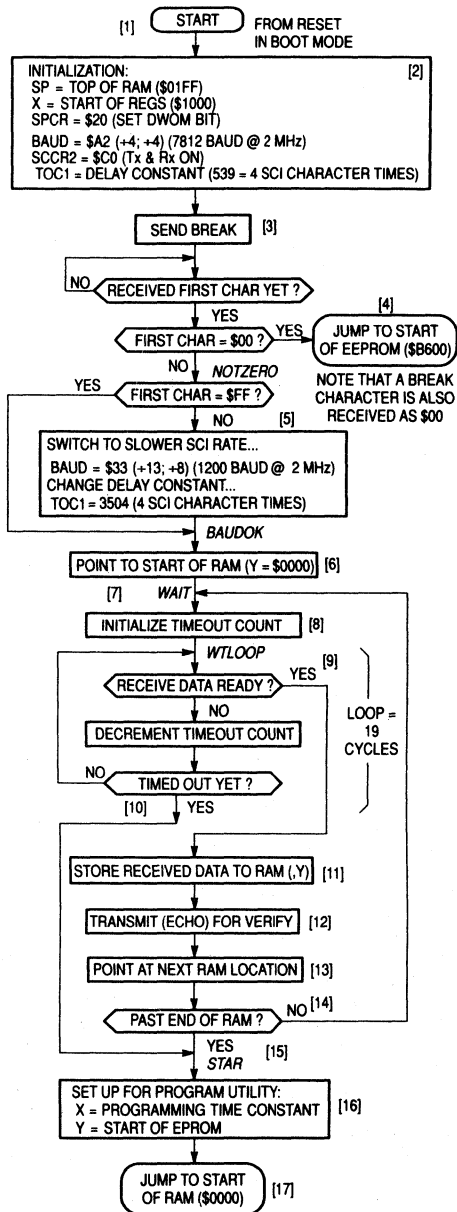


Figure 3. MC68HC711E9 Bootloader Flowchart

UPLOAD UTILITY

The UPLOAD utility subroutine transfers data from the MCU to a host computer system over the SCI serial data link. Note that M68HC11 versions that support EEPROM security do not include this utility. Verification of EPROM contents is one example of how the UPLOAD utility could be used. Before calling this program, the Y index register is loaded (by user firmware) with the address of the first data byte to be uploaded. If a baud rate other than the current SCI baud rate is to be used for the upload process, the user's firmware must also write to the BAUD register. The UPLOAD program sends successive bytes of data out the SCI transmitter until a reset is issued (the upload loop is infinite). For a complete commented listing of the UPLOAD utility, refer to Listings at the back of this application note.

EPROM PROGRAMMING UTILITY

The EPROM programming utility is one way of programming data into the internal EPROM of the MC68HC711E9 MCU. An external 12-V programming power supply is required to program on-chip EPROM. The simplest way to use this utility program is to bootload a three-byte program consisting of a single jump instruction to the start of the PROGRAM utility program (\$BF00). The bootloader program sets the X and Y index registers to default values before jumping to the downloaded program (see [16] at the bottom of Figure 3). When the host computer sees the \$FF character, data to be programmed into the EPROM is sent, starting with the character for location \$D000. After the last byte to be programmed is sent to the MC68HC711E9 and the corresponding verification data is returned to the host, the programming operation is terminated by resetting the MCU.

The number of bytes to be programmed, the first address to be programmed, and the programming time can be controlled by the user if values other than the default values are desired.

To understand the detailed operation of the EPROM programming utility, refer to Figure 4 during the following discussion. Figure 4 is composed of three interrelated parts. The upper-left portion shows the flowchart of the PROGRAM utility running in the boot ROM of the MCU. The upper-right portion shows the flowchart for the user-supplied driver program running in the host computer. The lower portion of Figure 4 is a timing sequence showing the relationship of operations between the MCU and the host computer. Reference numbers in the flowcharts in the upper half of Figure 4 have matching numbers in the lower half to help the reader relate the three parts of the figure.

The shaded area [1] refers to the software and hardware latency in the MCU leading to the transmission of a character (in this case, the \$FF). The shaded area [2] refers to a similar latency in the host computer (in this case, leading to the transmission of the first data character to the MCU).

The overall operation begins when the MCU sends the first character (\$FF) to the host computer, indicating that it is ready for the first data character. The host computer sends the first data byte [3] and enters its main loop. The second data character is sent [4], and the host then waits [5] for the first verify byte to come back from the MCU.

After the MCU sends \$FF [8], it enters the WAIT1 loop [9] and waits for the first data character from the host. When this character is received [10] the MCU programs it into the address pointed to by the Y index register. When the programming time delay is over, the MCU reads the programmed data, transmits it to the host for verification [11], and returns to the top of the WAIT1 loop to wait for the next data character [12]. Because the host previously sent the second data character, it is already waiting in the SCI receiver of the MCU. Steps [13], [14], and [15] correspond to the second pass through the WAIT1 loop.

Back in the host, the first verify character is received, and the third data character is sent [6]. The host then waits for the second verify character [7] to come back from the MCU. The sequence continues as long as the host continues to send data to the MCU. Since the WAIT1 loop in the PROGRAM utility is an indefinite loop, reset is used to end the process in the MCU after the host has finished sending data to be programmed.

ALLOWING FOR BOOTSTRAP MODE

Since bootstrap mode requires very few connections to the MCU, it is easy to design systems that accommodate the bootstrap mode. Bootstrap mode is useful for diagnosing or repairing systems that have failed due to changes in the CONFIG register or failures of the expansion address/data buses, (rendering programs in external memory useless). Bootstrap mode can also be used to load information into the EPROM or EEPROM of an M68HC11 after final assembly of a module. Bootstrap mode is also useful for performing system checks and calibration routines. The following paragraphs explain system requirements for use of bootstrap mode in a product.

MODE SELECT PINS: It must be possible to force the MODA and MODB pins to logic zero, which implies that these two pins should be pulled up to V_{DD} through resistors rather than being tied directly to V_{DD}. If mode pins are connected directly to V_{DD} it is not possible to force a mode other than the one the MCU is hard wired for. It is also good practice to use pulldown resistors to V_{SS} rather than connecting mode pins directly to V_{SS} because it is sometimes a useful debug aid to attempt reset in modes other than the one the system was primarily designed for. Physically, this requirement sometimes calls for the addition of a test point or a wire connected to one or both mode pins. Mode selection only uses the mode pins while RESET is active.

RESET: It must be possible to initiate a reset while the mode select pins are held low. In systems where there is no provision for manual reset, it is usually possible to generate a reset by turning power off and back on.

RxD PIN: It must be possible to drive the PD0/RxD pin with serial data from a host computer (or another MCU). In many systems, this pin is already used for SCI communications; thus no changes are required.

In systems where the PD0/RxD pin is normally used as a general-purpose output, a serial signal from the host can be connected to the pin without resulting in output driver conflicts. It may be important to consider what the existing logic will do with the SCI serial data instead of the signals that would have been produced by the PD0 pin. In systems where the PD0 pin is normally used as a general-purpose input, the driver circuit

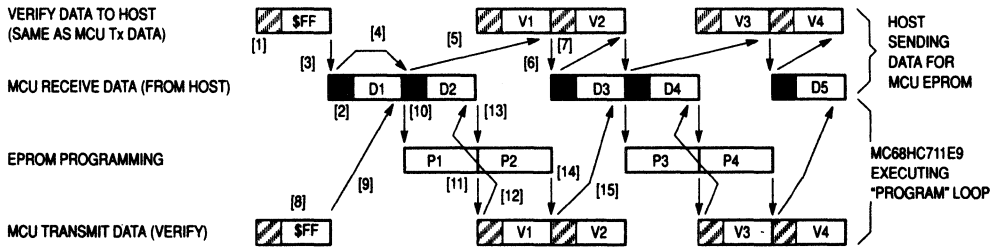
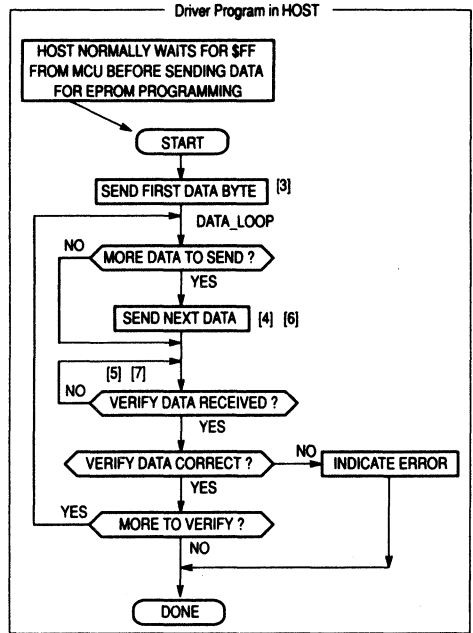
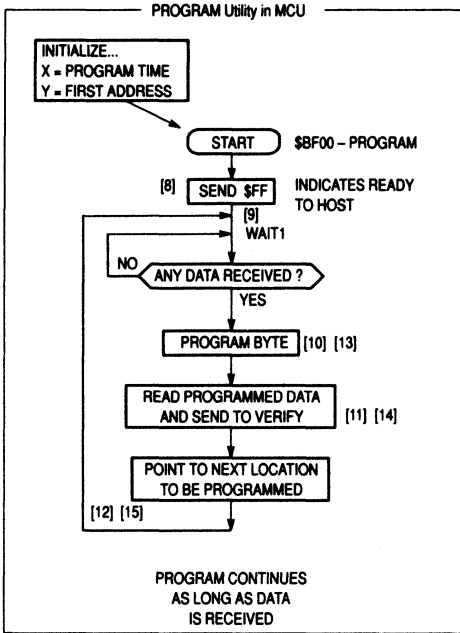


Figure 4. Host and MCU Activity during EPROM PROGRAM Utility

that drives the PD0 pin must be designed so that the serial data can override this driver, or the driver must be disconnected during the bootstrap download. A simple series resistor between the driver and the PD0 pin solves this problem as shown in Figure 5. The serial data from the host computer can then be connected to the PD0/RxD pin, and the series resistor will prevent direct conflict between the host driver and the normal PD0 driver.

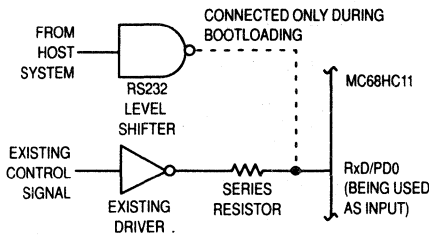


Figure 5. Preventing Driver Conflict

TxD PIN: The bootloader program uses the PD1/TxD pin to send verification data back to the host computer. To minimize the possibility of conflicts with circuitry connected to this pin, port D is configured for wire-OR mode by the bootloader program during initialization. Since the wire-OR configuration prevents the pin from driving active high levels, a pullup resistor to V_{DD} is needed if the TxD signal is used.

In systems where the PD1/TxD pin is normally used as a general-purpose output, there are no output driver conflicts. It may be important to consider what the existing logic will do with the SCI serial data instead of the signals that would have been produced by the PD1 pin.

In systems where the PD1 pin is normally used as a general-purpose input, the driver circuit that drives the PD1 pin must be designed so that the PD1/TxD pin driver in the MCU can override this driver. A simple series resistor between the driver and the PD1 pin can solve this problem. The TxD pin can then be configured as an output, and the series resistor will prevent direct conflict between the internal TxD driver and the external driver connected to PD1 through the series resistor.

OTHER: The bootloader firmware sets the DWOM control bit, which configures all port D pins for wire-OR operation. During the bootloading process, all port D pins except the PD1/TxD pin are configured as high-impedance inputs. Any port D pin that is normally used as an output should have a pullup resistor so it does not float during the bootloading process.

DRIVING BOOT MODE FROM ANOTHER M68HC11

A second M68HC11 system can easily act as the host to drive bootstrap loading of an M68HC11 MCU. This method is used to examine and program nonvolatile memories in target M68HC11s in Motorola EVMs. The following hardware and software example will demonstrate this and other bootstrap mode features.

The schematic in Figure 6 shows the circuitry for a simple EPROM duplicator for the MC68HC711E9. The circuitry is built in the wire-wrap area of an M68HC11EVBU Evaluation

Board to simplify construction. The schematic shows only the important portions of the EVBU circuitry to avoid confusion. To see the complete EVBU schematic, refer to the M68HC11EVBU/D, *M68HC11EVBU Universal Evaluation Board User's Manual*.

The default configuration of the EVBU must be changed to make the appropriate connections to the circuitry in the wire-wrap area and to configure the master MCU for bootstrap mode. A fabricated jumper must be installed at J6 to connect the XTAL output of the master MCU to the wire-wrap connector P5, which has been wired to the EXTAL input of the target MCU. Cut traces that short across J8 and J9 must be cut on the solder side of the printed circuit board to disconnect the normal SCI connections to the RS232 level translator (U4) of the EVBU. The J8 and J9 connections can easily be restored at a later time by installing fabricated jumpers on the component side of the board. A fabricated jumper must be installed across J3 to configure the master MCU for bootstrap mode.

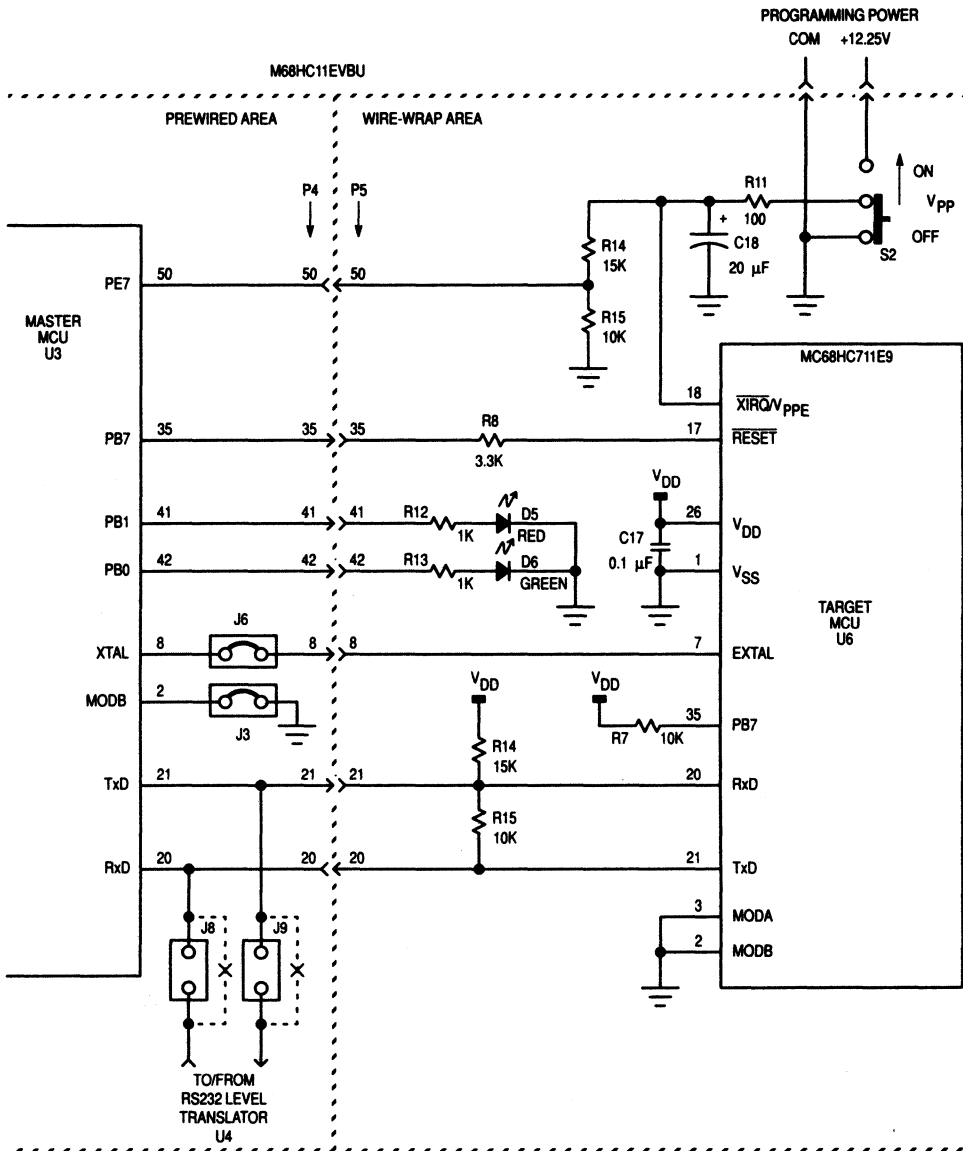
One MC68HC711E9 is first programmed by other means with a desired 12K-byte program in its EPROM and a small duplicator program in its EEPROM. Alternately, the ROM program in an MC68HC11E9 can be copied into the EPROM of a target MC68HC711E9 by programming only the duplicator program into the EEPROM of the master MC68HC11E9. The master MCU is installed in the EVBU at socket U3. A blank MC68HC711E9 to be programmed is placed in the socket in the wire-wrap area of the EVBU (U6).

With the V_{pp} power switch off, power is applied to the EVBU system. As power is applied to the EVBU, the master MCU (U3) comes out of reset in bootstrap mode. Target MCU (U6) is held in reset by the PB7 output of master MCU (U3). The PB7 output of U3 is forced to zero when U3 is reset. The master MCU will later release the reset signal to the target MCU under software control. The RxD and TxD pins of the target MCU (U6) are high-impedance inputs while U6 is in reset so they will not affect the TxD and RxD signals of the master MCU (U3) while U3 is coming out of reset. Since the target MCU is being held in reset with MODA and MODB at zero, it is configured for the EPROM emulation mode, and PB7 is the output enable signal for the EPROM data I/O pins. Pullup resistor R7 causes the port D pins including RxD and TxD, to remain in the high-impedance state so they do not interfere with the RxD and TxD pins of the master MCU as it comes out of reset.

As U3 leaves reset, its mode pins select bootstrap mode so the bootloader firmware begins executing. A break is sent out the TxD pin of U3. Pullup resistor R10 and resistor R9 cause the break character to be seen at the RxD pin of U3. The bootloader performs a jump to the start of EEPROM in the master MCU (U3) and starts executing the duplicator program. This sequence demonstrates how to use bootstrap mode to pass control to the start of EEPROM after reset.

The complete listing for the duplicator program in the EEPROM of the master MCU is provided in Listing 1.

The duplicator program in EEPROM clears the DWOM control bit to change port D (thus, TxD) of U3 to normal driven outputs. This configuration will prevent interference due to R9 when TxD from the target MCU (U6) becomes active. Series resistor R9 demonstrates how TxD of U3 can drive RxD of U3 and later TxD of U6 can drive RxD of U3 without a destructive conflict between the TxD output buffers.



NOTE: Only the most important portions of EVBU circuitry are shown.

Figure 6. MCU to MCU EPROM Duplicator Schematic

As the target MCU (U6) leaves reset, its mode pins select bootstrap mode so the bootloader firmware begins executing. A break is sent out the TxD pin of U6. At this time, the TxD pin of U3 is at a driven high so R9 acts as a pullup resistor for TxD of the target MCU (U6). The break character sent from U6 is received by U3 so the duplicator program that is running in the EEPROM of the master MCU knows that the target MCU is ready to accept a bootloaded program.

The master MCU sends a leading \$FF character to set the baud rate in the target MCU. Next, the master MCU passes a three-instruction program to the target MCU and pauses so the bootstrap program in the target MCU will stop the loading process and jump to the start of the downloaded program. This sequence demonstrates the variable-length download feature of the MC68HC711E9 bootloader.

The short program downloaded to the target MCU clears the DWOM bit to change its TxD pin to a normal driven CMOS output and jumps to the EPROM programming utility in the bootstrap ROM of the target MCU.

Note that the small downloaded program did not have to set up the SCI or initialize any parameters for the EPROM programming process. The bootstrap software that ran prior to the loaded program left the SCI turned on and configured in a way that was compatible with the SCI in the master MCU (the duplicator program in the master MCU also did not have to set up the SCI for the same reason). The programming time and starting address for EPROM programming in the target MCU were also set to default values by the bootloader software before jumping to the start of the downloaded program.

Before the EPROM in the target MCU can be programmed, the Vpp power supply must be available at the XIRQ/VppE pin of the target MCU. The duplicator program running in the master MCU monitors this voltage (for presence or absence — not level) at PE7 through resistor divider R14 - R15. The PE7 input was chosen because the internal circuitry for port E pins can tolerate voltages slightly higher than VDD; therefore resistors R14 and R15 are less critical. No data to be programmed is passed to the target MCU until the master MCU senses that Vpp has been stable for about 200 ms.

When Vpp is ready, the master MCU turns on the red LED and begins passing data to the target MCU. **EPROM PROGRAMMING UTILITY** explains the activity as data is sent from the master MCU to the target MCU and programmed into the EPROM of the target. The master MCU in the EVBU corresponds to the HOST in the programming utility description, and the "PROGRAM utility in MCU" is running in the bootstrap ROM of the target MCU.

Each byte of data sent to the target is programmed and then the programmed location is read and sent back to the master for verification. If any byte fails, the red and green LEDs are turned off, and the programming operation is aborted. If the entire 12K bytes are programmed and verified successfully, the red LED is turned off, and the green LED is turned on to indicate success. The programming of all 12K bytes takes about 30 sec.

After a programming operation, the Vpp switch (S2) should be turned off before the EVBU power is turned off.

Listing 1. MCU to MCU Duplicator Program

Sheet 1 of 2

```

1 *****
2 * 68HC711E9 Duplicator Program for AN1060
3 *****
4
5 *****
6 * Equates - All reg addr except INIT are 2-digit
7 * for direct addressing
8 *****
9 103D INIT EQU $103D RAM, Reg mapping
10 0028 SPCR EQU $28 DWOM in bit-5
11 0004 PORTB EQU $04 Red LED = bit-1, Grn = bit-0
12 * Reset of prog socket = bit-7
13 0080 RESET EQU %10000000
14 0002 RED EQU %00000010
15 0001 GREEN EQU %00000001
16 000A PORTE EQU $0A Vpp Sense in bit-7, 1=ON
17 002E SCSR EQU $2E SCI status register
18 * TDRE, TC, RDRF, IDLE; OR, NF, FE, -
19 0080 TDRE EQU %10000000
20 0020 RDRF EQU %00100000
21 002F SCDR EQU $2F SCI data register
22 BF00 PROGRAM EQU $BF00 EPROM prog utility in boot ROM
23 D000 EPSTRT EQU $D000 Starting address of EPROM
24
25 B600 ORG $B600 Start of EEPROM
26
27 *****
28 *
29 B600 7F103D BEGIN CLR INIT Moves Registers to $0000-3F
30 B603 8604 LDAA #S04 Pattern for DWOM off, no SPI
31 B605 9728 STAA SPCR Turns off DWOM in EVBU MCU
32 B607 8680 LDAA #RESET
33 B609 9704 STAA PORTB Release reset to target MCU
34 B60B 132E20FC WT4BRK BRCLR SCSR RDRF WT4BRK Loop till char received
35 B60F 86FF LDAA #$FF Leading char for bootload ...
36 B611 972F STAA SCDR to target MCU
37 B613 CEB675 LDX #BLPROG Point at program for target
38 B616 8D53 BSR SEND1 Bootload to target
39 B618 8CB67D CPX #ENDBPR Past end ?
40 B61B 26F9 BNE BLOOP Continue till all sent
41
42 *****
43 * Delay for about 4 char times to allow boot related
44 * SCI communications to finish before clearing
45 * Rx related flags
45 B61D CE06A7 LDY #1703 # of 6 cyc loops
46 B620 09 DLYLP DEX [3]
47 B621 26FD BNE DLYLP [3] Total loop time = 6 cyc
48 B623 962E LDAA SCSR Read status (RDRF will be set)
49 B625 962F LDAA SCDR Read SCI data reg to clear RDRF
50
51 *****
52 * Now wait for character from target to indicate it's ready for
53 * data to be programmed into EPROM
53 B627 132E20FC WT4FF BRCLR SCSR RDRF WT4FF Wait for RDRF
54 B62B 962F LDAA SCDR Clear RDRF, don't need data
55 B62D CED000 LDY #EPSTRT Point at start of EPROM
56
57 * Handle turn-on of Vpp
57 B630 18CE523D WT4VPP LDY #21053 Delay counter (about 200ms)
58 B634 150402 BCLR PORTB RED Turn off RED LED
59 B637 960A DLYLP2 LDAA PORTE [3] Wait for Vpp to be ON
60 B639 2AF5 BPL WT4VPP [3] Vpp sense is on port E MSB
61 B63B 140402 BSET PORTB RED [6] Turn on RED LEDdd
62 B63E 1809 DEY [4]
63 B640 26F5 BNE DLYLP2 [3] Total loop time = 19 cyc
64
65 * Vpp has been stable for 200ms
66
66 B642 18CED000 LDY #EPSTRT X=Tx pointer, Y=verify pointer
67 B646 8D23 BSR SEND1 Send first data to target
68 B648 8C0000 CPX #0 X points at $0000 after last
69 B64B 2702 BEQ VERR Skip send if no more
70 B64D 8D1C BSR SEND1 Send another data char
71 B64F 132E20FC VERR BRCLR SCSR RDRF VERR Wait for Rx ready
72 B653 962F LDAA SCDR Get char and clr RDRF
73 B655 18A100 CMPA 0,Y Does char verify ?
74 B658 2705 BEQ VERRFOK Skip error if OK
75 B65A 150403 BCLR PORTB (RED+GREEN) Turn off LEDs
76 B65D 2007 BRA DUNPRG Done (programming failed)
77 B65F
78 B65F 1808 VERRFOK INY Advance verify pointer
79 B661 26E5 BNE DATALP Continue till all done

```


Listing 1. MCU to MCU Duplicator Program

Sheet 2 of 2

```

80 B663
81 B663 140401          BSET  PORTB GREEN      Grn LED ON
82 B666
83 B666 150482  DUNPRG  BCLR  PORTB (RESET+RED) Red OFF, apply reset
84 B669 20FE          BRA   *                   Done so just hang
85 B66B
86
87 * Subroutine to get & send an SCI char. Also
88 * advances pointer (X).
89 *****
90 B66B A600          SEND1  LDAA  0,X          Get a character
91 B66D 132E80FC TRDYLP  BRCLR  SCSR TDRE TRDYLP Wait for TDRE
92 B671 972F          STAA  SCDR          Send character
93 B673 08           INX                    Advance pointer
94 B674 39           RTS                     ** Return **
95
96 *****
97 * Program to be bootloaded to target '711E9
98 *****
99 B675 8604          BLPROG LDAA  #$04          Pattern for DWOM off, no SPI
100 B677 B71028        STAA  $1028          Turns off DWOM in target MCU
101 * NOTE: Can't use direct addressing in target MCU because
102 * regs are located at $1000.
103 B67A 7EBF00        JMP   PROGRAM        Jumps to EPROM prog routine
104 B67D          ENDBPR  EQU   *

```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BEGIN	B600	*00029		
BLLOOP	B616	*00038	00040	
BLPROG	B675	*00099	0037	
DATALP	B648	*00068	00079	
DLVLP	B620	*00046	00047	
DLVLP2	B637	*00059	00063	
DUNPRG	B666	*00083	00076	
ENDBPR	B67D	*00104	00039	
EPSTRT	D000	*00023	00055	00066
GREEN	0001	*00015	00075	00081
INIT	103D	*00009	00029	
PORTB	0004	*00011	00033	00058 00061 00075 00081 00083
PORTE	000A	*00016	00059	
PROGRAM	BF00	*00022	00103	
RDRF	0020	*00020	00034	00053 00071
RED	0002	*00014	00058	00061 00075 00083
RESET	0080	*00013	00032	00083
SCDR	002F	*00021	00036	00049 00054 00072 00092
SCSR	002E	*00017	00034	00048 00053 00071 00091
SEND1	B66B	*00090	00038	00067 00070
SPCR	0028	*00010	00031	
TDRE	0080	*00019	00091	
TRDYLP	B66D	*00091	00091	
VERF	B64F	*00071	00069	00071
VERFOK	B65F	*00078	00074	
WT4BRK	B60B	*00034	00034	
WT4FF	B627	*00053	00053	
WT4VPP	B630	*00057	00060	

```

Errors: None
Labels: 28
Last Program Address: $B67C
Last Storage Address: $0000
Program Bytes: $007D 125
Storage Bytes: $0000 0

```

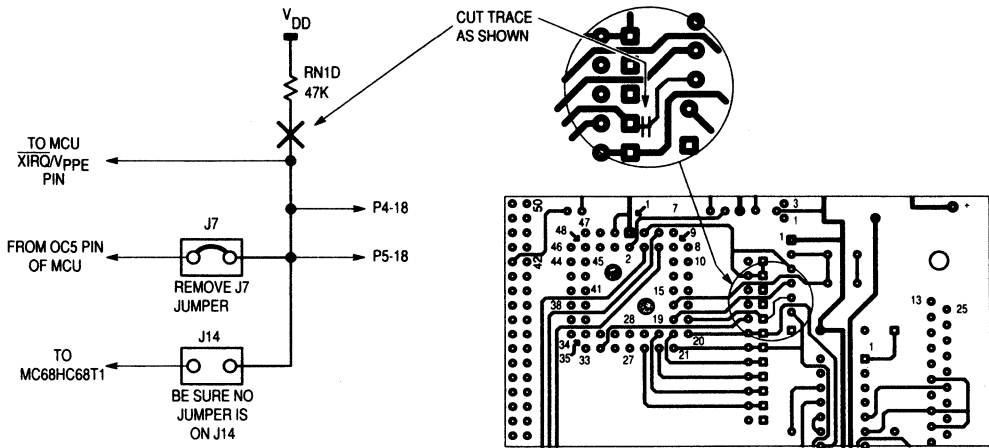


Figure 7. Isolating EVBU XIRQ Pin

DRIVING BOOT MODE FROM A PERSONAL COMPUTER

In this example, a personal computer is used as the host to drive the bootloader of an MC68HC711E9. An M68HC11 EVBU is used for the target MC68HC711E9. A large program is transferred from the personal computer into the EPROM of the target MC68HC711E9.

HARDWARE: Figure 7 shows a small modification to the EVBU to accommodate the 12-V (nominal) EPROM programming voltage. The XIRQ pin is connected to a pullup resistor, two jumpers, and the 60-pin connectors, P4 and P5. The object of the modification is to isolate the XIRQ pin and then connect it to the programming power supply. Carefully cut the trace on the solder side of the EVBU as indicated in Figure 7. This disconnects the pullup resistor RN1D from XIRQ but leaves P4-18, P5-18, and jumpers J7 and J14 connected so the EVBU can still be used for other purposes after programming is done. Remove any fabricated jumpers from J7 and J14. The EVBU normally has a jumper at J7 to support the trace function

Figure 8 shows a small circuit that is added to the wire-wrap area of the EVBU. The three-terminal jumper allows the XIRQ line to be connected to either the programming power supply or to a substitute pullup resistor for XIRQ. The 100-ohm resistor is a current limiter to protect the 12-V input of the MCU. The resistor and LED connected to P5 pin 9 (port C bit 0) is an optional indicator that lights when programming is complete.

SOFTWARE: BASIC was chosen as the programming language due to its readability and availability in parallel versions on both the IBM™1 PC and the Macintosh™2. The program demonstrates several programming techniques for use with an M68HC11 and is not necessarily intended to be a finished, commercial program. For example, there is very little error checking, and the user interface is very elementary. A complete listing of the BASIC program is included in Listing 2 with moderate comments. The following paragraphs include a more detailed discussion of the program as it pertains to communicating with and programming the target

MC68HC711E9. Lines 25–45 initialize and define the variables and array used in the program. Changes to this section would allow for other programs to be downloaded.

1. IBM is a trademark of International Business Machines.
2. Macintosh is a trademark of Apple Computers, Inc.

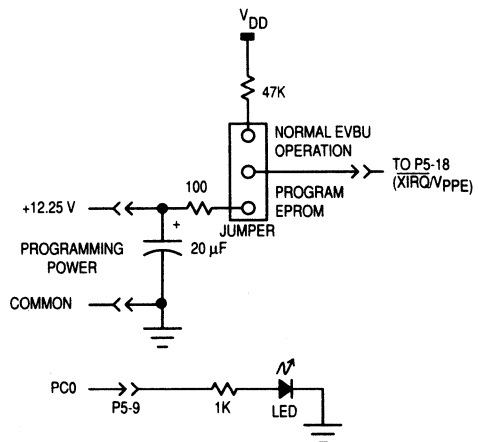


Figure 8. PC to MCU Programming Circuit

Lines 50–95 read in the small bootloader from DATA statements at the end of the listing. The source code for this bootloader is presented in the DATA statements. The bootloaded code makes port C bit 0 low, initializes the X and Y registers for use by the EPROM programming utility routine contained in the boot ROM, and then jumps to that routine. The hexadecimal values read in from the DATA statements are converted to binary values by a subroutine. The binary values are then saved as one string (BOOTCODE\$).

The next long section of code (lines 97–1250) reads in the S-records from an external disk file (in this case, BUF34.S19), converts them to integer, and saves them in an array. The techniques used in this section show how to convert ASCII S-records to binary form that can be sent (bootloaded) to an M68HC11.

This S-record translator only looks for the S1 records that contain the actual object code. All other S-record types are ignored.

When an S1 record is found (line 1000–1024), the next two characters form the hex byte giving the number of hex bytes to follow. This byte is converted to integer by the same subroutine that converted the bootloaded code from the DATA statements. This BYTECOUNT is adjusted by subtracting 3, which accounts for the address and checksum bytes and leaves just the number of object-code bytes in the record.

Starting at line 1100, the two-byte (four-character) starting address is converted to decimal. This address is the starting address for the object-code bytes to follow. An index into the CODE% array is formed by subtracting the base address initialized at the start of the program from the starting address for this S-record.

A FOR-NEXT loop starting at line 1130 converts the object-code bytes to decimal and saves them in the CODE% array. When all the object-code bytes have been converted from the current S-record, the program loops back to find the next S1 record.

A problem arose with the BASIC programming technique used. The draft versions of this program tried saving the object-code bytes directly as binary in a string array. This caused "Out of Memory" or "Out of String Space" errors on both a 2M Macintosh and a 640K PC. The solution was to make the array an integer array and perform the integer-to-binary conversion on each byte as it is sent to the target part.

The one compromise made to accommodate both Macintosh and PC versions of BASIC is in lines 1500 and 1505. Use line 1500 and comment out line 1505 if the program is to be run on a Macintosh and, conversely, use line 1505 and comment out line 1500 if a PC is used.

After the COM port is opened, the code to be bootloaded is modified by adding the \$FF to the start of the string. \$FF synchronizes the bootloader in the MC68HC711E9 to 1200 baud. The entire string is simply sent to the COM port by PRINTing the string. This is possible since the string is actually queued in BASIC's COM buffer, and the operating system takes care of sending the bytes out one at a time. The M68HC11 echoes the data received for verification. No automatic verification is provided, though the data is printed to the screen for manual verification.

Once the MCU has received this bootloaded code, the bootloader automatically jumps to it. The small bootloaded program in turn includes a jump to the EPROM programming routine in the boot ROM.

Refer to the previous explanation of the **EPROM PROGRAMMING UTILITY** for the following discussion. The host system sends the first byte to be programmed through the COM port to the SCI of the MCU. The SCI port on the MCU buffers one byte while receiving another byte, increasing the throughput of the EPROM programming operation by sending the second byte while the first is being programmed.

When the first byte has been programmed, the MCU reads the EPROM location and sends the result back to the host system. The host then compares what was actually programmed to what was originally sent. A message indicating which byte is being verified is displayed in the lower half of the screen. If there is an error, it is displayed at the top of the screen.

As soon as the first byte is verified, the third byte is sent. In the meantime, the MCU has already started programming the second byte. This process of verifying and queuing a byte continues until the host finishes sending data. If the programming is completely successful, no error messages will have been displayed at the top of the screen. Subroutines follow the end of the program to handle some of the repetitive tasks. These routines are short, and the commenting in the source code should be sufficient explanation.

MODIFICATIONS: This example programmed version 3.4 of the BUFFALO monitor into the EPROM of an MC68HC711E9; the changes to the BASIC program to download some other program are minor. The necessary changes are as follows:

1. In line 30, the length of the program to be downloaded must be assigned to the variable "CODESIZE%".
2. Also in line 30, the starting address of the program is assigned to the variable "ADRSTART".
3. In line 9570, the start address of the program is stored in the third and fourth items in that DATA statement in hexadecimal.
4. If any changes are made to the number of bytes in the boot code in the DATA statements in lines 9500–9580, then the new count must be set in the variable "BOOTCOUNT" in line 25.

OPERATION: Configure the EVBU for boot mode operation by putting a jumper at J3. Ensure that the trace command jumper at J7 is not installed because this would connect the 12-V programming voltage to the OC5 output of the MCU.

Connect the EVBU to its DC power supply. When it is time to program the MCU EPROM, turn on the 12-V programming power supply to the new circuitry in the wire-wrap area.

Connect the EVBU serial port to the appropriate serial port on the host system. For the Macintosh, this is the modem port with a modem cable. For the MS-DOS computer, it is connected to COM1 with a "straight through" or modem cable. Power up the host system and start the BASIC program. If the program has not been compiled, this is accomplished from within the appropriate BASIC compiler or interpreter. Power up the EVBU.

Answer the prompt for filename with either a [RETURN] to accept the default shown or by typing in a new filename and pressing [RETURN].

The program will inform the user that it is working on converting the file from S-records to binary. This process will take from 30 sec to a few minutes, depending on the computer.

A prompt reading, "Comm port open?" will appear at the end of the file conversion. This is the last chance to ensure that everything is properly configured on the EVBU. Pressing [RETURN] will send the bootcode to the target MC68HC711E9. The program then informs the user that the bootload code is being sent to the target, and the results of the echoing of this code are displayed on the screen.

Another prompt reading "Programming is ready to begin. Are you?" will appear. Turn on the 12-V programming power supply and press [RETURN] to start the actual programming of the target EPROM.

A count of the byte being verified will be continually updated on the screen as the programming progresses. Any failures will be flagged as they occur.

When programming is complete, a message will be displayed as well as a prompt requesting you to press [RETURN] to quit.

Turn off the 12-V programming power supply before turning off 5 V to the EVBU.

Listing 2. BASIC Program for Personal Computer

Sheet 1 of 3

```

1 *****
2 **
3 **   E9BUF.BAS - A PROGRAM TO DEMONSTRATE THE USE OF THE BOOT MODE
4 **   ON THE HC11 BY PROGRAMMING AN MC68HC711E9 WITH
5 **   BUFFALO 3.4
6 **
7 **   REQUIRES THAT THE S-RECORDS FOR BUFFALO (BUF34.S19)
8 **   BE AVAILABLE IN THE SAME DIRECTORY OR FOLDER
9 **
10 **   THIS PROGRAM HAS BEEN RUN BOTH ON A MS-DOS COMPUTER
11 **   USING QUICKBASIC 4.5 AND ON A MACINTOSH USING
12 **   QUICKBASIC 1.0.
13 **
14 **
15 *****
25 H$ = "0123456789ABCDEF" 'STRING TO USE FOR HEX CONVERSIONS
30 DEFINT B, I: CODESIZE% = 8192: ADRSTART= 57344!
35 BOOTCOUNT = 25 'NUMBER OF BYTES IN BOOT CODE
40 DIM CODE$(CODESIZE%) 'BUFFALO 3.4 IS 8K BYTES LONG
45 BOOTCODE$ = "" 'INITIALIZE BOOTCODE$ TO NULL
49 REM ***** READ IN AND SAVE THE CODE TO BE BOOT LOADED *****
50 FOR I = 1 TO BOOTCOUNT '# OF BYTES IN BOOT CODE
55 READ QS
60 A$ = MID$(QS, 1, 1)
65 GOSUB 7000 'CONVERTS HEX DIGIT TO DECIMAL
70 TEMP = 16 * X 'HANG ON TO UPPER DIGIT
75 A$ = MID$(QS, 2, 1)
80 GOSUB 7000
85 TEMP = TEMP + X
90 BOOTCODE$ = BOOTCODE$ + CHR$(TEMP) 'BUILD BOOT CODE
95 NEXT I
96 REM ***** S-RECORD CONVERSION STARTS HERE *****
97 FILNAM$="BUF34.S19" 'DEFAULT FILE NAME FOR S-RECORDS
100 CLS
105 PRINT "Filename.ext of S-record file to be downloaded (";FILNAM$;") ";
107 INPUT QS
110 IF QS<>" " THEN FILNAM$=QS
120 OPEN FILNAM$ FOR INPUT AS #1
130 PRINT : PRINT "Converting "; FILNAM$; " to binary..."
999 REM ***** SCANS FOR 'S1' RECORDS *****
1000 GOSUB 6000 'GET 1 CHARACTER FROM INPUT FILE
1010 IF FLAG THEN 1250 'FLAG IS EOF FLAG FROM SUBROUTINE
1020 IF A$ <> "S" THEN 1000
1022 GOSUB 6000
1024 IF A$ <> "1" THEN 1000
1029 REM ***** S1 RECORD FOUND, NEXT 2 HEX DIGITS ARE THE BYTE COUNT *****
1030 GOSUB 6000
1040 GOSUB 7000 'RETURNS DECIMAL IN X
1050 BYTECOUNT = 16 * X 'ADJUST FOR HIGH NIBBLE
1060 GOSUB 6000
1070 GOSUB 7000
1080 BYTECOUNT = BYTECOUNT + X 'ADD LOW NIBBLE
1090 BYTECOUNT = BYTECOUNT - 3 'ADJUST FOR ADDRESS + CHECKSUM
1099 REM ***** NEXT 4 HEX DIGITS BECOME THE STARTING ADDRESS FOR THE DATA *****
1100 GOSUB 6000 'GET FIRST NIBBLE OF ADDRESS
1102 GOSUB 7000 'CONVERT TO DECIMAL

```

Listing 2. BASIC Program for Personal Computer

Sheet 2 of 3

```

1104 ADDRESS= 4096 * X
1106 GOSUB 6000 'GET NEXT NIBBLE
1108 GOSUB 7000
1110 ADDRESS= ADDRESS+ 256 * X
1112 GOSUB 6000
1114 GOSUB 7000
1116 ADDRESS= ADDRESS+ 16 * X
1118 GOSUB 6000
1120 GOSUB 7000
1122 ADDRESS= ADDRESS+ X
1124 ARRAYCNT = ADDRESS-ADRSTART 'INDEX INTO ARRAY
1129 REM ***** CONVERT THE DATA DIGITS TO BINARY AND SAVE IN THE ARRAY *****
1130 FOR I = 1 TO BYTECOUNT
1140 GOSUB 6000
1150 GOSUB 7000
1160 Y = 16 * X 'SAVE UPPER NIBBLE OF BYTE
1170 GOSUB 6000
1180 GOSUB 7000
1190 Y = Y + X 'ADD LOWER NIBBLE
1200 CODE%(ARRAYCNT) = Y 'SAVE BYTE IN ARRAY
1210 ARRAYCNT = ARRAYCNT + 1 'INCREMENT ARRAY INDEX
1220 NEXT I
1230 GOTO 1000
1250 CLOSE 1
1499 REM ***** DUMP BOOTLOAD CODE TO PART *****
1500 'OPEN "R",#2,"COM1:1200,N,8,1" 'Macintosh COM statement
1505 OPEN "COM1:1200,N,8,1,CD0,CS0,DS0,RS" FOR RANDOM AS #2 'DOS COM statement
1510 INPUT "Comm port open"; Q$
1512 WHILE LOC(2) > 0 'FLUSH INPUT BUFFER
1513 GOSUB 8020
1514 WEND
1515 PRINT : PRINT "Sending bootload code to target part..."
1520 AS = CHR$(255) + BOOTCODE$ 'ADD HEX FF TO SET BAUD RATE ON TARGET HC11
1530 GOSUB 6500
1540 PRINT
1550 FOR I = 1 TO BOOTCOUNT '# OF BYTES IN BOOT CODE BEING ECHOED
1560 GOSUB 8000
1564 K=ASC(B$):GOSUB 8500
1565 PRINT "Character #"; I; " received = "; HX$
1570 NEXT I
1590 PRINT "Programming is ready to begin.": INPUT "Are you ready"; Q$
1595 CLS
1597 WHILE LOC(2) > 0 'FLUSH INPUT BUFFER
1598 GOSUB 8020
1599 WEND
1600 XMT = 0: RCV = 0 'POINTERS TO XMIT AND RECEIVE BYTES
1610 AS = CHR$(CODE%(XMT))
1620 GOSUB 6500 'SEND FIRST BYTE
1625 FOR I = 1 TO CODESIZE% - 1 'ZERO BASED ARRAY 0 -> CODESIZE-1
1630 AS = CHR$(CODE%(I)) 'SEND SECOND BYTE TO GET ONE IN QUEUE
1635 GOSUB 6500 'SEND IT
1640 GOSUB 8000 'GET BYTE FOR VERIFICATION
1650 RCV = I - 1
1660 LOCATE 10,1:PRINT "Verifying byte #"; I; " "
1664 IF CHR$(RCV) = B$ THEN 1670
1665 K=CODE%(RCV):GOSUB 8500
1666 LOCATE 1,1:PRINT "Byte #"; I; " ", " - Sent "; HX$;
1668 K=ASC(B$):GOSUB 8500
1669 PRINT " Received "; HX$;
1670 NEXT I
1680 GOSUB 8000 'GET BYTE FOR VERIFICATION
1690 RCV = CODESIZE% - 1
1700 LOCATE 10,1:PRINT "Verifying byte #"; CODESIZE%; " "
1710 IF CHR$(RCV) = B$ THEN 1720
1713 K=CODE%(RCV):GOSUB 8500
1714 LOCATE 1,1:PRINT "Byte #"; CODESIZE%; " ", " - Sent "; HX$;
1715 K=ASC(B$):GOSUB 8500
1716 PRINT " Received "; HX$;
1720 LOCATE 8, 1: PRINT : PRINT "Done!!!!"
4900 CLOSE
4910 INPUT "Press [RETURN] to quit...", Q$
5000 END

```

```

5900 '*****
5910 '* SUBROUTINE TO READ IN ONE BYTE FROM A DISK FILE
5930 '* RETURNS BYTE IN A$
5940 '*****
6000 FLAG = 0
6010 IF EOF(1) THEN FLAG = 1: RETURN
6020 A$ = INPUT$(1, #1)
6030 RETURN
6490 '*****
6492 '* SUBROUTINE TO SEND THE STRING IN A$ OUT TO THE DEVICE
6494 '* OPENED AS FILE #2.
6496 '*****
6500 PRINT #2, A$;
6510 RETURN
6590 '*****
6594 '* SUBROUTINE THAT CONVERTS THE HEX DIGIT IN A$ TO AN INTEGER
6596 '*****
7000 X = INSTR(H$, A$)
7010 IF X = 0 THEN FLAG = 1
7020 X = X - 1
7030 RETURN
7990 '*****
7992 '* SUBROUTINE TO READ IN ONE BYTE THROUGH THE COMM PORT OPENED
7994 '* AS FILE #2. WAITS INDEFINITELY FOR THE BYTE TO BE
7996 '* RECEIVED. SUBROUTINE WILL BE ABORTED BY ANY
7998 '* KEYBOARD INPUT. RETURNS BYTE IN B$. USES Q$.
7999 '*****
8000 WHILE LOC(2) = 0 'WAIT FOR COMM PORT INPUT
8005 Q$ = INKEY$: IF Q$ <> "" THEN 4900 'IF ANY KEY PRESSED, THEN ABORT
8010 WEND
8020 B$ = INPUT$(1, #2)
8030 RETURN
8490 '*****
8491 '* DECIMAL TO HEX CONVERSION
8492 '* INPUT: K - INTEGER TO BE CONVERTED
8493 '* OUTPUT: HX$ - TWO CHARACTER STRING WITH HEX CONVERSION
8494 '*****
8500 IF K > 255 THEN HX$="Too big":GOTO 8530
8510 HX$=MID$(H$,K16+1,1) 'UPPER NIBBLE
8520 HX$=HX$+MID$(H$,K MOD 16)+1,1) 'LOWER NIBBLE
8530 RETURN
9499 '***** BOOT CODE *****
9500 DATA 86, 23 'LDAA #S23
9510 DATA B7, 10, 02 'STAA OPT2 make port C wire or
9520 DATA 86, FE 'LDAA #SFE
9530 DATA B7, 10, 03 'STAA PORTC light 1 LED on port C bit 0
9540 DATA C6, FF 'LDAB #SFF
9550 DATA F7, 10, 07 'STAB DDRC make port C outputs
9560 DATA CE, 0F, A0 'LDX #4000 2msec at 2MHz
9570 DATA 18, CE, E0, 00 'LDY #SE000 Start of BUFFALO 3.4
9580 DATA 7E, BF, 00 'JMP #BF00 EPROM routine start address
9590 '*****

```

COMMON BOOTSTRAP MODE PROBLEMS

It is not unusual for a user to encounter problems with bootstrap mode because it is new to many users. By knowing some of the common difficulties, the user can avoid them or at least recognize and quickly correct them.

Reset conditions vs. conditions as bootloaded program starts.

It is common to confuse the reset state of systems and control bits with the state of these systems and control bits when a bootloaded program in RAM starts. Between these times, the bootloader program is executed, which changes the states of some systems and control bits.

- The SCI system is initialized and turned on (Rx and Tx).
- The SCI system has control of the PD0 and PD1 pins.
- Port D outputs are configured for wire-OR operation.

- The stack pointer is initialized to the top of RAM.
- Time has passed (two or more SCI character times).
- Timer has advanced from its reset count value.

Users also forget that bootstrap mode is a special mode; thus privileged control bits are accessible, and write protection for some registers is not in effect. The bootstrap ROM is in the memory map. The DISR bit in the TEST1 control register is set, which disables resets from the COP and clock monitor systems.

Since bootstrap is a special mode, these conditions can be changed by software. The bus can even be switched from single-chip mode to expanded mode to gain access to external memories and peripherals.

Connecting RxD to V_{SS} does not cause the SCI to receive a break — To force an immediate jump to the start of EEPROM, the bootstrap firmware looks for the first received

Table 2. Summary of Boot-ROM-Related Features

MCU Part #	BOOT ROM Revision (@\$BFD1)	Mask Set I.D. (@\$BFD2,3)	MCU Type I.D. (@\$BFD4,5)	Security	Download Length	JMP on BRK or \$00 ¹	JMP to RAM ²	Default RAM Location	EPROM ³ PROGRAM Utility	UPLOAD ⁴ Utility	Notes
MC68HC11A0	—	—	Mask Set #	—	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68HC11A1	—	—	Mask Set #	—	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68HC11A8	—	—	Mask Set #	—	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68SEC11A8	—	—	Mask Set #	Yes	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68HC11D3	\$00	Mask Set #	\$11D3	—	0–192	\$F000-ROM	—	\$0040–FF	—	—	6
MC68HC711D3	\$42 (B)	\$0000	\$71D3	—	0–192	\$F000-EPROM	—	\$0040–FF	Yes	Yes	6
MC68HC811E2	—	\$0000	\$E2E2	—	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68SEC811E2	—	—	\$E25C	Yes	256	\$B600	\$0000	\$0000–FF	—	—	5
MC68HC11E0	—	Mask Set #	\$E9E9	—	0–512	\$B600	—	\$0000–1FF	—	—	5
MC68HC11E1	—	Mask Set #	\$E9E9	—	0–512	\$B600	—	\$0000–1FF	—	—	5
MC68HC11E9	—	Mask Set #	\$E9E9	—	0–512	\$B600	—	\$0000–1FF	—	—	5
MC68SEC11E9	—	Mask Set #	\$E95C	Yes	0–512	\$B600	—	\$0000–1FF	—	—	5
MC68HC711E9	\$41 (A)	\$0000	\$71E9	—	0–512	\$B600	—	\$0000–1FF	Yes	Yes	6
MC68HC11F1	\$42 (B)	\$0000	\$F1F1	—	0–1024	\$FE00	—	\$0000–3FF	—	—	6, 8
MC68HC11K4	\$30 (0)	Mask Set #	\$044B	—	0–768	\$0D80	—	\$0080–37F	—	—	6, 8
MC68HC711K4	\$42 (B)	\$0000	\$744B	—	0–768	\$0D80	—	\$0080–37F	Yes	Yes	6, 8

NOTES:

1. By sending \$00 or a break as the first SCI character after reset in bootstrap mode, a jump (JMP) is executed to the address in this table rather than doing a download. Unless otherwise noted, this address is the start of EEPROM. Tying RxD to TxD and using a pullup resistor from TxD to V_{DD} will cause the SCI to see a break as the first received character.
2. If \$55 is received as the first character after reset in bootstrap mode, a jump (JMP) is executed to the start of on-chip RAM rather than doing a download. This \$55 character must be sent at the default baud rate (7812 baud @ E = 2MHz).

For devices with variable-length download, the same effect can be achieved by sending \$FF and no other SCI characters. After four SCI character times, the download terminates, and a jump (JMP) to the start of RAM is executed.

The jump to RAM feature is only useful if the RAM was previously loaded with a meaningful program.

3. A callable utility subroutine is included in the bootstrap ROM of the indicated versions to program bytes of on-chip EPROM with data received via the SCI.
4. A callable utility subroutine is included in the bootstrap ROM of the indicated versions to upload contents of on-chip memory to a host computer via the SCI.
5. The complete listing for this bootstrap ROM may be found in the M68HC11RM/AD, M68HC11 Reference Manual.
6. The complete listing for this bootstrap ROM is included in this application note.
7. Due to the extra program space needed for EEPROM security on this device, there are no pseudo-vectors for SCI, SPI, PAIF, PAOVF, TOF, OC5F, or OC4F interrupts.
8. This bootloader extends the automatic software detection of baud rates to include 9600 baud at 2-MHz E-clock rate.

character to be \$00 (or break). The data reception logic in the SCI looks for a one-to-zero transition on the RxD pin to synchronize to the beginning of a receive character. If the RxD pin is tied to ground, no one-to-zero transition occurs. The SCI transmitter sends a break character when the bootloader firmware starts, and this break character can be fed back to the RxD pin to cause the jump to EEPROM. Since TxD is configured as an open-drain output, a pullup resistor is required.

An \$FF character is required before data is loaded into RAM — The initial character (usually \$FF) that sets the download baud rate is often forgotten.

Original M68HC11 versions required exactly 256 bytes to be downloaded to RAM — Even users that know about the 256 bytes of download data sometimes forget the initial \$FF that makes the total number of bytes required for the entire download operation equal to 256 + 1 or 257 bytes.

Variable-length download — When on-chip RAM surpassed 256 bytes, the time required to serially load this many characters became more significant. The variable-length download feature allows shorter programs to be loaded without sacrificing compatibility with earlier fixed-length download versions of the bootloader. The end of a download is indicated by an idle RxD line for at least four character times. If a personal computer is being used to send the download data to the MCU, there can be problems keeping characters close enough together to avoid tripping the end-of-download detect mechanism. Using 1200 as the baud rate rather than the faster default rate may help this problem.

Assemblers often produce S-record encoded programs which must be converted to binary before bootloading them to the MCU. The process of reading S-record data from a file and translating it to binary can be slow, depending on the personal computer and the programming language used for the translation. One strategy that can be used to overcome this problem is to translate the file into binary and store it into a RAM array before starting the download process. Data can then be read and downloaded without the translation or file-read delays.

The end-of-download mechanism goes into effect when the initial \$FF is received to set the baud rate. Any amount of time may pass between reset and when the \$FF is sent to start the download process.

EPROM/OTP versions of M68HC11 have an EPROM emulation mode — The conditions that configure the MCU for EPROM emulation mode are essentially the same as those for resetting the MCU in bootstrap mode. While RESET is low and mode select pins are configured for bootstrap mode (low), the MCU is configured for EPROM emulation mode.

The port pins that are used for EPROM data I/O lines may be inputs or outputs, depending on the pin that is emulating the

EPROM output enable pin (\overline{OE}). To make these data pins appear as high-impedance inputs as they would on a non-EPROM part in reset, connect the PB7/ \overline{OE} pin to a pull-up resistor.

Bootloading a program to perform a ROM checksum — The bootloader ROM must be turned off before performing the checksum program. To remove the boot ROM from the memory map, clear the RBOOT bit in the HPRIO register. This is normally a write-protected bit that is zero, but in bootstrap mode it is reset to one and can be written. If the boot ROM is not disabled, the checksum routine will read the contents of the boot ROM rather than the user's mask ROM or EPROM at the same addresses.

Inherent delays caused by double buffering of SCI data — This problem is troublesome in cases where one MCU is bootloading to another MCU.

Because of transmitter double buffering, there may be one character in the serial shifter as a new character is written into the transmit data register. In cases such as downloading in which this two-character pipeline is kept full, a two-character time delay occurs between when a character is written to the transmit data register and when that character finishes transmitting. A little more than one more character time delay occurs between the target MCU receiving the character and echoing it back. If the master MCU waits for the echo of each downloaded character before sending the next one, the download process takes about twice as long as it would if transmission is treated as a separate process or if verify data is ignored.

BOOT ROM VARIATIONS

Different versions of the M68HC11 have different versions of the bootstrap ROM program. Table 2 summarizes the features of the boot ROMs in 16 members of the M68HC11 Family.

The boot ROMs for the MC68HC11F1, the MC68HC711K4, and the MC68HC11K4 allow additional choices of baud rates for bootloader communications. For the three new baud rates, the first character used to determine the baud rate is not \$FF as it was in earlier M68HC11s. The intercharacter delay that terminates the variable-length download is also different for these new baud rates. Table 3 shows the synchronization characters, delay times, and baud rates as they relate to E-clock frequency.

COMMENTED BOOT ROM LISTINGS

Listings 3-8 contain complete commented listings of the boot ROM programs in six specific versions of the M68HC11. Other versions can be found in appendix B of the M68HC11RM/AD, *M68HC11 Reference Manual*.

Table 3. Bootloader Baud Rates

Sync Character	Timeout Delay	Baud Rates at E-clock =					
		2 MHz	2.1 MHz	3 MHz	3.15 MHz	4 MHz	4.2 MHz
\$FF	4 Characters	7812	8192	11,718	12,288	15,624	16,838
\$FF	4 Characters	1200	1260	1800	1890	2400	2520
\$F0	4.9 Characters	9600	10,080	14,400	15,120	19,200	20,160
\$FD	17.3 Characters	5208	5461	7812	8192	10,416	10,922
\$FD	13 Characters	3906	4096	5859	6144	7812	8192

Listing 3. MC68HC711E9 Bootloader ROM

Sheet 1 of 4

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

```

```

*****
* BOOTLOADER FIRMWARE FOR 68HC711E9 - 21 Aug 89
*****
* Features of this bootloader are...
*
* Auto baud select between 7812.5 and 1200 (8 MHz)
* 0 - 512 byte variable length download
* Jump to EEPROM at $B600 if 1st download byte = $00
* PROGRAM - Utility subroutine to program EPROM
* UPLOAD - Utility subroutine to dump memory to host
* Mask I.D. at $BFD4 = $71E9
*****
* Revision A -
*
* Fixed bug in PROGRAM routine where the first byte
* programmed into the EPROM was not transmitted for
* verify.
* Also added to PROGRAM routine a skip of bytes
* which were already programmed to the value desired.
*
* This new version allows variable length download
* by quitting reception of characters when an idle
* of at least four character times occurs
*
*****
* EQUATES FOR USE WITH INDEX OFFSET = $1000
*
PORTD          EQU      $08
TCNT           EQU      $0E
TOC1           EQU      $16
TFLG1         EQU      $23
* BIT EQUATES FOR TFLG1
OC1F           EQU      $80
*
SPCR           EQU      $28          (FOR DWOM BIT)
BAUD           EQU      $2B
SCCR2         EQU      $2D
SCSR          EQU      $2E
SCDAT         EQU      $2F
PPROG         EQU      $3B
* BIT EQUATES FOR PPROG
ELAT          EQU      $20
EPGM          EQU      $01
*
* MEMORY CONFIGURATION EQUATES
*
EEPSTR        EQU      $B600        Start of EEPROM
EPMEND       EQU      $B7FF        End of EEPROM
*
EPRMSTR      EQU      $D000        Start of EPROM
EPRMEND     EQU      $FFFF        End of EPROM
*
RAMSTR       EQU      $0000
RAMEND      EQU      $01FF
*
* DELAY CONSTANTS
*
DELAYS       EQU      3504        Delay at slow baud
DELAYF      EQU      539         Delay at fast baud
*
PROGDEL      EQU      4200        2 ms programming delay
*
At 2.1 MHz

```

Listing 3. MC68HC711E9 Bootloader ROM

Sheet 2 of 4

```

66
67 BF00          *****
68              ORG      $BF00
69              *****
70
71              * Next two instructions provide a predictable place
72              * to call PROGRAM and UPGLOAD even if the routines
73              * change size in future versions.
74
75 BF00 7EBF13   PROGRAM      JMP      PRGROUT      EPROM programming utility
76 BF03          UPGLOAD      EQU      *          Upload utility
77
78              *****
79              * UPGLOAD - Utility subroutine to send data from
80              * inside the MCU to the host via the SCI interface.
81              * Prior to calling UPGLOAD set baud rate, turn on SCI
82              * and set Y=first address to upload.
83              * Bootloader leaves baud set, SCI enabled, and
84              * Y pointing at EPROM start ($D000) so these default
85              * values do not have to be changed typically.
86              * Consecutive locations are sent via SCI in an
87              * infinite loop. Reset stops the upload process.
88              *****
89 BF03 CE1000   UPGLOOP      LDX      #$1000      Point to internal registers
90 BF06 18A600   LDAA     0,Y          Read byte
91 BF09 1F2E80FC BRCLR   SCSR,X $80 *   Wait for TDRE
92 BF0D A72F     STAA     SCDAT,X      Send it
93 BF0F 1808     INY
94 BF11 20F3     BRA      UPGLOOP     Next...
95
96              *****
97              * PROGRAM - Utility subroutine to program EPROM.
98              * Prior to calling PROGRAM set baud rate, turn on SCI
99              * set X=2ms prog delay constant, and set Y=first
100             * address to program. SP must point to RAM.
101             * Bootloader leaves baud set, SCI enabled, X=4200
102             * and Y pointing at EPROM start ($D000) so these
103             * default values don't have to be changed typically.
104             * Delay constant in X should be equivalent to 2 ms
105             * at 2.1 MHz X=4200; at 1 MHz X=2000.
106             * An external voltage source is required for EPROM
107             * programming.
108             * This routine uses 2 bytes of stack space
109             * Routine does not return. Reset to exit.
110             *****
111 BF13          PRGROUT      EQU      *
112 BF13 3C       PSHX
113 BF14 CE1000   LDX      #$1000      Save program delay constant
114                             Point to internal registers
115
116             * Send $FF to indicate ready for program data
117 BF17          *
118 BF17 1F2E80FC BRCLR   SCSR,X $80 *   Wait for TDRE
119 BF1B 86FF     LDAA     #$FF
120 BF1D A72F     STAA     SCDAT,X
121
122 BF1F          WAIT1      EQU      *
123 BF1F 1F2E20FC BRCLR   SCSR,X $20 *   Wait for RDRF
124 BF23 E62F     LDAB    SCDAT,X      Get received byte
125 BF25 18E100  CMPB    $0,Y          See if already programmed
126 BF28 271D     BEQ     DONEIT        If so, skip prog cycle
127 BF2A 8620     LDAA     #ELAT        Put EPROM in prog mode
128 BF2C A73B     STAA     PPROG,X
129 BF2E 18E700  STAB    0,Y          Write the data
130 BF31 8621     LDAA     #ELAT+EPGM
131 BF33 A73B     STAA     PPROG,X      Turn on prog voltage
132 BF35 32      PULA    Pull delay constant
133 BF36 33      PULB    into D-reg
134 BF37 37      PSHB   But also keep delay
135 BF38 36      PSHA   keep delay on stack
136 BF39 E30E     ADDD   TCNT,X        Delay const + present TCNT
137 BF3B ED16     STD    TOCL,X        Schedule OCl (2ms delay)
138 BF3D 8680     LDAA     #OC1F
139 BF3F A723     STAA     TFLG1,X      Clear any previous flag
140
141 BF41 1F2380FC BRCLR   TFLG1,X OC1F *   Wait for delay to expire
142 BF45 6F3B     CLR     PPROG,X      Turn off prog voltage
143
144 BF47          *
145 BF47          DONEIT     EQU      *

```

Listing 3. MC68HC711E9 Bootloader ROM

Sheet 3 of 4

```

143 BF47 1F2E80FC          BRCLR  SCSR,X $80 *      Wait for TDRE
144 BF4B 18A600           LDAA  $0,Y              Read from EPROM and...
145 BF4E A72F            STAA  SCDAT,X          Xmit for verify
146 BF50 1808           INY   *                Point at next location
147 BF52 20CB           BRA   WAIT1            Back to top for next
148
149 * Loops indefinitely as long as more data sent.
150
151 *****
152 * Main bootloader starts here
153 *****
154 * RESET vector points to here
155
156 BF54          BEGIN          EQU   *
157 BF54 8E01FF          LDS   #RAMEND          Initialize stack ptrr
158 BF57 CE1000          LDX  #$1000            Point at internal regs
159 BF5A 1C2820          BSET SPCR,X $20        Select port D wire-OR mode
160 BF5D CCA20C          LDD  #$A20C            BAUD in A, SCCR2 in B
161 BF60 A72B           STAA  BAUD,X           SCPx = +4, SCRx = +4
162 * Writing 1 to MSB of BAUD resets count chain
163 BF62 E72D           STAB  SCCR2,X          Rx and Tx Enabled
164 BF64 CC021B          LDD  #DELAYF           Delay for fast baud rate
165 BF67 ED16           STD  TOC1,X            Set as default delay
166
167 * Send BREAK to signal ready for download
168 BF69 1C2D01          BSET SCCR2,X $01        Set send break bit
169 BF6C 1E0801FC        BRSET PORTD,X $01 *     Wait for RxD pin to go low
170 BF70 1D2D01          BRCLR SCCR2,X $01       Clear send break bit
171 BF73
172 BF73 1F2E20FC        BRCLR SCSR,X $20 *      Wait for RDRF
173 BF77 A62F           LDAA  SCDAT,X          Read data
174 * Data will be $00 if BREAK OR $00 received
175 BF79 26C3           BNE   NOTZERO          Bypass JMP if not 0
176 BF7B EB60C           JMP  EPROMSTR           Jump to EPROM if it was 0
177 BF7E NOTZERO          EQU   *
178 BF7E 81FF           CMPA  #$FF             $FF will be seen as $FF
179 BF80 2708           BEQ  BAUDOK            If baud was correct
180 * Or else change to +104 (+13 & +8) 1200 @ 2MHZ
181 BF82 1C2B33          BSET SCCR2,X $33        Works because $22 -> $33
182 BF85 CC0DB0          LDD  #DELAYS           And switch to slower...
183 BF88 ED16           STD  TOC1,X            delay constant
184 BF8A          BAUDOK          EQU   *
185 BF8A 18CE0000        LDY  #RAMSTR           Point at start of RAM
186
187 BF8E          WAIT          EQU   *
188 BF8E EC16           LDD  TOC1,X            Move delay constant to D
189 BF90          WTLOOP         EQU   *
190 BF90 1E2E2007        BRSET SCSR,X $20 NEWONE Exit loop if RDRF set
191 BF94 8F           XGDX *                Swap delay count to X
192 BF95 C9           DEX  *                Decrement count
193 BF96 8F           XGDX *                Swap back to D
194 BF97 26F7          BNE  WTLOOP           Loop if not timed out
195 BF99 200F          BRA  STAR             Quit download on timeout
196
197 BF9B          NEWONE        EQU   *
198 BF9B A62F           LDAA SCDAT,X           Get received data
199 BF9D 18A700          STAA $00,Y            Store to next RAM location
200 BFA0 A72F           STAA SCDAT,X          Transmit it for handshake
201 BFA2 1808           INY  *                Point at next RAM location
202 BFA4 188C0200        CPY  #RAMEND+1         See if past end
203 BFA8 26E4           BNE  WAIT             If not, Get another
204
205 BFAA          STAR          EQU   *
206 BFAA CE1068          LDX  #PROGDEL          Init X with programming delay
207 BFAD 18CED000        LDY  #EPRMSTR          Init Y with EPROM start addr
208 BFB1 7E0000          JMP  RAMSTR            ** EXIT to start of RAM **
209 BFB4
210 *****
211 * Block fill unused bytes with zeros
212 *****
213 BFB4 000000000000          BSZ  $BFD1-*
214 000000000000
215 000000000000
216 000000000000
217 000000000000
218
219 *****
220 * Boot ROM revision level in ASCII
221 * (ORG $BFD1)
222 BFD1 41          FCC  "A"

```

Listing 3. MC68HC711E9 Bootloader ROM

Sheet 4 of 4

```

218 *****
219 * Mask set I.D. ($0000 FOR EPROM PARTS)
220 * (ORG $BFD2) FDB $0000
221 BFD2 0000 *****
222 *****
223 * '711E9 I.D. - Can be used to determine MCU type
224 * (ORG $BFD4) FDB $71E9
225 BFD4 71E9 *****
226 *****
227 * VECTORS - point to RAM for pseudo-vector JUMPs
228 *****
229 *****
230 BFD6 00C4 FDB $100-60 SCI
231 BFD8 00C7 FDB $100-57 SPI
232 BFDA 00CA FDB $100-54 PULSE ACCUM INPUT EDGE
233 BFDC 00CD FDB $100-51 PULSE ACCUM OVERFLOW
234 BFDE 00D0 FDB $100-48 TIMER OVERFLOW
235 BFE0 00D3 FDB $100-45 TIMER OUTPUT COMPARE 5
236 BFE2 00D6 FDB $100-42 TIMER OUTPUT COMPARE 4
237 BFE4 00D9 FDB $100-39 TIMER OUTPUT COMPARE 3
238 BFE6 00DC FDB $100-36 TIMER OUTPUT COMPARE 2
239 BFE8 00DF FDB $100-33 TIMER OUTPUT COMPARE 1
240 BFEA 00E2 FDB $100-30 TIMER INPUT CAPTURE 3
241 BFEC 00E5 FDB $100-27 TIMER INPUT CAPTURE 2
242 BFEE 00E8 FDB $100-24 TIMER INPUT CAPTURE 1
243 BFF0 00EB FDB $100-21 REAL TIME INT
244 BFF2 00EE FDB $100-18 IRQ
245 BFF4 00F1 FDB $100-15 XIRQ
246 BFF6 00F4 FDB $100-12 SWI
247 BFF8 00F7 FDB $100-9 ILLEGAL OP-CODE
248 BFFA 00FA FDB $100-6 COP FAIL
249 BFFC 00FD FDB $100-3 CLOCK MONITOR
250 BFFE BF54 FDB BEGIN RESET
251 C000 END

```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUD	002B	*00037	00160	00180
BAUDOK	BF8A	*00183	00178	
BEGIN	BF54	*00155	00250	
DELAYF	021B	*00061	00163	
DELAYS	0DB0	*00060	00181	
DONEIT	BF47	*00142	00124	
EPPMEND	B7FF	*00050		
EPPMSTR	B600	*00049	00175	
ELAT	0020	*00043	00125	00128
EPGM	0001	*00044	00128	
EPRMEND	FFFF	*00053		
EPRMSTR	D000	*00052	00206	
NEWONE	BF9B	*00196	00189	
NOTZERO	BF7E	*00176	00174	
OCIF	0080	*00034	00136	00139
PORTD	0008	*00029	00168	
PPROG	003B	*00041	00126	00129 00140
PRGROUT	BF13	*00110	00074	
PROGDEL	1068	*00063	00205	
PROGRAM	BF00	*00074		
RAMEND	01FF	*00056	00156	00201
RAMSTR	0000	*00055	00184	00207
SCCR2	002D	*00038	00162	00167 00169
SCDAT	002F	*00040	00091	00118 00122 00145 00172 00197 00199
SCSR	002E	*00039	00090	00116 00121 00143 00171 00189
SPCR	0028	*00036	00158	
STAR	BFAA	*00204	00194	
TCNT	000E	*00030	00134	
TFLG1	0023	*00032	00137	00139
TOC1	0016	*00031	00135	00164 00182 00187
UPLOAD	BF03	*00075		
UPLLOOP	BF06	*00089	00093	
WAIT	BF8E	*00186	00202	
WAIT1	BF1F	*00120	00147	
WTLOOP	BF90	*00188	00193	

Errors: None

Labels: 35

Last Program Address: \$BFFF

Last Storage Address: \$0000

Program Bytes: \$0100 256

Storage Bytes: \$0000 0

Listing 4. MC68HC11D3 Bootloader ROM

Sheet 1 of 3

```

1          * *****
2          * BOOTLOADER FIRMWARE FOR MC68HC11D3 - 13 Apr 89
3          * *****
4          * Features of this bootloader are...
5          *
6          * Auto baud select between 7812 and 1200 (E = 2 MHz).
7          * 0 - 192 byte variable length download:
8          *   reception of characters quits when an idle of at
9          *   least four character times occurs.
10         * Jump to EPROM at $F000 if first download byte = $00.
11         * PROGRAM - Utility subroutine to program EPROM.
12         * UPLOAD - Utility subroutine to dump memory to host.
13         * Part I.D. at $BFD4 is $71D3.
14         * *****
15
16         * Equates (registers in direct space)
17         *
18 0008     PORTD           EQU    $08
19 0009     DDRD           EQU    $09
20 000E     TCNT          EQU    $0E
21 0016     TOC1          EQU    $16
22 0023     TFLG1         EQU    $23
23         * Bit equates for TFLG1
24 0080     OC1F          EQU    $80
25         *
26 0028     SPCR          EQU    $28           (For DWOM bit)
27 002B     BAUD          EQU    $2B
28 002C     SCCR1         EQU    $2C
29 002D     SCCR2         EQU    $2D
30 002E     SCSR          EQU    $2E
31 002F     SCDAT         EQU    $2F
32 003B     PPROG         EQU    $3B
33         * Bit equates for PPROG
34 0020     LAT           EQU    $20
35 0001     EPGM          EQU    $01
36         *
37 003E     TEST1         EQU    $3E
38 003F     CONFIG        EQU    $3F
39
40
41         * Memory configuration equates
42         *
43 F000     ROMSTR         EQU    $F000       Start of ROM
44 FFFF     ROMEND        EQU    $FFFF       End of ROM
45         *
46 0040     RAMSTR         EQU    $0040       Start of RAM
47 00FF     RAMEND        EQU    $00FF       End of RAM
48
49         * Delay constants
50         *
51 0DB0     DELAYS         EQU    3504       Delay at slow baud
52 021B     DELAYF         EQU    539        Delay at fast baud
53
54
55         * *****
56 BF40     ORG            SBF40
57         * *****
58         * Main bootloader starts here
59         * *****
60         * RESET vector points to here
61
62 BF40     BEGIN          EQU    *
63 BF40 8E00FF     LDS      #RAMEND          Initialize stack ptr
64 BF43 142820     BSET     SPCR $20         Select port D wire-OR mode
65 BF46 CCA20C     LDD      #A20C           Baud in A, SCCR2 in B
66 BF49 972B      STAA     BAUD             SCFX = /4, SCRX = /4
67
68 BF4B D72D      * Writing 1 to MSB of BAUD resets count chain
69 BF4D CC021B     STAB     SCCR2           Rx and Tx enabled
70 BF50 DD16      LDD      #DELAYF         Delay for fast baud rate
71
72
73 BF52 142D01     STD      TOC1           Set as default delay
74
75         * Send BREAK to signal ready for download
76         *
77 BF52 142D01     BSET     SCCR2 $01       Set send break bit
78 BF55 120801FC  BRSET   PORTD $01 *                   Wait for RxD pin to go low
79 BF59 152D01     BCLR    SCCR2 $01       Clear send break bit
80
81 BF5C 132E20FC  BRCLR   SCSR $20 *                   Wait for RDRF
82 BF60 962F      LDAA     SCDAT           Read data
83
84         * Data will be $00 if BREAK or $00 received
85 BF62 2603     BNE      NOTZERO          Bypass jump if not $00

```

```

81 BF64 7EF000          JMP    ROMSTR          Jump to ROM if it was $00
82 BF67                EQU    *
83 BF67 81FF          NOTZERO    CMPA    #SFF          SFF will be seen as SFF...
84 BF69 2708          BEQ    BAUDOK         if baud was correct
85                * Or else change to /104 (/13 & /8) 1200 @ 2MHz
86 BF6B 142B33        BSET    BAUD $33      Works because $22 -> $33
87 BF6E CC0DB0        LDD    #DELAYS       And switch to slower...
88 BF71 DD16          STD    TOC1          delay constant
89 BF73                EQU    *
90 BF73 18CE0040      BAUDOK    LDY    #RAMSTR       Point to start of RAM
91
92 BF77                WAIT      EQU    *
93 BF77 DE16          LDY    TOC1          Move delay constant to X
94 BF79                WTLOOP   EQU    *
95 BF79 122E2009      BRSET   SCSR $20 NEWONE Exit loop if RDRF set
96 BF7D 09            DEX
97 BF7E 01            NOP
98 BF7F 01            NOP
99 BF80 2100          BRN    *+2           ..to match original program
100 BF82 26F5         BNE    WTLOOP       Loop if not timed out
101 BF84 200F         BRA    STAR         Quit download on timeout
102
103 BF86                NEWONE   EQU    *
104 BF86 962F         LDAA   SCDAT        Get received data
105 BF88 18A700       STAA   $00,Y        Store to next RAM location
106 BF8B 972F         STAA   SCDAT        Transmit it for handshake
107 BF8D 1808        INY
108 BF8F 188C0100     CPY    #RAMEND+1    Point to next RAM location
109 BF93 26E2         BNE    WAIT        See if past end
110                If not, get another
111 BF95                STAR    EQU    *
112 BF95 7E0040       JMP    RAMSTR       ** Exit to start of RAM **
113                *****
114                * Block fill unused bytes with zero
115
116 BF98 000000000000 BSZ    $BFD1-*
117                000000000000
118                000000000000
119                000000000000
120                000000000000
121                000000000000
122                000000000000
123                000000000000
124                000000000000
125                000000
126
127                *****
128                * Boot ROM revision level in ASCII
129 BF98 00          (ORG    $BFD1)    FCB    0
130                *****
131                * Mask set I.D. -
132                (ORG    $BFD2)    FDB    $0000
133                *****
134                * 11D3 I.D. - can be used to determine MCU type
135 BF98 11D3       (ORG    $BFD4)    FDB    $11D3
136                *****
137                * VECTORS - point to RAM for pseudo-vector JUMPS
138 BF98 00C4        FDB    $100-60      SCI
139 BF98 00C7        FDB    $100-57      SPI
140 BF98 00CA        FDB    $100-54      PULSE ACCUM INPUT EDGE
141 BF98 00CD        FDB    $100-51      PULSE ACCUM OVERFLOW
142 BF98 00D0        FDB    $100-48      TIMER OVERFLOW
143 BF98 00D3        FDB    $100-45      TIMER OUTPUT COMPARE 5
144 BF98 00D6        FDB    $100-42      TIMER OUTPUT COMPARE 4
145 BF98 00D9        FDB    $100-39      TIMER OUTPUT COMPARE 3
146 BF98 00DC        FDB    $100-36      TIMER OUTPUT COMPARE 2
147 BF98 00DF        FDB    $100-33      TIMER OUTPUT COMPARE 1
148 BF98 00E2        FDB    $100-30      TIMER INPUT CAPTURE 3
149 BF98 00E5        FDB    $100-27      TIMER INPUT CAPTURE 2
150 BF98 00E8        FDB    $100-24      TIMER INPUT CAPTURE 1
151 BF98 00EB        FDB    $100-21      REAL TIME INT
152 BF98 00EE        FDB    $100-18      IRQ
153 BF98 00F1        FDB    $100-15      XIRQ
154 BF98 00F4        FDB    $100-12      SWI
155 BF98 00F7        FDB    $100-9       ILLEGAL OP-CODE
156 BF98 00FA        FDB    $100-6       COP FAIL

```

Listing 4. MC68HC11D3 Bootloader ROM

Sheet 3 of 3

```

152 BFFC 00FD          FDB $100-3      CLOCK MONITOR
153 BFFE BF40          FDB BEGIN       RESET
154 C000              END
    
```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUD	002B	*00027	00066	00086
BAUDOK	BF73	*00089	00084	
BEGIN	BF40	*00062	00153	
CONFIG	003F	*00038		
DDRD	0009	*00019		
DELAYF	021B	*00052	00069	
DELAYS	0DB0	*00051	00087	
EPGM	0001	*00035		
LAT	0020	*00034		
NEWONE	BF86	*00103	00095	
NOTZERO	BF67	*00082	00080	
OC1F	0080	*00024		
PORTD	0008	*00018	00074	
PPIRG	003B	*00032		
RAMEND	00FF	*00047	00063	00108
RAMSTR	0040	*00046	00090	00112
ROMEND	FFFF	*00044		
ROMSTR	F000	*00043	00081	
SCCR1	002C	*00028		
SCCR2	002D	*00029	00068	00073 00075
SCDAT	002F	*00031	00078	00104 00106
SCSR	002E	*00030	00077	00095
SPCR	0028	*00026	00064	
STAR	BF95	*00111	00101	
TCNT	000E	*00020		
TEST1	003E	*00037		
TFLG1	0023	*00022		
TOC1	0016	*00021	00070	00088 00093
WAIT	BF77	*00092	00109	
WTLOOP	BF79	*00094	00100	

```

Errors: None
Labels: 30
Last Program Address: $BFFF
Last Storage Address: $0000
Program Bytes: $00C0 192
Storage Bytes: $0000 0
    
```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30 0008
31 0009
32 000E
33 0016
34 0023
35
36 0080
37
38 0028
39 002B
40 002C
41 002D
42 002E
43 002F
44 003B
45
46 0020
47 0001
48
49 003E
50 003F
51
52
53
54
55 F000
56 FFFF
57
58 0040
59 00FF
60
61
62
63 0DB0
64 021B
65
66 1068
67
68
69
70 BF00
71
72
73
74
75
76
77 BF00 7EBF10
78 BF03
79
*****
* BOOTLOADER FIRMWARE FOR MC68HC711D3 - 28 Aug 90
*****
* Features of this bootloader are...
*
* Auto baud select between 7812 and 1200 (E = 2 MHz).
* 0 - 192 byte variable length download:
*   reception of characters quits when an idle of at
*   least four character times occurs.
* Jump to EPROM at $F000 if first download byte = $00.
* PROGRAM - Utility subroutine to program EPROM.
* UPLOAD - Utility subroutine to dump memory to host.
* Part I.D. at $BFD4 is $71D3.
*****
* Revision B -
*
* Changed program delay to 2 mSec at E = 2 MHz.
*****
* Revision A -
*
* Fixed bug in PROGRAM routine where the first byte
* programmed into the EPROM was not transmitted for
* verify.
* Also added to PROGRAM routine a skip of bytes
* which were already programmed to the value desired.
*****
* Equates (registers in direct space) -
*
PORTD      EQU    $08
DDRD      EQU    $09
TCNT      EQU    $0E
TOCL      EQU    $16
TFLG1     EQU    $23
* Bit equates for TFLG1
OC1F      EQU    $80
*
SPCR      EQU    $28          (For DWOM bit)
BAUD      EQU    $2B
SCCR1     EQU    $2C
SCCR2     EQU    $2D
SCSR      EQU    $2E
SCDAT     EQU    $2F
PPROG     EQU    $3B
* Bit equates for PPROG
LAT       EQU    $20
EPGM      EQU    $01
*
TEST1     EQU    $3E
CONFIG    EQU    $3F
*
* Memory configuration equates
*
EPRMSTR   EQU    $F000      Start of EPROM
EPRMEND   EQU    $FFFF      End of EPROM
*
RAMSTR    EQU    $0040      Start of RAM
RAMEND    EQU    $00FF      End of RAM
*
* Delay constants
*
DELAYS    EQU    3504       Delay at slow baud
DELAYF    EQU    539       Delay at fast baud
*
PROGDEL   EQU    4200       2 mSec programming delay
*
*****
ORG       SBF00
*****
* Next two instructions provide a predictable place
* to call PROGRAM and UPLOAD even if the routines
* change size in future versions.
*
PROGRAM   JMP    PRGROUT    EPROM programming utility
UPLOAD    EQU    *          Upload utility

```


Listing 5. MC68HC711D3 Bootloader ROM

Sheet 2 of 4

```

80
81
82
83
84
85
86
87
88
89
90
91 BF03          UPLoop      EQU      *
92 BF03 18A600   LDAA     0,Y          Read byte
93 BF06 132E80FC BRCLR   SCSR $80 *   Wait for TDRE
94 BF0A 972F     STAA    SCDAT        Send it
95 BF0C 1808     INY
96 BF0E 20F3     BRA     UPLoop      Next....
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113 BF10          PRGROUT   EQU      *
114
115 BF10 132E80FC * Send $FF to indicate ready for program data
116 BF14 86FF     BRCLR   SCSR $80 *   Wait for TDRE
117 BF16 972F     LDAA    #$FF
118              STAA    SCDAT
119
120 BF18          WAIT1      EQU      *
121 BF18 132E20FC BRCLR   SCSR $20 *   Wait for RDRF
122 BF1C D62F     LDAB    SCDAT        Get received byte
123 BF1E 18E100   CMPB    $0,Y         See if already programmed
124 BF21 271D     BEQ    DONEIT        If so, skip prog cycle
125 BF23 8620     LDAA    #LAT         Put EPROM in prog mode
126 BF25 973B     STAA    PPROG        Write data
127 BF27 18E700   STAB    0,Y
128 BF2A 8621     LDAA    #LAT+EPGM    Turn on prog voltage
129 BF2C 973B     STAA    PPROG        Save delay on stack
130 BF2E 3C       PSHX
131 BF2F 8F       XGDX
132 BF30 38       PULX
133 BF31 D30E     ADDD    TCNT         Delay const + present TCNT
134 BF33 DD16     STD     TOC1         Schedule OC1 (prog delay)
135 BF35 8680     LDAA    #OC1F
136 BF37 9723     STAA    TFLG1        Clear any previous flag
137 BF39 132380FC BRCLR   TFLG1 OC1F * Wait for delay to expire
138 BF3D 7F003B   CLR     PPROG        Turn off prog voltage
139 BF40
140 BF40 132E80FC DONEIT    BRCLR   SCSR $80 *   Wait for TDRE
141 BF44 18A600   LDAA    $0,Y         Read from EPROM and...
142 BF47 972F     STAA    SCDAT        Xmit for verify
143 BF49 1808     INY
144 BF4B 20CB     BRA     WAIT1        Point to next location
145
146
147
148
149
150
151
152 BF4D          BEGIN      EQU      *
153 BF4D 8E00FF   LDS     #FRAMEND     Initialize stack pntr
154 BF50 142820   BSET    SPCR $20     Select port D wire-OR mode
155 BF53 CCA20C   LDD    #$A20C        Baud in A, SCCR2 in B
156 BF56 972B     STAA    BAUD         SCPx = /4, SCRx = /4
157
158 BF58 D72D     * Writing 1 to MSB of BAUD resets count chain
159 BF5A CC021B   STAB    SCCR2        Rx and Tx enabled
                      LDD     #DELAY     Delay for fast baud rate

```

```

160 BF5D DD16          STD   TOC1          Set as default delay
161
162          * Send BREAK to signal ready for download
163 BF5F 142D01        BSET  SCCR2 $01     Set send break bit
164 BF62 120801FC      BRSET PORTD $01 *   Wait for RxD pin to go low
165 BF66 152D01        BCLR  SCCR2 $01     Clear send break bit
166
167 BF69 132E20FC      BRCLR SCSR $20 *    Wait for RDRF
168 BF6D 962F          LDAA  SCDAT          Read data
169          * Data will be $00 if BREAK or $00 received
170 BF6F 2603          BNE   NOTZERO       Bypass jump if not $00
171 BF71 7EF000        JMP   EPRMSTR       Jump to EEPROM if $00
172 BF74          NOTZERO EQU   *
173 BF74 81FF          CMPA  #$FF          $FF will be seen as $FF...
174 BF76 2708          BEQ   BAUDOK        if baud was correct
175          * Or else change to /104 (/13 & /8) 1200 @ 2MHz
176 BF78 142B33        BSET  BAUD $33      Works because $22 -> $33
177 BF7B CC0DB0        LDD   #DELAYS       And switch to slower...
178 BF7E DD16          STD   TOC1          delay constant
179 BF80          BAUDOK EQU   *
180 BF80 18CE0040      LDY   #RAMSTR       Point to start of RAM
181
182 BF84          WAIT   EQU   *
183 BF84 DE16          LDX   TOC1          Move delay constant to X
184 BF86          WTLOOP EQU   *
185 BF86 122E2009      BRSET SCSR $20 NEWONE Exit loop if RDRF set
186 BF8A 09           DEX          Decrement count
187 BF8B 01           NOP          Kill...
188 BF8C 01           NOP          ...seven cycles....
189 BF8D 2100         BRN   **2       ..to match original program
190 BF8F 26F5         BNE   WTLOOP    Loop if not timed out
191 BF91 200F         BRA   STAR      Quit download on timeout
192
193 BF93          NEWONE EQU   *
194 BF93 962F          LDAA  SCDAT        Get received data
195 BF95 18A700        STAA  $00,Y       Store to next RAM location
196 BF98 972F          STAA  SCDAT        Transmit it for handshake
197 BF9A 1808          INY          Point to next RAM location
198 BF9C 188C0100     CPY   #RAMEND+1   See if past end
199 BFA0 26E2         BNE   WAIT       If not, get another
200
201 BFA2          STAR   EQU   *
202 BFA2 CE1068        LDX   #PROGDEL     Init X with program delay
203 BFA5 18CEF000      LDY   #EPRMSTR     Init Y with EPROM start addr
204 BFA9 7E0040        JMP   RAMSTR       ** Exit to start of RAM **
205 *****
206          * Block fill unused bytes with zero
207
208 BFAC 000000000000   BSZ   $BFD1-*
209          000000000000
210          000000000000
211          000000000000
212          000000000000
213          000000000000
214          00
215
216 *****
217          * Boot ROM revision level in ASCII
218          * (ORG $BFD1)
219          * "B"
220          * Mask set I.D. ($0000 for EPROM parts)
221          * (ORG $BFD2)
222          * FDB $0000
223          * 711D3 I.D. - can be used to determine MCU type
224          * (ORG $BFD4)
225          * FDB $71D3
226          * VECTORS - point to RAM for pseudo-vector JUMPs
227
228 BFD1 42          FDB   $100-60      SCI
229 BFD2 0000       FDB   $100-57      SPI
230          FDB   $100-54      PULSE ACCUM INPUT EDGE
231          FDB   $100-51      PULSE ACCUM OVERFLOW
232          FDB   $100-48      TIMER OVERFLOW
233          FDB   $100-45      TIMER OUTPUT COMPARE 5
234          FDB   $100-42      TIMER OUTPUT COMPARE 4
235          FDB   $100-39      TIMER OUTPUT COMPARE 3
236          FDB   $100-36      TIMER OUTPUT COMPARE 2

```

Listing 5. MC68HC711D3 Bootloader ROM

Sheet 4 of 4

234 BFE8 00DF	FDB	\$100-33	TIMER OUTPUT COMPARE 1
235 BFEA 00E2	FDB	\$100-30	TIMER INPUT CAPTURE 3
236 BFEC 00E5	FDB	\$100-27	TIMER INPUT CAPTURE 2
237 BFEE 00E8	FDB	\$100-24	TIMER INPUT CAPTURE 1
238 BFF0 00EB	FDB	\$100-21	REAL TIME INT
239 BFF2 00EE	FDB	\$100-18	IRQ
240 BFF4 00F1	FDB	\$100-15	XIRQ
241 BFF6 00F4	FDB	\$100-12	SWI
242 BFF8 00F7	FDB	\$100-9	ILLEGAL OP-CODE
243 BFFA 00FA	FDB	\$100-6	COP FAIL
244 BFFC 00FD	FDB	\$100-3	CLOCK MONITOR
245 BFFE BF4D	FDB	BEGIN	RESET
246 C000	END		

Symbol Table:

Symbol Name	Value	Def. #	Line Number	Cross Reference
BAUD	002B	*00039	00156	00176
BAUDOK	BF80	*00179	00174	
BEGIN	BF4D	*00152	00245	
CONFIG	003F	*00050		
DDR	0009	*00031		
DELAY	021B	*00064	00159	
DELAYS	0DB0	*00063	00177	
DONEIT	BF40	*00139	00123	
EPGM	0001	*00047	00127	
EPRMEND	FFFF	*00056		
EPRMSTR	F000	*00055	00171	00203
LAT	0020	*00046	00124	00127
NEWONE	BF93	*00193	00185	
NOTZERO	BF74	*00172	00170	
OC1F	0080	*00036	00134	00137
PORTD	0008	*00030	00164	
PPROG	003B	*00044	00125	00128 00138
PRGROUT	BF10	*00113	00077	
PROGDEL	1068	*00066	00202	
PROGRAM	BF00	*00077		
RAMEND	00FF	*00059	00153	00198
RAMSTR	0040	*00058	00180	00204
SCCR1	002C	*00040		
SCCR2	002D	*00041	00158	00163 00165
SCDAT	002F	*00043	00094	00117 00121 00142 00168 00194 00196
SCSR	002E	*00042	00093	00115 00120 00140 00167 00185
SPCR	0028	*00038	00154	
STAR	BFA2	*00201	00191	
TCNT	000E	*00032	00132	
TEST1	003E	*00049		
TFLG1	0023	*00034	00135	00137
TOC1	0016	*00033	00133	00160 00178 00183
UPLOAD	BF03	*00078		
UPL0OP	BF03	*00091	00096	
WAIT	BF84	*00182	00199	
WAIT1	BF18	*00119	00144	
WTLOOP	BF86	*00184	00190	

Errors: None
 Labels: 37
 Last Program Address: \$BFFF
 Last Storage Address: \$0000
 Program Bytes: \$0100 256
 Storage Bytes: \$0000 0

Listing 6. MC68HC11F1 Bootloader ROM

```

1
2 * BOOTLOADER FIRMWARE FOR MC68HC11F1 - 04 May 90
3 *****
4 * Features of this bootloader are...
5 *
6 * Auto baud select between 7812, 1200, 9600, 5208
7 * and 3906 (E = 2 MHz).
8 * 0 - 1024 byte variable length download:
9 * reception of characters quits when an idle of at
10 * least four character times occurs. (Note: at 9600
11 * baud rate this is almost five bit times and at
12 * 5208 and 3906 rates the timeout is even longer).
13 * Jump to EEPROM at $FE00 if first download byte = $00.
14 * Part I.D. at $BFD4 is $F1F1.
15 *****
16 * Revision B -
17 *
18 * Added new baud rates: 5208, 3906.
19 *****
20 * Revision A -
21 *
22 * Added new baud rate: 9600.
23 *****
24
25 * Equates (use with index offset = $1000)
26 *
27 0008 PORTD EQU $08
28 0009 DDRD EQU $09
29 0016 TOC1 EQU $16
30 0028 SPCR EQU $28 (for DWOM bit)
31 002B BAUD EQU $2B
32 002C SCCR1 EQU $2C
33 002D SCCR2 EQU $2D
34 002E SCSR EQU $2E
35 002F SCDAT EQU $2F
36 003B PPROG EQU $3B
37 003E TEST1 EQU $3E
38 003F CONFIG EQU $3F
39
40 * Memory configuration equates
41 *
42 FE00 EEPSTR EQU $FE00 Start of EEPROM
43 FFFF EEPEND EQU $FFFF End of EEPROM
44 *
45 0000 RAMSTR EQU $0000 Start of RAM
46 03FF RAMEND EQU $03FF End of RAM
47
48 * Delay constants
49 *
50 0DB0 DELAYS EQU 3504 Delay at slow baud rate
51 021B DELAYF EQU 539 Delay at fast baud rates
52 *
53 *****
54 BF00 ORG $BF00
55 *****
56 * Main bootloader starts here
57 *****
58 * RESET vector points to here
59 BF00 BEGIN EQU *
60 BF00 8E03FF LDS #RAMEND Initialize stack pntr
61 BF03 CE1000 LDX #1000 X points to registers
62 BF06 1C2820 BSET SPCR,X $20 Select port D wire-OR mode
63 BF09 CCB00C LDD #B00C Baud in A, SCCRx in B
64 BF0C A72B STAA BAUD,X SCPx = /13, SCRx = /1
65
66 * Writing 1 to MSB of BAUD resets count chain
67 BF0E E72D STAB SCCR2,X Rx and Tx enabled
68 BF10 CC021B LDD #DELAYF Delay for fast baud rates
69 BF13 ED16 STD TOC1,X Set as default delay
70
71 * Send BREAK to signal start of download
72 BF15 1C2D01 BSET SCCR2,X $01 Set send break bit
73 BF18 1E0801FC BRSET PORTD,X $01 * Wait for RxD pin to go low
74 BF1C 1D2D01 BCLR SCCR2,X $01 Clear send break bit
75
76 BF1F 1F2E20FC BRCLR SCSR,X $20 * Wait for RDRF
77 BF23 A62F LDAA SCDAT,X Read data
78
79 * Data will be $00 if BREAK or $00 received

```

Listing 6. MC68HC11F1 Bootloader ROM

Sheet 2 of 3

```

78 BF25 2603          BNE  NOTZERO          Bypass jump if not $00
79 BF27 7EFE00       JMP  EEPSTR           Jump to EEPROM if it was $00
80 BF2A              EQU  *
81                  * Check div by 13 (9600 baud at 2 MHz)
82 BF2A 81F0         CMPA #SF0             SF0 will be seen as SF0...
83 BF2C 271D         BEQ  BAUDOK           if baud was correct
84                  * Check div by 104 (1200 baud at 2 MHz)
85 BF2E C633         LDAB #S33            Initialize B for this rate
86 BF30 8180         CMPA #S80            SFF will be seen as S80...
87 BF32 2710         BEQ  SLOBAUD         if baud was correct
88                  * Check div by 32 (3906 baud at 2 MHz)
89                  * (equals: 8192 baud at 4.2 MHz)
90 BF34 C605         LDAB #S05            Initialize B for this rate
91 BF36 8520         BITA #S20            SFD shows as bit 5 clear...
92 BF38 270A         BEQ  SLOBAUD         if baud was correct
93                  * Change to div by 16 (7812 baud at 2 MHz)
94                  * (equals: 8192 baud at 2.1 MHz)
95 BF3A C622         LDAB #S22            Initialize B for this rate
96 BF3C E72B         STAB BAUD,X          Store baud rate
97 BF3E 8508         BITA #S08            Switch to slower...
98 BF40 2609         BNE  BAUDOK           delay constant
99                  * Change to div by 24 (5208 baud at 2 MHz)
100                  * (equals: 8192 BAUD at 3.15 MHz)
101 BF42 C613         LDAB #S13            By default
102
103 BF44              SLOBAUD             EQU  *
104 BF44 E72B         STAB BAUD,X          Store baud rate
105 BF46 CCCC80       LDD #DELAYS          Switch to slower...
106 BF49 ED16         STD  TOC1,X          delay constant
107 BF4B              BAUDOK             EQU  *
108 BF4B 18CEC0C0     LDY  #RAMSTR         Point to start of RAM
109
110 BF4F              WAIT              EQU  *
111 BF4F EC16         LDD  TOC1,X          Move delay constant to D
112 BF51              WTLOOP             EQU  *
113 BF51 1E2E2007     BRSET SCSR,X $20 NEWONE Exit loop if RDRF set
114 BF53 8F          XGDX                Swap delay count to X
115 BF55 09          DEX                  Decrement count
116 BF57 8F          XGDX                Swap back to D
117 BF58 26F7         BNE  WTLOOP         Loop if not timed out
118 BF5A 20CF         BRA  WTLOOP         Quit download on timeout
119
120 BF5C              NEWONE             EQU  *
121 BF5C A62F         LDAA SCDAT,X        Get received data
122 BF5E 18A700     STAA $00,Y          Store to next RAM location
123 BF61 A72F         STAA SCDAT,X        Transmit it for handshake
124 BF63 1809         INY                  Point to next RAM location
125 BF65 18CC4000    CPY  #RAMEND+1     See if past end
126 BF69 26E4         BNE  WAIT           If not, get another
127
128 BF6B              STAR              EQU  *
129 BF6B 7E0000       JMP  RAMSTR         ** Exit to start of RAM **
130
131                  * Block fill unused bytes with zero
132
133 BF6E C00000000000    BSZ  $BFD1-*
134
135
136
137                  * Boot ROM revision level in ASCII
138 BFD1 42          (ORG  $BFD1)      FCC  "B"
139
140                  * Mask set I.D. - ($0000 for ROMless parts)
141                  (ORG  $BFD2)      FDB  $0000
142 BFD2 0000
143
144                  * 11F1 I.D. - can be used to determine MCU type
145                  (ORG  $BFD4)      FDB  $F1F1
146 BFD4 F1F1
147

```

Listing 6. MC68HC11F1 Bootloader ROM

Sheet 3 of 3

```

148          * VECTORS - point to RAM for pseudo-vector JUMPs
149
150 BFD6 00C4          FDB $100-60          SCI
151 BFD8 00C7          FDB $100-57          SPI
152 BFDA 00CA          FDB $100-54          PULSE ACCUM INPUT EDGE
153 BFDC 00CD          FDB $100-51          PULSE ACCUM OVERFLOW
154 BFDE 00D0          FDB $100-48          TIMER OVERFLOW
155 BFE0 00D3          FDB $100-45          TIMER OUTPUT COMPARE 5
156 BFE2 00D6          FDB $100-42          TIMER OUTPUT COMPARE 4
157 BFE4 00D9          FDB $100-39          TIMER OUTPUT COMPARE 3
158 BFE6 00DC          FDB $100-36          TIMER OUTPUT COMPARE 2
159 BFE8 00DF          FDB $100-33          TIMER OUTPUT COMPARE 1
160 BFEA 00E2          FDB $100-30          TIMER INPUT CAPTURE 3
161 BFEC 00E5          FDB $100-27          TIMER INPUT CAPTURE 2
162 BFEE 00E8          FDB $100-24          TIMER INPUT CAPTURE 1
163 BFF0 00EB          FDB $100-21          REAL TIME INT
164 BFF2 00EE          FDB $100-18          IRQ
165 BFF4 00F1          FDB $100-15          XIRQ
166 BFF6 00F4          FDB $100-12          SWI
167 BFF8 00F7          FDB $100-9           ILLEGAL OP-CODE
168 BFFA 00FA          FDB $100-6           COP FAIL
169 BFFC 00FD          FDB $100-3           CLOCK MONITOR
170 BFFE BF00          FDB BEGIN           RESET
171 C000          END

```

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUD	002B	*00031	00064	00096 00104
BAUDOK	BF4B	*00107	00083	00098
BEGIN	BF00	*00059	00170	
CONFIG	003F	*00038		
DDRD	0009	*00028		
DELAYF	021B	*00051	00067	
DELAYS	0DB0	*00050	00105	
EEPEND	FFFF	*00043		
EEPSTR	FE00	*00042	00079	
NEWONE	BF5C	*00120	00113	
NOTZERO	BF2A	*00080	00078	
PORTD	0008	*00027	00072	
PPROG	003B	*00036		
RAMEND	03FF	*00046	00060	00125
RAMSTR	0000	*00045	00108	00129
SCCR1	002C	*00032		
SCCR2	002D	*00033	00066	00071 00073
SCDAT	002F	*00035	00076	00121 00123
SCSR	002E	*00034	00075	00113
SLOBAUD	BF44	*00103	00087	00092
SPCR	0028	*00030	00062	
STAR	BF6B	*00128	00118	
TEST1	003E	*00037		
TOC1	0016	*00029	00068	00106 00111
WAIT	BF4F	*00110	00126	
WTLOOP	BF51	*00112	00117	

```

Errors: None
Labels: 26
Last Program Address: $BFFF
Last Storage Address: $0000
Program Bytes: $0100 256
Storage Bytes: $0000 0

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 0004
22 0005
23 0008
24 0009
25
26 000E
27 0016
28 0023
29
30 0080
31
32 002B
33
34 0020
35 0001
36
37 003B
38 003E
39 003F
40
41 0070
42 0072
43 0073
44 0074
45 0075
46 0076
47 0077
48
49
50
51 0D80
52 0FFF
53
54 2000
55 7FFF
56
57 0080
58 037F
59
60
61
62 15AB
63 0356
64
65 1068
66
67
68 BE40
69
70
71 BF00
72
73
74
75
76 BF00
77 BF00 8E037F
78
79
80
*****
* BOOTLOADER FIRMWARE FOR MC68HC11K4 - 18 Jul 90
*****
* Features of this bootloader are...
*
* Auto baud select between 7812, 1200, 9600, 5208
  and 3906 (E = 2 MHz).
* 0 - 768 byte variable length download:
* reception of characters quits when an idle of at
  least four character times occurs. (Note: at 9600
  baud rate this is almost five bit times and at
  5208 and 3906 rates the timeout is even longer).
* Jump to EEPROM at $0D80 if first download byte = $00.
* PROGRAM - Utility subroutine to program EPROM.
* UPLOAD - Utility subroutine to dump memory to host.
  Part I.D. at $BFD4 is $044B.
*****
* Equates (registers in direct space)
*
PORTB EQU $04
PORTF EQU $05
PORTD EQU $08
DDRD EQU $09
*
TCNT EQU $0E
TOC1 EQU $16
TFLG1 EQU $23
* Bit equates for TFLG1
OC1F EQU $80
*
EPROG EQU $2B
* Bit equates for EPROG
ELAT EQU $20
EPGM EQU $01
*
PPROG EQU $3B
TEST1 EQU $3E
CONFIG EQU $3F
*
SCBD EQU $70
SCCR1 EQU $72
SCCR2 EQU $73
SCSR1 EQU $74
SCSR2 EQU $75
SCDRH EQU $76
SCDRL EQU $77
*
* Memory configuration equates
*
EEPSTR EQU $0D80 Start of EEPROM
EPMEND EQU $0FFF End of EEPROM
*
ROMSTR EQU $2000 Start of ROM
ROMEND EQU $7FFF End of ROM
*
RAMSTR EQU $0080 Start of RAM
RAMEND EQU $037F End of RAM
*
* Delay constants
*
DELAYS EQU 5547 Delay at slow baud rate
DELAYF EQU 854 Delay at fast baud rates
*
PROGDEL EQU 4200 at 2.1MHz 2 mSec programming delay
*
CYCLCOD EQU $BE40 EPROM cycling code (TEST)
*
*****
71 BF00 ORG $BF00
*****
* Main bootloader starts here
*****
* RESET vector points to here
BEGIN EQU *
77 BF00 8E037F LDS #RAMEND Initialize stack ptr
*****
* Special jump for EEPROM Cycling routine
* (This is intended for factory test only)

```

Listing 7. MC68HC11K4 Bootloader ROM

Sheet 2 of 4

```

81                                     * If ports B and F both have %1001 0110 on them ...
82 BF03 CC9696                         LDD    #$9696
83 BF06 1A9304                         CPD    PORTB          Port F follows port B
84 BF09 2603                            BNE    CONTINU
85                                     * ... then execute the cycling code
86 BF0B 7EBE40                         JMP    CYCLCOD
87 BF0E                                CONTINU    EQU    *
88
89 BF0E CC001A                         LDD    #$001A          Initialize baud for...
90 BF11 DD70                           STD    SCBD            9600 baud at 2 MHz
91 BF13 CC400C                         LDD    #$400C          Put SCI in wire-OR mode...
92 BF16 DD72                           STD    SCCR1           Enable Xmtr and Rcvr
93 BF18 CC0356                         LDD    #DELAYF        Delay for fast baud rates
94 BF1B DD16                           STD    TOC1           Set as default delay
95
96 BF1D 147301                         * Send BREAK to signal ready for download
97 BF20 120801FC                       BSET   SCCR2 $01      Set send break bit
98 BF24 157301                         BRSET  PORTD $01 *    Wait for RxD pin to go low
99                                     BCLR  SCCR2 $01      Clear send break bit
100 BF27 137420FC                       BRCLR  SCSR1 $20 *    Wait for RDRF
101 BF2B 9677                            LDAA  SCDRL           Read data
102                                     * Data will be $00 if BREAK or $00 received
103 BF2D 2603                            BNE    NOTZERO        Bypass jump if not $00
104 BF2F 7E0D80                         JMP    EEPROMSTR      Jump to EEPROM if $00
105 BF32                                NOTZERO    EQU    *
106                                     * Check div by 26 (9600 baud at 2 MHz)
107 BF32 81F0                            CMPA  #$F0            $F0 will be seen as $F0...
108 BF34 271D                            BEQ   BAUDOK          if baud was correct
109                                     * Check div by 208 (1200 baud at 2 MHz)
110 BF36 C6D0                            LDAB  #$D0            Initialize B for this rate
111 BF38 8180                            CMPA  #$80            $FF will be seen as $80...
112 BF3A 2710                            BEQ   SLOBAUD        if baud was correct
113                                     * Check div by 64 (3906 baud at 2 MHz)
114                                     * (equals: 8192 baud at 4.2 MHz)
115 BF3C C640                            LDAB  #$40            Initialize B for this rate
116 BF3E 8520                            BITA  #$20            $FD shows as bit 5 clear...
117 BF40 270A                            BEQ   SLOBAUD        if baud was correct
118                                     * Change to div by 32 (7812 baud at 2 MHz)
119                                     * (equals: 8192 baud at 2.1 MHz)
120 BF42 C620                            LDAB  #$20            Initialize B for this rate
121 BF44 D771                            STAB  SCBD+1
122 BF46 8508                            BITA  #$08            $FF shows as bit 3 set...
123 BF48 2609                            BNE  BAUDOK          if baud was correct
124                                     * Change to div by 48 (5208 baud at 2 MHz)
125                                     * (equals: 8192 BAUD at 3.15 MHz)
126 BF4A C630                            LDAB  #$30            By default
127
128 BF4C                                SLOBAUD    EQU    *
129 BF4C D771                            STAB  SCBD+1          Store baudrate
130 BF4E CC15AB                         LDD  #DELAYS          Switch to slower...
131 BF51 DD16                            STD  TOC1             delay constant
132 BF53                                BAUDOK     EQU    *
133 BF53 18CE0080                       LDY  #RAMSTR          Point to start of RAM
134
135 BF57                                WAIT       EQU    *
136 BF57 DE16                           LDX  TOC1            Move delay constant to X
137 BF59                                WTLOOP    EQU    *
138 BF59 12742005                       BRSET  SCSR1 $20 NEWONE Exit loop if RDRF set
139 BF5D 09                             DEX                                     Decrement count
140 BF5E 26F9                           BNE  WTLOOP          Loop if not timed out
141 BF60 200F                           BRA  STAR            Quit download on timeout
142
143 BF62                                NEWONE    EQU    *
144 BF62 9677                            LDAA  SCDRL           Get received data
145 BF64 18A700                         STAA  $00,Y          Store to next RAM location
146 BF67 9777                            STAA  SCDRL          Transmit it for handshake
147 BF69 1808                            INY                                     Point to next RAM location
148 BF6B 188C0380                       CPY  #RAMEND+1       See if past end
149 BF6F 26E6                            BNE  WAIT            If not, get another
150
151 BF71                                STAR      EQU    *
152 BF71 7E0080                         JMP  RAMSTR          ** Exit to start of RAM **
153                                     *****
154                                     * Block fill unused bytes with zero
155

```


Listing 7. MC68HC11K4 Bootloader ROM

```

156 BF74 000000000000          BSZ  $BFD1-*
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000000000
      000000
157
158
159
160
161 BFD1 30
162
163
164
165 BFD2 0000
166
167
168
169
170 BFD4 044B
171
172
173
174 BFD6 00C4
175 BFD8 00C7
176 BFDA 00CA
177 BFDC 00CD
178 BFDE 00D0
179 BFEC 00D3
180 BFE2 00D6
181 BFE4 00D9
182 BFE6 00DC
183 BFE8 00DF
184 BFEA 00E2
185 BFEC 00E5
186 BFEE 00E8
187 BFF0 00EB
188 BFF2 00EE
189 BFF4 00F1
190 BFF6 00F4
191 BFF8 00F7
192 BFFA 00FA
193 BFFC 00FD
194 BFFE BF00
195 C000

```

```

*****
* Boot ROM revision level in ASCII
* (ORG $BFD1)
      FCC "0"
*****
* Mask set I.D. - set with user's ROM code mask layer
* (ORG $BFD2)
      FDB $0000 Reserve 2 bytes
*****
* 11K4 I.D. - can be used to determine MCU type
* (note: $4B = K in ASCII)
* (ORG $BFD4)
      FDB $044B
*****
* VECTORS - point to RAM for pseudo-vector JUMPs

```

```

      FDB $100-60 SCI
      FDB $100-57 SPI
      FDB $100-54 PULSE ACCUM INPUT EDGE
      FDB $100-51 PULSE ACCUM OVERFLOW
      FDB $100-48 TIMER OVERFLOW
      FDB $100-45 TIMER OUTPUT COMPARE 5
      FDB $100-42 TIMER OUTPUT COMPARE 4
      FDB $100-39 TIMER OUTPUT COMPARE 3
      FDB $100-36 TIMER OUTPUT COMPARE 2
      FDB $100-33 TIMER OUTPUT COMPARE 1
      FDB $100-30 TIMER INPUT CAPTURE 3
      FDB $100-27 TIMER INPUT CAPTURE 2
      FDB $100-24 TIMER INPUT CAPTURE 1
      FDB $100-21 REAL TIME INT
      FDB $100-18 IRQ
      FDB $100-15 XIRO
      FDB $100-12 SWI
      FDB $100-9 ILLEGAL OP-CODE
      FDB $100-6 COP FAIL
      FDB $100-3 CLOCK MONITOR
      FDB BEGIN RESET
      END

```

Listing 7. MC68HC11K4 Bootloader ROM

Sheet 4 of 4

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUDOK	BF53	*00132	00108	00123
BEGIN	BF00	*00076	00194	
CONFIG	003F	*00039		
CONTINU	BF0E	*00087	00084	
CYCLCOD	BE40	*00068	00086	
DDRD	0009	*00024		
DELAYF	0356	*00063	00093	
DELAYS	15AB	*00062	00130	
EPMEND	0FFF	*00052		
EPMSTR	0D80	*00051	00104	
ELAT	0020	*00034		
EPGM	0001	*00035		
EPROG	002B	*00032		
NEWONE	BF62	*00143	00138	
NOTZERO	BF32	*00105	00103	
OC1F	0080	*00030		
PORTB	0004	*00021	00083	
PORTD	0008	*00023	00097	
PORTF	0005	*00022		
PPROG	003B	*00037		
PROGDEL	1068	*00065		
RAMEND	037F	*00058	00077	00148
RAMSTR	0080	*00057	00133	00152
ROMEND	7FFF	*00055		
ROMSTR	2000	*00054		
SCBD	0070	*00041	00090	00121 00129
SCCR1	0072	*00042	00092	
SCCR2	0073	*00043	00096	00098
SCDRH	0076	*00046		
SCDRL	0077	*00047	00101	00144 00146
SCSR1	0074	*00044	00100	00138
SCSR2	0075	*00045		
SLOBAUD	BF4C	*00128	00112	00117
STAR	BF71	*00151	00141	
TCNT	000E	*00026		
TEST1	003E	*00038		
TFLG1	0023	*00028		
TOC1	0016	*00027	00094	00131 00136
WAIT	BF57	*00135	00149	
WTLOOP	BF59	*00137	00140	

Errors: None
 Labels: 40
 Last Program Address: \$BFFF
 Last Storage Address: \$0000
 Program Bytes: \$0100 256
 Storage Bytes: \$0000 0

```

1 *****
2 * BOOTLOADER FIRMWARE FOR MC68HC711K4 - 25 Apr 90
3 *****
4 * Features of this bootloader are...
5 *
6 * Auto baud select between 7812, 1200, 9600, 5208
7 * and 3906 (E = 2 MHz).
8 * 0 - 768 byte variable length download:
9 * reception of characters quits when an idle of at
10 * least four character times occurs. (Note: at 9600
11 * baud rate this is almost five bit times and at
12 * 5208 and 3906 rates the timeout is even longer).
13 * Jump to EEPROM at $0D80 if first download byte = $00.
14 * PROGRAM - Utility subroutine to program EPROM.
15 * UPLOAD - Utility subroutine to dump memory to host.
16 * Part I.D. at $BFD4 is $744B.
17 *****
18 * Revision B -
19 *
20 * Added new baud rates: 5208, 3906.
21 *****
22 * Revision A -
23 *
24 * Added new baud rate: 9600.
25 *****
26
27 * Equates (registers in direct space)
28 *
29 0004 PORTB EQU $04
30 0005 PORTF EQU $05
31 0008 PORTD EQU $08
32 0009 DDRD EQU $09
33 *
34 000E TCNT EQU $0E
35 0016 TOC1 EQU $16
36 0023 TFLG1 EQU $23
37 * Bit equates for TFLG1
38 0080 OC1F EQU $80
39 *
40 002B EPROG EQU $2B
41 * Bit equates for EPROG
42 0020 ELAT EQU $20
43 0001 EFGM EQU $01
44 *
45 003B PPROG EQU $3B
46 003E TEST1 EQU $3E
47 003F CONFIG EQU $3F
48 *
49 0070 SCBD EQU $70
50 0072 SCCR1 EQU $72
51 0073 SCCR2 EQU $73
52 0074 SCSR1 EQU $74
53 0075 SCSR2 EQU $75
54 0076 SCDRH EQU $76
55 0077 SCDRL EQU $77
56
57 * Memory configuration equates
58 *
59 0D80 EEPMSTR EQU $0D80 Start of EEPROM
60 0FFF EEPMEND EQU $0FFF End of EEPROM
61 *
62 2000 EPRMSTR EQU $2000 Start of EPROM
63 7FFF EPRMEND EQU $7FFF End of EPROM
64 *
65 0080 RAMSTR EQU $0080 Start of RAM
66 037F RAMEND EQU $037F End of RAM
67
68 * Delay constants
69 *
70 15AB DELAYS EQU 5547 Delay at slow baud rate
71 0356 DELAYF EQU 854 Delay at fast baud rates
72 *
73 1068 PROGDEL EQU 4200 2 mSec programming delay
74 * at 2.1MHz
75 *
76 BE40 CYCLCOD EQU $BE40 EPROM cycling code (TEST)
77 *
78 *****
79 BF00 ORG $BF00
80 *****
81

```

```

82          * Next two instructions provide a predictable place
83          * to call PROGRAM and UPLOAD even if the routines
84          * change size in future versions.
85          *
86 BF00 7EBF1D    PROGRAM      JMP      PRGROUT      EPROM programming utility
87 BF03          UPLOAD      EQU      *          Upload utility
88
89          *****
90          * UPLOAD - Utility subroutine to send data from
91          * inside the MCU to the host via the SCI interface.
92          * Prior to calling UPLOAD set baud rate, turn on SCI
93          * and set Y=first address to upload.
94          * Bootloader leaves baud set, SCI enabled.
95          * Consecutive locations are sent via SCI in an
96          * infinite loop. Reset stops the upload process.
97          *****
98 BF03 8D0D          BSR      INIT          Initialization subroutine
99
100 BF05          UPL00P     EQU      *
101 BF05 18A600      LDAA    0,Y            Read byte
102 BF08 137480FC   BRCLR  SCSR1 $80 *    Wait for TDRE
103 BF0C 9777      STAA   SCDRL          Send it
104 BF0E 1808      INY
105 BF10 20F3      BRA    UPL00P        Next....
106
107          *****
108          * Initialization subroutine - Forces EPROM to be
109          * enabled at $2000 so it is not overlapped by the
110          * BOOTLOADER firmware.
111          * User's address in index Y is adjusted to point to
112          * EPROM in this space instead of $A000.
113          *****
114 BF12          INIT      EQU      *
115 BF12 860F      LDAA  #$0F            EPROM is turned on
116 BF14 973F      STAA  CONFIG          at address $2000
117 BF16 188F      XGDY  Get user's address
118 BF18 847F      ANDA  #$7F            Clear bit 15 of address
119 BF1A 188F      XGDY  Return adjusted address
120 BF1C 39      RTS
121
122          *****
123          * PROGRAM - Utility subroutine to program EPROM.
124          * Prior to calling PROGRAM set baud rate, turn on SCI
125          * set X=2ms prog delay constant, and set Y=first
126          * address to program. SP must point to RAM.
127          * Bootloader leaves baud set, and SCI enabled so these
128          * default values do not have to be changed typically.
129          * Delay constant in X should be equivalent to 2 ms
130          * at 2.1 MHz X=4200; at 1 MHz X=2000, at 4MHz X=8000.
131          * An external voltage source is required for EPROM
132          * programming.
133          * This routine uses 2 bytes of stack space.
134          * Routine does not return. Reset to exit.
135          *****
136 BF1D          PRGROUT   EQU      *
137 BF1D 8DF3          BSR      INIT
138          * Send $FF to indicate ready for program data
139 BF1F 137480FC   BRCLR  SCSR1 $80 *    Wait for TDRE
140 BF23 86FF      LDAA  #$FF
141 BF25 9777      STAA  SCDRL
142          * WAIT FOR A BYTE
143 BF27          WAIT1     EQU      *
144 BF27 137420FC   BRCLR  SCSR1 $20 *    Wait for RDRF
145 BF2B D677      LDAB  SCDRL          Get received byte
146 BF2D 18E100    CMPB  $0,Y            See if already programmed
147 BF30 271D      BEQ   DONEIT         If so, skip prog cycle
148 BF32 8620      LDAA  #ELAT          Put EPROM in prog mode
149 BF34 972B      STAA  EPROG
150 BF36 18E700    STAB  0,Y            Write data
151 BF39 8621      LDAA  #ELAT+EPGM
152 BF3B 972B      STAA  EPROG          Turn on prog voltage
153 BF3D 3C      PSHX  Save delay on stack
154 BF3E 32      PULA  Put delay into D-reg
155 BF3F 33      PULB
156 BF40 D30E      ADDD  TCNT          Delay const + present TCNT
157 BF42 DD16      STD  TOC1           Schedule OC1 (prog delay)
158 BF44 8680      LDAA  #OC1F
159 BF46 9723      STAA  TFLG1        Clear any previous flag
160
161 BF48 132380FC   BRCLR  TFLG1 OC1F *    Wait for delay to expire

```

Listing 8. MC68HC711K4 Bootloader ROM

Sheet 3 of 5

```

162 BF4C 7F002B          CLR      EPROG          Turn off prog voltage
163 BF4F                DONEIT    EQU      *
164 BF4F 137480FC      BRCLR   SCSR1 $80 *    Wait for TDRE
165 BF53 18A600        LDAA    $0,Y           Read from EPROM and...
166 BF56 9777          STAA    SCDRL          Xmit for verify
167 BF58 1808          INY                    Point to next location
168 BF5A 20CB          BRA     WAIT1          Back to top for next
169                    * Loops indefinitely as long as more data sent.
170
171                    *****
172                    * Main bootloader starts here
173                    *****
174                    * RESET vector points to here
175 BF5C                BEGIN    EQU      *
176 BF5C 8E037F        LDS     #RAMEND        Initialize stack pntr
177                    *****
178                    * Special jump for EEPROM Cycling routine
179                    * (This is intended for factory test only)
180                    * If ports B and F both have %1001 0110 on them ...
181 BF5F CC9696        LDD     #$9696
182 BF62 1A9304        CPD     PORTB          Port F follows port B
183 BF65 2603          BNE     CONTINU
184                    * ... then execute the cycling code
185 BF67 7EBE40        JMP     CYCLCOD
186 BF6A                CONTINU  EQU      *
187
188 BF6A CC001A        LDD     #$001A        Initialize baud for...
189 BF6D DD70          STD     SCBD           9600 baud at 2 MHz
190 BF6F CC400C        LDD     #$400C        Put SCI in wire-OR mode...
191 BF72 DD72          STD     SCCR1          Enable Xmtr and Rcvr
192 BF74 CC0356        LDD     #DELAYF       Delay for fast baud rates
193 BF77 DD16          STD     TOC1          Set as default delay
194                    * Send BREAK to signal ready for download
195 BF79 147301        BSET   SCCR2 $01      Set send break bit
196 BF7C 120801FC      BRSET  PORTD $01 *    Wait for RxD pin to go low
197 BF80 157301        BCLR  SCCR2 $01      Clear send break bit
198
199 BF83 137420FC      BRCLR  SCSR1 $20 *    Wait for RDRF
200 BF87 9677          LDAA    SCDRL          Read data
201                    * Data will be $00 if BREAK or $00 received
202 BF89 2603          BNE     NOTZERO       Bypass jump if not $00
203 BF8B 7E0D80        JMP     EEPMSTR        Jump to EEPROM if $00
204 BF8E                NOTZERO  EQU      *
205                    * Check div by 26 (9600 baud at 2 MHz)
206 BF8E 81F0          CMPA   #$F0           $F0 will be seen as $F0...
207 BF90 271D          BEQ    BAUDOK         if baud was correct
208                    * Check div by 208 (1200 baud at 2 MHz)
209 BF92 C6D0          LDAB   #$D0           Initialize B for this rate
210 BF94 8180          CMPA   #$80           $FF will be seen as $80...
211 BF96 2710          BEQ    SLOBAUD        if baud was correct
212                    * Check div by 64 (3906 baud at 2 MHz)
213                    * (equals: 8192 baud at 4.2 MHz)
214 BF98 C640          LDAB   #$40           Initialize B for this rate
215 BF9A 8520          BITA   #$20           $FD shows as bit 5 clear...
216 BF9C 270A          BEQ    SLOBAUD        if baud was correct
217                    * Change to div by 32 (7812 baud at 2 MHz)
218                    * (equals: 8192 baud at 2.1 MHz)
219 BF9E C620          LDAB   #$20           Initialize B for this rate
220 BFA0 D771          STAB  SCBD+1
221 BFA2 8508          BITA   #$08           $FF shows as bit 3 set...
222 BFA4 2609          BNE   BAUDOK         if baud was correct
223                    * Change to div by 48 (5208 baud at 2 MHz)
224                    * (equals: 8192 BAUD at 3.15 Mhz)
225 BFA6 C630          LDAB   #$30           By default
226
227 BFA8                SLOBAUD  EQU      *
228 BFA8 D771          STAB  SCBD+1          Store baudrate
229 BFAA CC15AB        LDD   #DELAYS         Switch to slower...
230 BFAD DD16          STD   TOC1            delay constant
231 BFAF                BAUDOK   EQU      *
232 BFAF 18CE0080      LDY   #RAMSTR         Point to start of RAM
233
234 BFB3                WAIT     EQU      *
235 BFB3 DE16          LDX   TOC1            Move delay constant to X
236 BFB5                WTLOOP   EQU      *
237 BFB5 12742005      BRSET  SCSR1 $20 NEWONE Exit loop if RDRF set
238 BFB9 09            DEX                    Decrement count
239 BFBA 26F9          BNE   WTLOOP          Loop if not timed out
240 BFBC 200F          BRA   STAR            Quit download on timeout
241

```

```

242 BFBE          NEWONE          EQU      *
243 BFBE 9677          LDAA    SCDRL          Get received data
244 BFC0 18A700        STAA    $00,Y          Store to next RAM location
245 BFC3 9777          STAA    SCDRL          Transmit it for handshake
246 BFC5 1808          INY     INY            Point to next RAM location
247 BFC7 188C0380     CPY     #RAMEND+1      See if past end
248 BFCB 26E6          BNE     WAIT           If not, get another
249
250 BFCD          STAR            EQU      *
251 BFCD 7E0080        JMP     RAMSTR          ** Exit to start of RAM **
252          *****
253          * Block fill unused bytes with zero
254
255 BFD0 00          BSZ     $BFD1-*
256
257          *****
258          * Boot ROM revision level in ASCII
259          * (ORG $BFD1)
260 BFD1 42          FCC     "B"
261          *****
262          * Mask set I.D. ($0000 for EPROM parts)
263          * (ORG $BFD2)
264 BFD2 0000        FDB     $0000
265          *****
266          * 711K4 I.D. - can be used to determine MCU type
267          * (note: $4B = K in ASCII)
268          * (ORG $BFD4)
269 BFD4 744B        FDB     $744B
270          *****
271          * VECTORS - point to RAM for pseudo-vector JUMPS
272
273 BFD6 00C4          FDB     $100-60        SCI
274 BFD8 00C7          FDB     $100-57        SPI
275 BFDA 00CA          FDB     $100-54        PULSE ACCUM INPUT EDGE
276 BFDC 00CD          FDB     $100-51        PULSE ACCUM OVERFLOW
277 BFDE 00D0          FDB     $100-48        TIMER OVERFLOW
278 BFE0 00D3          FDB     $100-45        TIMER OUTPUT COMPARE 5
279 BFE2 00D6          FDB     $100-42        TIMER OUTPUT COMPARE 4
280 BFE4 00D9          FDB     $100-39        TIMER OUTPUT COMPARE 3
281 BFE6 00DC          FDB     $100-36        TIMER OUTPUT COMPARE 2
282 BFE8 00DF          FDB     $100-33        TIMER OUTPUT COMPARE 1
283 BFEA 00E2          FDB     $100-30        TIMER INPUT CAPTURE 3
284 BFEC 00E5          FDB     $100-27        TIMER INPUT CAPTURE 2
285 BFEE 00E8          FDB     $100-24        TIMER INPUT CAPTURE 1
286 BFF0 00EB          FDB     $100-21        REAL TIME INT
287 BFF2 00EE          FDB     $100-18        IRQ
288 BFF4 00F1          FDB     $100-15        XIRQ
289 BFF6 00F4          FDB     $100-12        SWI
290 BFF8 00F7          FDB     $100-9         ILLEGAL OP-CODE
291 BFFA 00FA          FDB     $100-6        COP FAIL
292 BFFC 00FD          FDB     $100-3        CLOCK MONITOR
293 BFFE BF5C          FDB     BEGIN          RESET
294 C000          END

```

Listing 8. MC68HC711K4 Bootloader ROM

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
BAUDOK	BFAF	*00231	00207	00222
BEGIN	BF5C	*00175	00293	
CONFIG	003F	*00047	00116	
CONTINU	BF6A	*00186	00183	
CYCLCOD	BE40	*00076	00185	
DDRD	0009	*00032		
DELAYF	0356	*00071	00192	
DELAYS	15AB	*00070	00229	
DONEIT	BF4F	*00163	00147	
EPMEND	0FFF	*00060		
EPMSTR	0D80	*00059	00203	
ELAT	0020	*00042	00148	00151
EPGM	0001	*00043	00151	
EPRMEND	7FFF	*00063		
EPRMSTR	2000	*00062		
EPROG	002B	*00040	00149	00152 00162
INIT	BF12	*00114	00098	00137
NEWONE	BFBE	*00242	00237	
NOTZERO	BF8E	*00204	00202	
OC1F	0080	*00038	00158	00161
PORTB	0004	*00029	00182	
PORTD	0008	*00031	00196	
PORTF	0005	*00030		
PProg	003B	*00045		
PRGROUT	BF1D	*00136	00086	
PROGDEL	1068	*00073		
PROGRAM	BF00	*00086		
RAMEND	037F	*00066	00176	00247
RAMSTR	0080	*00065	00232	00251
SCBD	0070	*00049	00189	00220 00228
SCCR1	0072	*00050	00191	
SCCR2	0073	*00051	00195	00197
SCDRH	0076	*00054		
SCDRL	0077	*00055	00103	00141 00145 00166 00200 00243 00245
SCSR1	0074	*00052	00102	00139 00144 00164 00199 00237
SCSR2	0075	*00053		
SLOBAUD	BFA8	*00227	00211	00216
STAR	BFCB	*00250	00240	
TCNT	000E	*00034	00156	
TEST1	003E	*00046		
TFLG1	0023	*00036	00159	00161
TOC1	0016	*00035	00157	00193 00230 00235
UPLOAD	BF03	*00087		
UPL00P	BF05	*00100	00105	
WAIT	BFB3	*00234	00248	
WAIT1	BF27	*00143	00168	
WTLOOP	BFB5	*00236	00239	

Errors: None
 Labels: 47
 Last Program Address: \$BFFF
 Last Storage Address: \$0000
 Program Bytes: \$0100 256
 Storage Bytes: \$0000 0

Use of Stack Simplifies M68HC11 Programming

By Gordon Doughman

INTRODUCTION

The architectural extensions of the M6800 incorporated into the M68HC11 allow easy manipulation of data residing on the stack of the microcontroller unit (MCU). The M68HC11 central processing unit (CPU) automatically uses the stack for two purposes. Each time the CPU executes a branch to subroutine (BSR) or jump to subroutine (JSR) instruction, it pushes a return address onto the stack. This procedure allows the CPU to resume execution with the instruction following the BSR or JSR when the program returns from the subroutine. Second, just before the MCU executes an interrupt service routine, the CPU saves its register contents on the stack, allowing the registers to be restored when the CPU executes a return from interrupt (RTI) instruction at the end of the interrupt service routine. Two additional uses of the M68HC11 stack discussed in this application note are the storage of local or temporary variable values and subroutine parameter passing.

Using the stack for local variables and parameter passing provides the assembly language programmer with the following benefits. First, since a routine allocates storage space for local variables and parameters upon entry and releases the storage upon exit, the same temporary memory space can be reused by program routines that run in succession. This reuse can result in a substantial savings in the total amount of RAM required by a program.

Second, allocating a new set of local variables and parameters when entering a routine makes it both reentrant and recursive. Routines that possess these two properties can make a programmer's job much easier when debugging a program in a real-time, interrupt-driven environment.

Third, placing local variables and parameters on the stack helps to promote modular programming. Because all temporary storage required by a routine is allocated and deallocated by the program module itself, it can be easily detached from the main program for reuse or replacement.

The final major benefit of using the stack for local variables and parameters becomes apparent during the debugging process. Because a routine's local variables and parameters exist only while it is executing, it is very unlikely that one routine will accidentally modify the local variables and parameters of another routine. Once the programmer has written and debugged a routine, time can be spent finding logical errors and/or problems associated with the interaction of the different routines in a program.

The goal of this application note is to help the assembly language programmer understand the following topics: 1) the basic operation of the M68HC11 stack, 2) the concept of local and global variables, 3) subroutine parameter passing, and 4) use of the M68HC11 instruction set to support local variables and parameter passing.

The source code for the examples and the macros described in this application note can be obtained by calling the Motorola FREEWARE Bulletin Board Service (BBS) at (512) 891-3733. The FREEWARE BBS operates 24 hours a day, 7 days a week, except for maintenance. The BBS's format is 300-2400 BAUD, 8 data bits, 1 stop bit, and no parity.

M68HC11 STACK OPERATION

The M68HC11 supports a stack through the use of the CPU stack pointer (SP) register. The SP is a 16-bit register that points to an area of RAM used for stack storage. Because the SP is 16 bits wide, the stack can be located anywhere in the M68HC11 64 K byte address space. The SP contents are undefined at power-up and are normally initialized in the first few instructions of a program. Each time a byte is pushed onto the stack, the SP is automatically decremented. Therefore, the initial value loaded into the SP is usually the address of the last RAM location in a system. Thus, as more information is pushed onto the stack, the stack area grows downward (the SP points to lower addresses) in the memory map. The SP always contains the address of the next available location on the stack.

As previously mentioned, the stack on the M68HC11 is used automatically by the CPU hardware during subroutine calls/returns and during the servicing of interrupts. When a subroutine is called by a JSR or BSR instruction, the address of the instruction following the JSR or BSR is automatically pushed onto the stack. Since the M68HC11 only has an 8-bit data bus, two separate push operations are performed by the CPU hardware. During the first push operation, the low-order eight bits (b7 – b0) of the return address are placed on the stack. The second push operation places the high-order eight bits (b15 – b8) of the return address on the stack at the next lower address in memory. Performing the operation in this order leaves the 16-bit return address on the stack in the order that all 16-bit numbers are stored in memory, with the high-order eight bits at the lower address. After a JSR or BSR instruction, the stack appears as shown in Figure 1.

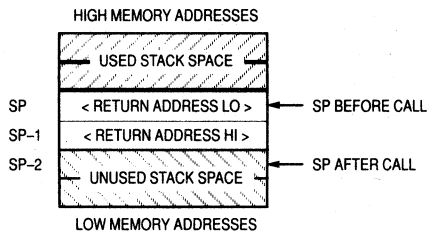


Figure 1. Stack Contents after Executing a JSR or BSR Instruction

Whenever an unmasked interrupt occurs, the contents of all CPU registers (with the exception of the SP itself) are pushed onto the stack as shown in Figure 2. After the registers are stacked, CPU execution continues at an address specified by the vector for the pending interrupt source. Upon completion of the interrupt service routine, the execution of an RTI instruction restores the previously saved CPU registers by pulling them off the stack in the reverse order in which they were pushed onto the stack. Since the entire state of the CPU is restored, execution resumes as if the interrupt had not occurred.

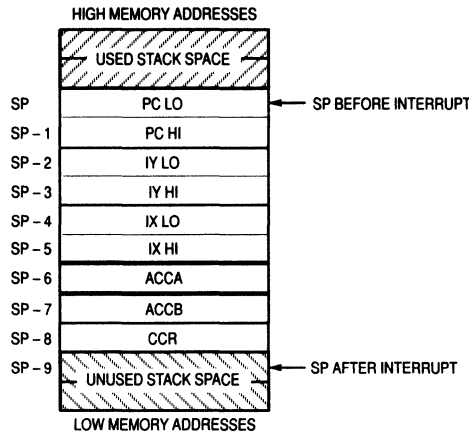


Figure 2. Stack Contents after an Interrupt

The M68HC11 instruction set contains instructions that allow the individual CPU registers to be pushed onto and pulled off the stack. For example, if the value contained in one of the CPU registers needs to be saved before a particular subroutine call, a push instruction places the register value on the stack. When the subroutine returns, a pull instruction restores the contents of the CPU register. These instructions not only allow the stack to be used as temporary data storage but also allow the construction of recursive and reentrant subroutines. M68HC11 instructions that involve the direct manipulation of the SP are listed in Table 1.

Table 1. Instructions Involving Direct Manipulation of the SP

Instruction Mnemonic	Description
PSHA	Push Accumulator A onto the Stack.
PSHB	Push Accumulator B onto the Stack.
PULA	Pull Accumulator A off the Stack.
PULB	Pull Accumulator B off the Stack.
PSHX	Push Index Register X onto the Stack.
PSHY	Push Index Register Y onto the Stack.
PULX	Pull Index Register X off the Stack.
PULY	Pull Index Register Y off the Stack.
INS	Increment the Stack Pointer by 1.
DES	Decrement the Stack Pointer by 1.
TXS	Place the Contents of Index Register X - 1 in the Stack Pointer.
TYS	Place the Contents of Index Register Y - 1 in the Stack Pointer.
TSX	Place the Contents of the Stack Pointer + 1 in Index Register X.
TSY	Place the Contents of the Stack Pointer + 1 in Index Register Y.

STACK USAGE

Although most assembly language programmers use the M68HC11 stack for subroutine return addresses, register contents during interrupt processing, and temporary CPU register storage, more powerful programming techniques can make additional use of the stack.

Most high-level language compilers for modern, block-structured, high-level languages make use of the stack for two additional functions: passing parameters and local or temporary variable storage. By borrowing some of these techniques, programmers can write assembly language programs that are much more reliable, easier to maintain, and easier to debug.

VARIABLES IN ASSEMBLY LANGUAGE

Computer programs rarely operate on data directly; instead, the program refers to variables. A variable is a physical location in computer memory that can be used to hold different values while the program runs. Variables usually have an identifier or name associated with them. Using names to refer to data contained in memory is much easier than trying to remember a long string of binary or hexadecimal numbers.

Besides a name and an address, variables may have several other attributes. Depending on the programming language, variable declarations may assign attributes to the variables restricting both the scope and extent of the variable. The scope of a variable is the range of program text in which a particular variable is known and can be used. The extent of a variable is the time during which a computer associates physical storage with a variable name.

In assembly language, the scope of variables is usually global — i.e., variables may be referenced throughout the text of a program. Though some assemblers may provide mechanisms to restrict the scope of declared variables, many assembly language programmers do not use these features. A programmer using assembly language usually declares variables by employing an assembler directive as shown in Listing 1. This method assigns fixed storage locations to the variables. The extent of variables declared this way is for the entire program execution — i.e., the storage locations assigned to the variables at assembly time remain allocated during the entire time the program is executing.

```

*
*          RAM LOCATIONS
*
*
*
*
*          ORG      $10
*
STANUM   RMB      1          STATION NUMBER REGISTER.
DATBLP   RMB      1          DATA TABLE POINTER REGISTER.
STAMSK   RMB      1          STATION BIT MASK REGISTER.
FCNUM    RMB      1          FUNCTION NUMBER REGISTER FOR MODE SET.
XTEMP    RMB      2          X-REG. TEMPORARY STORAGE.
XTEMP1   RMB      2          X-REGISTER TEMPORARY STORAGE.
ATEMP1   RMB      1          A-REGISTER TEMPORARY STORAGE.
COUNT1  RMB      1          COUNT USED DURING STATION POLLING LOOP.
KPCNT    RMB      1          'NUMBER OF KEYS PRESSED' COUNT.
LSTFCN   RMB      1          LAST T/L FUNCTION THAT WAS PROCESSED.
CALLST   RMB      1          REMOTE CALL STATUS BYTE.
ATEMP2   RMB      1          A-REG. TEMPORARY STORAGE FOR THE DELAY SUBROUTINE.
XTEMP3   RMB      2          X-REG. STORAGE BEFORE CALL TO DELAY SUBROUTINE.
COUNT2  RMB      1          COUNT USED IN DELAY SUBROUTINE.
NONESEL  RMB      1          'NONE SELECTED' REGISTER USED BY SSCHK.
*

```

Listing 1. Declaring Global Variables in Assembly Language

Further examination of the variable declarations in Listing 1 shows that several variables are used for intermediate calculation results or for temporary CPU register storage. This example is typical of the way many assembly language programmers allocate temporary storage. Each time they write a routine requiring temporary variable storage, they allocate an additional set of global variables. The

use of this technique can lead to the inefficient use of RAM if there are many routines within a program requiring temporary storage.

In an effort to make more efficient use of the limited amount of RAM on single-chip MCUs, some programmers use a technique known as “variable sharing.” Listing 2 shows a portion of a listing using this technique. In this program, more than one routine shares the use of a single temporary variable. To keep track of which routines use which variables, each line, in addition to the variable declaration, contains a list of the routines using that particular variable. In small programs, it may not be too difficult to manage temporary variables this way; however, in large programs having hundreds or thousands of routines using temporary variables, it becomes impossible to keep track of which routines use which temporary variables at any given time.

```
*
*   RAM LOCATIONS
*
*
*   ORG     $0
*
*** variables - used by: ***
PTR0   RMB 2   main,readbuff,incbuff,AS
PTR1   RMB 2   main,BR,DU,MO,AS,EX
PTR2   RMB 2   EX,DU,MO,AS
PTR3   RMB 2   EX,HO,MO,AS
PTR4   RMB 2   EX,AS
PTR5   RMB 2   EX,AS,BOOT
PTR6   RMB 2   EX,AS,BOOT
PTR7   RMB 2   EX,AS
PTR8   RMB 2   AS
TMP1   RMB 1   main,hexbin,buffarg,termarg
TMP2   RMB 1   GO,HO,AS,LOAD
TMP3   RMB 1   AS,LOAD
TMP4   RMB 1   TR,HO,ME,AS,LOAD
```

Listing 2. Declaring Global Variables in Assembly Language

The sharing of temporary variable storage shown in Listing 2 can produce debugging problems that are extremely hard to find. The chances of having one routine unintentionally modify the temporary storage of another can become quite high in large programs. In interrupt-driven, real-time systems, the sharing of temporary variables by various routines can become disastrous. Consider the situation illustrated in Figure 3. Subroutine A and subroutine B both share the temporary variable Temp1. Initially, there seems to be no problem since subroutine A and subroutine B do not call one another. Yet, consider what happens if an interrupt occurs during the execution of subroutine A. Because of the interrupt, subroutine B is called indirectly through subroutine C. The execution of subroutine B causes any value placed in Temp1 by subroutine A before the interrupt to be overwritten! Because interrupts usually occur asynchronously to main program execution, the program may appear to operate properly most of the time and crash randomly, depending on when an interrupt occurs. This type of apparently random program failure can be almost impossible to find.

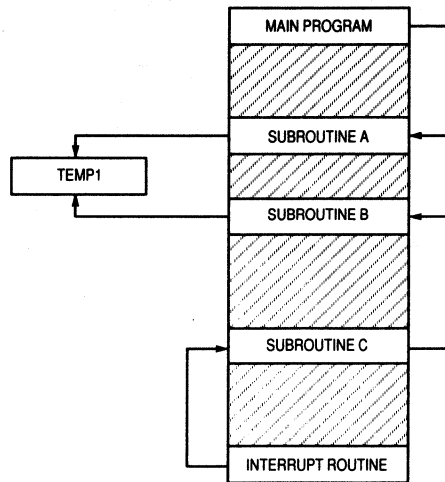


Figure 3. Two Subroutines Sharing a Single Temporary Variable

Though this example may seem overly simplistic, a program that contains hundreds or thousands of routines makes it nearly impossible to keep track of which subroutines are using what variables at any specific time, particularly if the main program and interrupt service routines share subroutines. The solution to this type of problem may seem simple — do not allow any subroutines to share globally declared temporary variables. This solution is acceptable provided enough RAM is available for all required temporary variables. A better solution to this problem can be found by examining the way modern, block-structured, high-level languages use temporary variables.

VARIABLES IN BLOCK-STRUCTURED HIGH-LEVEL LANGUAGES

Most block-structured, high-level languages, notably C and Pascal, provide the ability to limit both the scope and the extent of variables as part of the language definition. In both C and Pascal, the scope of a variable is local to the block in which it is declared. The scope of variables declared outside of a block (function or procedure) is usually global. These global variables are similar to the ones declared in the assembly language shown in Listing 1. They can be accessed by all routines within a program, and they remain in existence throughout the entire time the program executes. Listing 3 shows an example of how global variables are declared in C and Pascal.

Pascal	C
<pre> var x,y:integer; j:char; z:boolean; num:array[1..10] of integer; Date:record Month:integer; Day:integer; Year:integer; end; }; program(input,output); . . . end.</pre>	<pre> int x,y; char j; int z; int num[9]; struct Date { int x,y; int Day; int Year; }; main() { . . . }</pre>

Listing 3. Declaring Global Variables in High-Level Languages

Variables declared within a function or procedure have their scope limited to that function or procedure. The extent of these variables is also limited. These variables, known as local or automatic variables, come into existence when the functions or procedures that contain them are called. When a function or procedure finishes execution, the local variables disappear, and the memory locations occupied by them can be used again. Listing 4 shows an example of how local variables are declared in C and Pascal. In both examples, the variables *i* and *j* are local to procedure/function A and do not exist outside them.

Pascal	C
<pre> var x,y:integer; z:boolean; procedure A; var i,j:integer; begin . . . end;</pre>	<pre> int x,y; int z; A() { int i,j; . . . }</pre>

Listing 4. Declaring Local Variables in High-Level Languages

There are several benefits of using local variables. First, the restricted life of local variables can result in memory savings. Since storage for local variables is allocated upon entry to a routine and released upon exit from a routine, the same temporary memory space can be used by many different program routines. If two routines are run in succession, each can use the same storage locations.

Second, since a new set of local variables is allocated each time the procedure or function is entered, it makes the routine both reentrant and recursive. A reentrant routine is one that allocates a new set of local variables upon entry. When complex programs are run in a real-time, interrupt-driven environment, the interrupt handlers may call the routine that was interrupted. Making routines reentrant can greatly simplify a programmer's job during the debugging process in a real-time environment. The same properties that make a routine reentrant also makes a routine recursive. A recursive routine is one that can call itself.

Third, the use of local variables helps to promote modular programming. A program module is a self-contained program element that can be easily detached from the main program either for reuse in another program or for replacement. Since any storage space for local variables is allocated and deallocated by the program module itself, the module code can easily be copied from a single place within one program and reused in another program.

A fourth benefit of using local variables is evidenced during the debugging process. In complex programs, there may be hundreds or thousands of routines that have to interact with each other. Since local variables help isolate any changes made within a routine, debugging becomes a much simpler process. Once routines are written and debugged, the programmer does not have to worry about one routine accidentally modifying the local variables of another. Instead, time can be spent finding any logical errors and/or problems associated with the interaction of routines in the program.

Even with all the benefits provided by the use of local variables, there are some costs associated with their use. On the M68HC11, programs using local variables tend to be slightly larger and slower than programs using only global variables because the addressing modes required to access the local variables can make the instruction somewhat longer and may cause longer execution time. Given the benefits of using local variables, a slightly larger and slower program is usually well worth the cost.

The reusable memory storage for local variables is usually taken from the same memory space used for the MCU's hardware stack. Placing local variables on the hardware stack leaves them intact even if the routine using them is interrupted. The specifics of allocating, deallocating, and accessing local variables residing on the M68HC11 stack is discussed in **USING THE M68HC11 STACK**.

PASSING PARAMETERS

To make routines more flexible and to vary their actions each time they are called, different information must be passed to the routines. Generally, most assembly language programmers use the CPU registers to pass information to a subroutine. Using this technique is acceptable as long as the amount of information to be passed to the subroutine fits within the available CPU registers.

When the amount of information to be passed to a routine exceeds the space available in the CPU registers, the information can be passed in a set of global variables. This technique may be acceptable for some situations, but it can also cause problems that make debugging difficult. One problem with passing parameters in this manner is that it makes a routine non-reentrant. Referring to Figure 4, assume that subroutine A's parameters are passed in a set of global variables. If subroutine A is called either by the main program or by subroutine C as a result of an interrupt, the program will work correctly. If an interrupt occurs during the execution of subroutine A, the original parameters passed by the main program will be overwritten when subroutine C calls subroutine A. When the processor returns from the interrupt and resumes execution of subroutine A, it will be using incorrect parameter data, and the results passed back to the main program will most likely be incorrect.

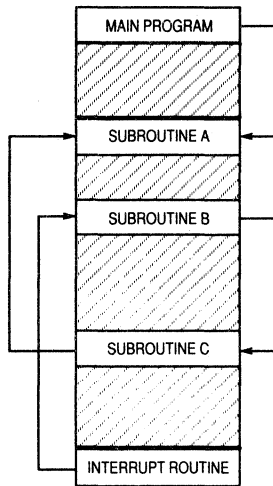


Figure 4. Subroutine Calling Chain

Because interrupts usually occur asynchronously to main program execution, the program may appear to operate properly most of the time and crash randomly. This type of problem can be extremely difficult to locate and can make debugging of real-time, interrupt-driven systems very difficult. Passing the parameters on the stack completely solves this problem. When subroutine C calls subroutine A as a result of the interrupt, a new set of parameters is placed on the stack while the original parameters remain undisturbed. Figure 5 shows the state of the stack after an interrupt.

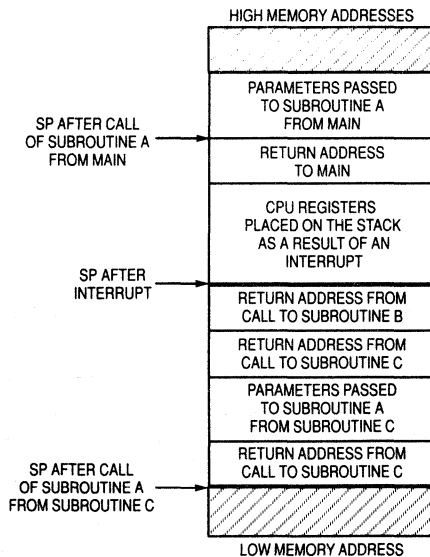


Figure 5. Stack State as a Result of an Interrupt

In addition to where parameters are passed, there is also an issue of how parameters are passed. Subroutine parameters can be passed either by value or by reference. When a parameter is passed by value, the parameter acts as a local variable whose initial value is provided by the calling routine. Any modification of the supplied value has no effect on the original data that was passed to the subroutine. Thus, a subroutine can import values but not export values by means of value parameters.

Passing a parameter by reference is one method used to pass results back to a calling subroutine. These types of parameters are known as variable parameters. When using variable parameters, the address of the actual parameter is passed to the subroutine rather than a value. The passed address can be a local variable of the calling routine or even the address of a global variable. Whenever a subroutine has to effect a permanent change in the values passed to it, the parameters must be passed by reference rather than by value.

Consider the following example in both C and Pascal that exchanges the value of two integers:

<pre> Pascal Call By Value procedure SwapInt (x,y:integer); var Temp:integer; begin Temp:=x; x:=y y:=Temp end; Call By Reference procedure SwapInt (var x,y:integer); var Temp:integer; begin Temp:=x; x:=y y:=Temp end; </pre>	<pre> C Call By Value void SwapInt (int x,y) { int Temp; Temp=x; x=y y=Temp } Call By Reference void SwapInt (int *x,*y) { int Temp; Temp=*x; *x=*y *y=Temp } </pre>
<pre> Call Of "SwapInt" Using Either Method program(output); var z,w:integer; begin z:=2; w:=4; Swapint (z,w); end; </pre>	<pre> Call Of "SwapInt" Using Call by Reference main() { int w,z; z=2; w=4; SwapInt (&z,&w); } </pre>

Listing 5. Passing Parameters by Reference and by Value

If the call-by-value routine were to be used in this example, the routine would not work as the programmer might expect. It would exchange the local values of x and y within the SwapInt routine, but it would have no effect on the actual variables in the routine's call statement. For the SwapInt routine to work properly, the routine must be declared so that the parameters are passed by reference rather than by value. As mentioned previously, passing a parameter by reference passes the address of the actual parameter. In the example in Listing 5, using the call-by-reference routine, the addresses of the variables z and w are passed to the SwapInt routine when it is called from the main program. This procedure allows the SwapInt routine to exchange the actual values of the variables passed to the routine.

FUNCTION/SUBROUTINE RETURN VALUES

Most subroutines or functions, if they are to perform a useful action in a program, will return one or more values to the calling routine. Any value or status can be returned using one of the three methods previously described. When a subroutine only needs to return a single value, one of the CPU registers is commonly used to pass the value back to the calling routine. This simple, safe technique allows the routine to remain reentrant. This method is used most often by C compilers to return a value from a function.

Similar to the situation that exists when passing parameters in the CPU registers, there may be times when a routine must return more information than will fit in the CPU registers. The information can be returned in a set of global variables; however, as previously described, this method poses the same problems as passing parameters in this manner. Returning results in global variables makes the routine non-reentrant and can cause the same debugging problems previously described.

A better way to return large amounts of data from a subroutine is to allocate the required amount of space on the stack either just before or just after pushing a routine's parameters onto the stack. This method possesses the same benefits of passing parameters on the stack — it makes the routine completely reentrant and self-contained. Most Pascal compilers return function values in this manner.

USING THE M68HC11 STACK

This section specifically discusses how to allocate, deallocate, and access both local variables and parameters residing on the M68HC11 stack. The programmer's model of the M68HC11 is shown in Figure 6. The following paragraphs briefly describe the CPU registers and their usage.

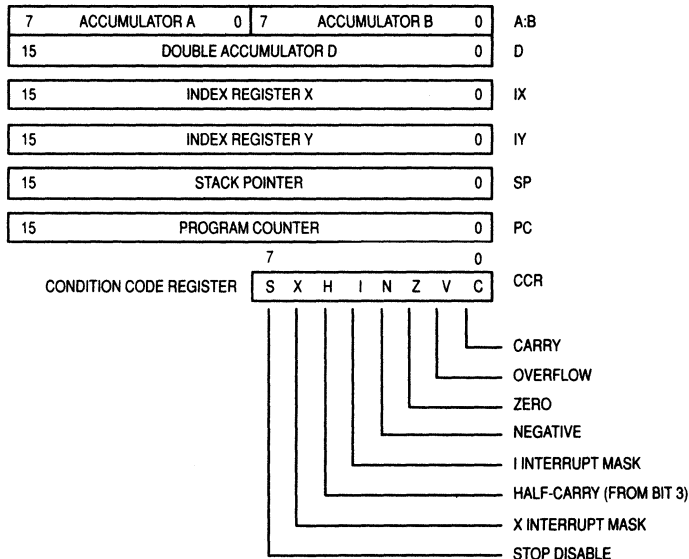


Figure 6. M68HC11 Programmer's Model

The A and B accumulators are used to hold operands and the results of arithmetic and logic operations. These two 8-bit registers can be concatenated to form a single 16-bit D accumulator to support the M68HC11 16-bit arithmetic instructions. The A and B accumulators can easily be used to push data onto or pull data off the stack.

The X and Y index registers are used in conjunction with the CPU indexed addressing mode. The indexed addressing mode uses the contents of the 16-bit index register in addition to a fixed 8-bit unsigned offset that is part of the instruction to form the effective address of the operand to be used by the instruction. The index registers play a very important role in accessing data residing on the stack.

The CPU SP is a 16-bit register that points to an area of RAM used for stack storage. The stack is used automatically during subroutine calls to save the address of the instruction that follows the call. When an interrupt occurs, the stack is used automatically by the CPU to save the entire CPU register contents on the stack (except for the SP itself). The SP always contains the address of the next available location on the stack.

The program counter (PC) is a 16-bit register used to hold the address of the next instruction to be executed.

The condition code register (CCR) contains five status indicators and two interrupt mask bits. The status bits reflect the results of arithmetic and other operations of the CPU as it performs instructions.

Before considering the specifics of parameter passing and the utilization of local variables that reside on the M68HC11 stack, the method used to access the information placed on the stack will be discussed. One M68HC11 index register and the CPU indexed addressing mode are used to access parameters or local variables residing on the stack. With respect to the indexed addressing mode, the contents of one of the 16-bit index registers plus a fixed unsigned offset is used in calculating the effective address of an instruction's operand. The unsigned offset, contained in a single byte following the instruction opcode, can only accommodate positive offsets in the range 0 – 255. Thus, the indexed addressing mode can only access information at addresses that are between 0 and 255 bytes greater than the base address contained in one of the index registers. Figure 7 illustrates how to calculate the effective address of an instruction using the indexed addressing mode.

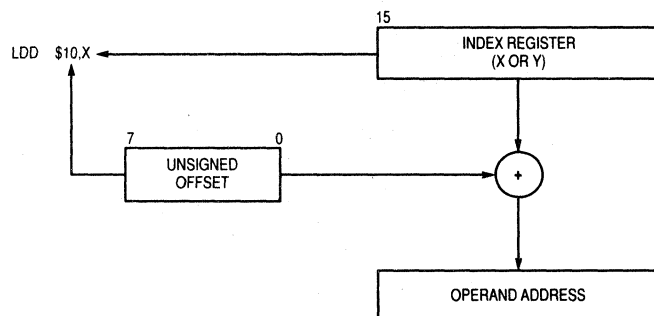


Figure 7. Effective Address Calculation for the Indexed Addressing Mode

As information is pushed onto the M68HC11 stack, the SP is decremented, signifying that the information placed on the stack resides at addresses greater than the address contained in the SP. The use of indexed addressing is ideal for accessing information residing on the M68HC11 stack. The example shown in Figure 8 illustrates how information on the stack is manipulated.

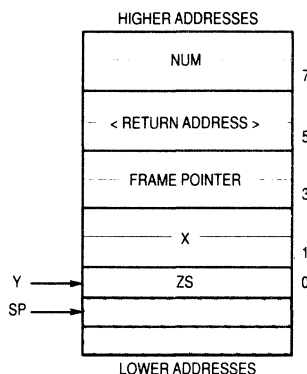


Figure 8. Stack Data Access Example

As Figure 8 shows, the SP is pointing to the next available address, and the Y index register is pointing to the last data placed on the stack. The instruction `LDD 1, Y` will load the value of the local variable 'x' into the D accumulator. To access the parameter 'Num,' the instruction `LDD 7, Y` can be used. Any instructions that support the indexed addressing mode can be used to manipulate stack data.

PASSING PARAMETERS

Parameters are easily placed on the M68HC11 stack by CPU push instructions. Table 2 lists the push instructions available on the M68HC11. Note that there is not a single instruction for pushing the D accumulator onto the stack. A PSHD instruction can easily be simulated by executing the two instructions PSHB, PSHA. These two instructions must be executed in this order to keep the value pushed onto the stack consistent with the way 16-bit values are stored in memory — i.e., 16-bit values are placed in memory with the most significant eight bits at a lower address than the least significant eight bits. By following this convention, a 16-bit parameter pushed onto the stack in this manner is easily retrieved using one of the 16-bit load instructions.

Table 2. Push Instructions in the M68HC11 Instruction Set

Instruction Mnemonic	Description
PSHA	Push Accumulator A onto the Stack.
PSHB	Push Accumulator B onto the Stack.
PSHX	Push Index Register X onto the Stack.
PSHY	Push Index Register Y onto the Stack.

As previously mentioned, parameters can be passed either by value or by reference. Consider a function, `Int2Asc`, that converts a signed 16-bit integer to ASCII text and places the ASCII characters in a text buffer. The function requires two parameters: the number to be converted into ASCII text and

a pointer to a buffer where the ASCII text is to be stored. The first parameter is passed to the subroutine by value because the actual number to be converted is passed to the function. The second parameter is passed by reference because a pointer to the buffer is passed to the routine and not the buffer itself. A function declaration written in C is shown in Listing 6.

```
void Int2Asc(int Num; char *Buff)
{
    int Pwr10 = 10000;
    char zs = 0;
    .
    .
    .
}
```

Listing 6. Function Declaration of Int2Asc

Before calling an equivalent routine written in M68HC11 assembly language, the two parameters will be pushed onto the stack as shown in Listing 7.

```
LDX      ErrorNum      ; Get the value of the current error.
PSHZ                      ; Place it on the stack.
LDX      #OutBuff      ; Get the address of the Output buffer.
PSHZ                      ; Place it on the stack.
JSP      Int2Asc        ; Go convert the number.
```

Listing 7. Placing Parameters on the M68HC11 Stack

Using the immediate addressing mode with the second load index register X (LDX) instruction loads the address of OutBuff into the X index register rather than the 16-bit value contained in the memory locations OutBuff and OutBuff+1. After both parameters have been pushed onto the stack, the function is called with a JSR instruction. Upon entry to the subroutine Int2Asc, the parameters reside just above the return address as shown in Figure 9.

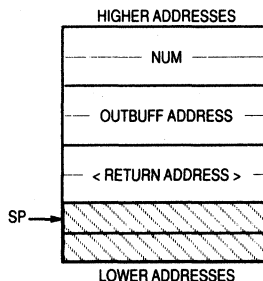


Figure 9. Location of Parameters Passed on the Stack

ALLOCATING LOCAL VARIABLES

Four basic techniques can be used to allocate local variables that reside on the stack. Choosing which one to use depends upon the total amount of storage required for the local variables and whether or

not the variables need to have an initial value assigned to them. Of course, a combination of all four techniques can be used.

One technique used to allocate space on the stack for local storage involves the use of the decrement stack pointer (DES) instruction. The DES instruction subtracts one from the value of the SP each time the instruction is executed, allocating one byte of local variable storage for each DES instruction. This technique is a simple and direct way of allocating local storage but becomes impractical when large amounts of local storage are required. For instance, if 100 bytes of local storage are required for a subroutine, 100 DES instructions are needed to allocate the required amount of storage. This required amount is clearly unacceptable since each DES instruction requires one byte of program memory. Even if a small program loop is set up to execute 100 DES instructions, the subroutine will suffer a severe execution speed penalty each time the routine is entered.

Using the previously described technique requires one byte of program storage for each byte of local storage that is allocated. Since allocating local storage simply involves decrementing the SP, the PSHX instruction can be used to allocate two bytes of local storage space for each executed PSHX instruction. The actual contents of the X index register are irrelevant because the only concern is decrementing the SP. The use of this technique can be confusing if not properly documented since it is not directly obvious what is being accomplished with five or six sequentially executed PSHX instructions.

Many times it is necessary to initialize local variables with a particular value before they are used. The same technique used to push parameters onto the stack before a subroutine call can also be used to allocate space for local variables and simultaneously assign initial values to them. This procedure is accomplished by loading one of the CPU registers with a variable's initial value and executing a PSH instruction. The program fragment in Listing 8 shows the use of this technique to allocate and initialize both an 8- and 16-bit local variable.

```
Int2Asc      equ      *
             .
             .
             .
             ldx      #10000      ; get the initial value of Pwr10.
             pshx     ; allocate and initialize it.
             clra    ; initial value of zs is zero.
             psha    ; allocate and initialize it.
```

Listing 8. Allocating and Initializing Local Variables

If more than 13 bytes of local storage are required by a subroutine, a fourth technique allocates storage more efficiently than using multiple DES or PSHX instructions. Since there are not any instructions that allow arithmetic to be performed directly on the SP, the fourth technique involves using several M68HC11 instructions. These instructions adjust the value of the SP downward in memory, allocating the required amount of local storage. Listing 9 shows the instruction sequence required to allocate an arbitrary number of bytes of local storage.

```
SinCos      equ      *
             .
             .
             .
             tsx     ; SP+1 → X.
             xgdx   ; exchange the contents of x and d.
             subd   #xxxx ; subtract the required amt. of storage.
             xgdx   ; place the result back into x.
             txs    ; X-1 → SP. Update the SP.
```

Listing 9. Allocation of More Than 13 Bytes for Local Storage

Since no single instruction allows the contents of the SP to be transferred to the D accumulator, the two-instruction sequence transfer from SP to index register X or Y; exchange double accumulator and index register X or Y (TSX; XGDX or TSY; XGDY) must be used. Placing the SP value in the D accumulator allows the use of the 16-bit subtract instruction to adjust the value of the SP. The subtract double accumulator (SUBD) instruction will subtract the 16-bit value xxxx from the contents of the D accumulator. To place this new value in the SP, the two-instruction sequence XGDX; TXS or XGDY; TYS is used.

NOTE

Actually the TSX or TSY instruction causes the SP value plus one to be transferred to either the X or Y index register ($SP + 1 \rightarrow X$ or $SP + 1 \rightarrow Y$). This transfer does not pose a problem because when the SP is updated with the TXS or TYS instruction, one is subtracted from the value of the index register ($X - 1 \rightarrow SP$ or $Y - 1 \rightarrow SP$) before the SP is updated. Remember that since the SP points to the next available location on the stack, adding one to its value before the execution of the TSX or TSY instruction makes the X or Y index register point to the last data placed on the stack.

CREATING A COMPLETE STACK FRAME

In addition to providing storage space for local variables and parameters, a complete stack frame (sometimes called an activation record) must contain two additional pieces of information: a return address and a pointer to the base of the stack frame of any previous routines. The return address is placed on the stack automatically by the M68HC11 when it executes either a JSR or BSR instruction. As shown in Figure 9, the return address is placed on the stack just below a subroutine's parameters.

Before using either the X or Y index register to access a routine's parameters or local variables, the contents of the register must first be saved. The index register contents, known as the stack frame pointer, may contain the base address of a stack frame for a routine from which control was transferred. This pointer must be maintained so that when control is returned to the calling routine, the calling routine's environment can be restored to its previous state. Even if a routine has no local variables or parameters, the contents of the index register being used as the stack frame pointer must be saved before the register is used for any other purpose.

The best time to save the value of the previous stack frame pointer is immediately upon entry to a subroutine, which places the previous stack frame pointer immediately below the return address as shown in Figure 10.

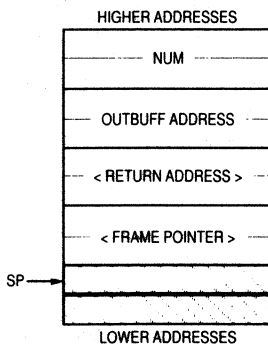


Figure 10. Location of the Stack Frame Pointer

After space for local variables has been allocated, the stack frame pointer for the new subroutine needs to be initialized. By transferring the contents of the SP to either the X or Y index register using the TSX or TSY instruction, a new stack frame is created.

In summary, creating a complete stack frame involves the following three steps after entering a subroutine:

1. Immediately upon entry to a subroutine, the contents of the index register being used as the stack frame pointer must be saved by using either the PSHX or PSHY instruction.
2. Storage space for the routine's local variables should be allocated using one of the three methods described earlier.
3. The new stack frame pointer must be initialized using either the TSX or TSY instruction.

The last issue to discuss is which index register to use as the stack frame pointer. In terms of code size and speed, the X index register would be the most logical choice since ALL instructions involving the Y index register require one additional opcode byte and one additional clock cycle to execute. However, if a program is not making extensive use of the stack for local variables and parameters but is performing extensive array or table manipulations, the Y index register may be a better choice. No matter which index register is used as the stack frame pointer, it should be, if at all possible, dedicated to that use throughout a program. Program debugging is much easier if the contents of a single index register can always be expected to point to the current stack frame.

ACCESSING PARAMETERS AND LOCAL VARIABLES

As mentioned in **USING THE M68HC11 STACK**, local variables and parameters are accessed by using instructions that support the indexed addressing mode. The following list identifies the load and store instructions as well as all arithmetic and logic instructions that support indexed addressing. Because most M68HC11 instructions support indexed addressing, it is just as code efficient to manipulate local variables that reside on the stack as it is to manipulate global variables using direct or extended addressing. Figure 11(a) illustrates a complete allocation frame as used by a subroutine.

ADCA	ADCB	ADDA	ADDB	ADDD
ANDA	ANDB	ASL	ASR	BCLR
BITA	BITB	BRCLR	BRSET	BSET
CLR	CMPA	CMPB	COM	CPD
CPX	CPY	DEC	EORA	EORB
INC	JMP	JSR	LDAA	LDAB
LDD	LDS	LDX	LDY	LSL
LSR	NEG	ORA	ORB	ROL
POR	SBCA	SBCB	STAA	STAB
STD	STS	STX	STY	SUBA
SUBB	SUBD	TST		

Using the indexed addressing mode to access data contained in a stack frame places a restriction on the combined size of local variables and parameters. Since the indexed addressing mode functions by adding an unsigned 8-bit offset to the contents of the 16-bit index register, the indexed addressing mode can only access information at addresses that are between 0 and 255 bytes greater than the base address contained in one of the index registers. Consequently, the maximum size of a single stack frame is restricted to 256 bytes. If no parameters are passed to a routine on the stack, then the entire 256 bytes are available for local variables. However, when parameters are passed on the stack, not only is the space occupied by the parameters unavailable for use as local variables, but the subroutine return address and previous stack frame pointer reduce the amount of available space by an additional four bytes.

In most embedded control applications that use the M68HC11 in the single-chip mode, this limit on the combined size of parameters and local variables for a single stack frame is rarely a concern since the amount of on-chip RAM is limited. Several techniques can be used to work around the limit imposed by the indexed addressing mode; however, they are extremely wasteful in terms of code space and execution speed.

NOTE

In reality, the amount of memory available for local storage in a single stack frame is 257 bytes. Because the M68HC11 is capable of loading and storing 16 bits of data with a single instruction, it is possible to access one byte beyond the contents of the index register plus the fixed offset of 255 with the 16-bit load and store instructions.

DEALLOCATING THE STACK FRAME

When a subroutine has completed execution, the stack space allocated for the stack frame must be released so the memory can be reused by subsequent subroutine calls. The deallocation of the stack frame includes not only the removal of the space occupied by the local storage, but also the restoration of the previous stack frame pointer and the removal of space occupied by any parameters that were passed to the subroutine.

The process of freeing the memory occupied by the stack frame is simply a matter of adjusting the value of the SP upward in memory. The SP must be adjusted upward by the same amount that it was adjusted downward when the space for the stack frame was allocated. Either of the following methods can be used to perform this task.

The most obvious way to perform the deallocation is to reverse the process used to allocate the storage. Removing the stack frame in this manner involves three basic steps. First, the storage occupied by any local variables must be removed from the stack area by using the reverse of one of the techniques described in **ALLOCATING LOCAL VARIABLES**. Alternately, the technique shown in Listing 10 can be used. This technique involves adjusting the value of the SP upward in memory by the same amount it was adjusted downward when the space was allocated.

```
LDAB    #LOCLN    Get size of local storage into the B register.
ABX     Add it to the current stack frame pointer.
TXS     Deallocate the local storage.
```

Listing 10. Alternate Method for Deallocating Local Storage

Second, the previous stack frame pointer must be restored. Because the previous stack frame pointer is now on the top of the stack, the use of a pull index register X or Y from the stack (PULX or PULY) instruction is all that is needed to perform this operation. At this point, the return address is on the top of the stack. Simply executing a return from subroutine (RTS) instruction returns program execution to the instruction following the subroutine call.

After returning to the calling routine, any parameters that were pushed onto the stack before the subroutine call must now be removed. This places the burden of removing subroutine parameters on the calling routine rather than on the called routine. This method of removing subroutine parameters is perfectly acceptable and is the one most often used by C language compilers.

Removing the parameters can be as simple as a one-instruction operation. If the X or Y index register contains the address of the current stack frame pointer, simply executing a TXS or TYS instruction places the SP just below the stack frame pointer. If the X or Y index register does not contain the

address of the current stack frame pointer, an alternate method must be used to remove the parameters. Figure 11 illustrates the state of the stack at each stage of the deallocation process.

An alternate method requires the called routine to remove the entire stack frame, including any parameters passed to it. This method may not be as code efficient as the first method since it requires a fixed number of instructions to release the storage space occupied by the entire stack frame. Listing 11 shows the instruction sequence necessary to deallocate the stack frame when the X index register is being used as the stack frame pointer. This four-instruction sequence requires nine bytes of program storage space and 18 cycles to execute but removes the entire stack frame, regardless of the size. This method of stack frame deallocation has one drawback — the X or Y index register must always contain a valid stack frame pointer. Thus, all subroutines, even if they do not require parameters or local variables, must “mark” the current state of the stack upon entry by executing a PSHX; TSX or PSHY; TSY instruction sequence.

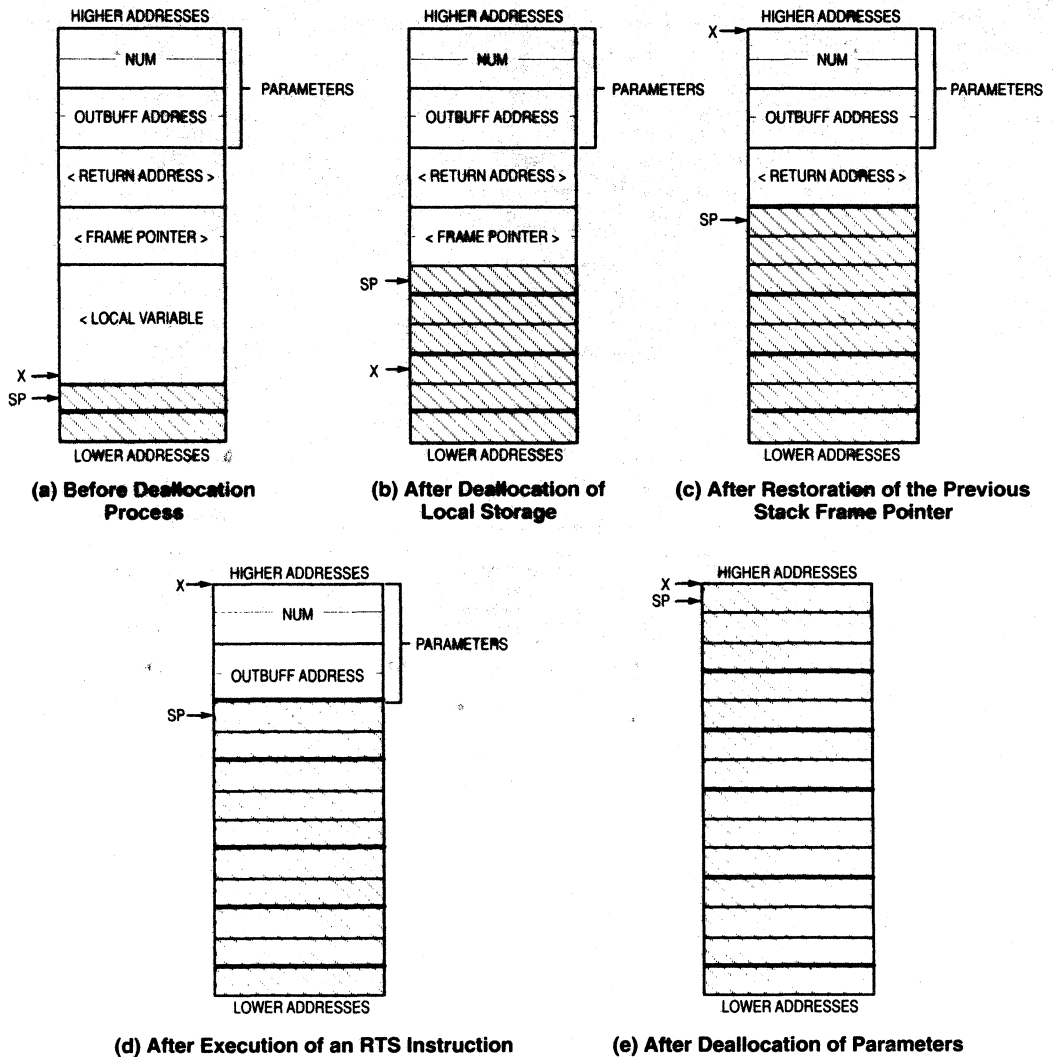


Figure 11. Deallocation of the Stack Frame

NOTE

In Listing 11, RA is the offset value to the <Return Address> and PSFP is the offset value to the <Previous Stack Frame Pointer>.

LDY	RA,X	Load the return address into the Y register.
LDX	PSFP,X	Restore the previous stack frame pointer.
TXS		Remove the entire stack frame.
JMP	0,Y	Return to the calling routine.

Listing 11. Alternate Method for Deallocating the Entire Stack Frame

In summary, choosing a method to deallocate the stack frame involves a tradeoff between code size and execution speed. Using the first method results in the smallest amount of code being generated but may take longer to execute than the method shown in Listing 11.

SUPPORT MACROS

The following macros may be used to help in managing stack frames in M68HC11 programs. Using these macros may not provide the smallest or fastest code in all situations but should make the program easier to write and debug. Although the macros were written for the Micro Dialects μ ASM-HC11™ assembler that runs on the Macintosh™, they can be used with other assemblers with some modification. The following paragraph explains the way parameters are passed and referenced in the Micro Dialects assembler and should help in the conversion process.

When a macro is defined, parameters are not declared. When a macro is invoked, the parameters appear in the operand field following the macro name. Within a macro definition, parameters are referenced by using a colon (:) followed by a single decimal digit (0–9). Therefore, within the body of the macro, the first parameter is referenced by using ':0', the second parameter is referenced by using ':1', and so forth. Parameter substitution is performed strictly on a textual substitution basis.

The link macro shown in Listing 12 can be used to allocate a complete stack frame after entry into a subroutine. The link macro performs the following functions: 1) saves the previous stack frame pointer, 2) allocates the required number of bytes of local storage, and 3) initializes a new stack frame pointer. The calling convention for the link macro is as follows:

```
link    <s.f. reg>,<storage bytes>
```

The first parameter passed to the macro is the name of the index register being used as the stack frame pointer (either X or Y). Although no check is made to ensure that a legal index register name is passed to the macro, the assembler will produce an "Unrecognized Mnemonic" error message when the macro is expanded. The second parameter is the number of bytes of local storage required by the subroutine.

```
link    macro
psrh:0  ; Save the previous stack frame pointer.
r:s:0   ; Transfer the stack pointer into :0.
xgd:0   ; Transfer :0 into D.
subd =:1 ; subtract the required amount of local storage.
xgd:0   ; Initialize the new stack frame pointer
t:0s    ; Update the stack pointer with new value.
endm
```

Listing 12. The Link Macro

The return and deallocate (rtd) macro shown in Listing 13 can be used to partially deallocate a subroutine stack frame. The rtd macro performs the following functions: 1) deallocates local storage, 2) restores the previous stack frame pointer, and 3) returns to the calling routine. The rtd macro DOES NOT remove any parameters from the stack that may have been passed to the subroutine. Removal of any parameters must be performed by the calling routine. This macro is useful when no parameters are passed to a subroutine or when parameters are passed in registers. The calling convention for the rtd macro is as follows:

```
rtd    <s.f. reg>,<storage bytes>
```

Like the link macro, the first parameter passed to the rtd macro is the name of the index register being used as the stack frame pointer (either X or Y). Again, although no check is made to ensure that a legal index register name is passed to the macro, the assembler will produce an "Unrecognized Mnemonic" error message when the macro is expanded. The second parameter is the number of bytes of local storage allocated when the subroutine was entered.

```
rtd      macro
        ldab  #:1      ; number of bytes to deallocate.
        ab:0      ; add it to the current stack frame pointer.
        t:0s     ; deallocate storage by updating the stack pointer.
        pul:0    ; restore the previous stack frame pointer.
        rts     ; return to the calling routine.
        endm
```

Listing 13. The Return and Deallocate Macro

The only drawback in using this macro is that it uses the B accumulator when deallocating a subroutine's local storage, preventing a subroutine from returning a 16-bit result in the D accumulator. A simple solution to the problem is to surround the load accumulator B (LDAB) and add accumulator B to index register X or Y (ABX/ABY) instructions with the PSHB/PULB instruction pair as shown in Listing 14. This macro, renamed frtd for function return and deallocate, allows the D accumulator to be loaded with a return value immediately before the macro is called. A second solution to this problem is to place all return values on the stack as described in **FUNCTION/SUBROUTINE RETURN VALUES**, allowing the calling routine to retrieve the returned value and then remove it along with the parameters.

```
frtd     macro
        pshb     ; save the lower byte of the return value.
        ldab  #:1      ; number of bytes to deallocate.
        ab:0      ; add it to the current stack frame pointer.
        pulb     ; restore the lower byte of the return value.
        t:0s     ; deallocate storage by updating the stack pointer.
        pul:0    ; restore the previous stack frame pointer.
        rts     ; return to the calling routine.
        endm
```

Listing 14. The Function Return and Deallocate Macro

The return and deallocate using x (rtdx) and return and deallocate using y (rtdy) macros shown in Listing 15 can be used to completely deallocate a subroutine stack frame, including any parameters that were passed on the stack. The rtdx and rtdy macros perform the following functions: 1) deallocates the entire stack frame, including local storage and passed parameters, 2) restores the previous stack frame pointer, and 3) returns to the calling routine. The calling convention for the rtdx and rtdy macros is as follows:

```
rtdx    <storage bytes> or rtdy <storage bytes>
```

The only parameter passed to the macros is the number of local storage bytes allocated upon entry to the subroutine. These macros have an advantage over the rtd macro in that the A and B accumulators are not used during deallocation, which allows a return value to be loaded into the A, B, or D registers before execution of the rtdx or rtdy macro.

```

rtdx      macro
          ldy  :0+2,x ; Load the return address into the Y index register.
          ldx  :0,x   ; restore the previous stack frame pointer.
          txs                ; Update the stack pointer, removing the storage space.
          jmp  0,y   ; Return to the calling routine.
          endm
*
*
rtdy      macro
          ldx  :0+2,y ; Load the return address into the X index register.
          ldy  :0,y   ; restore the previous stack frame pointer.
          tys                ; Update the stack pointer, removing the storage space.
          jmp  0,x   ; Return to the calling routine.
          endm

```

Listing 15. The rtdx and rtdy Macro

The only restriction to using the rtdx and rtdy macros is that a valid stack frame pointer for the previous subroutine must be present in either the X or Y index register when the register is pushed onto the stack at the beginning of the subroutine. Even if a subroutine has no local variables in it or no parameters passed to it, a PSHX and TSX instruction must be executed immediately upon entry to a subroutine to save the previous stack frame pointer and “mark” the current state of the stack. Before returning, a PULX instruction must be executed to restore the previous stack frame pointer.

This restriction implies that, somewhere in the program, the index register to be used as the stack frame pointer must be initialized with a valid value. If either the X or Y index register is to be dedicated for use as a stack frame pointer, the index register must be initialized at the beginning of the program. The initial value loaded into the index register should be one more than the value loaded into the stack pointer, which is easily accomplished by executing the TSX instruction immediately after initializing the stack pointer.

In summary, the use of the rtdx and rtdy macros are convenient in that they remove both parameters and local variables passed to subroutines. However, their use will cost three extra instructions in subroutines that do not have local variables or parameters but call subroutines that use local variables or have parameters passed to them.

EXAMPLES

Appendix A contains several examples that use the techniques described to manage local storage, parameter passing, and allocation/deallocation of stack frames.

APPENDIX A EXAMPLE LISTINGS

```

1                               Include "Stack Macros"
2                               *****
3                               *
4                               *
5                               *           Written By
6                               *           Gordon Doughman
7                               *           For
8                               *           Motorola Semiconductor
9                               *
10                              *           The author reserves the right to make changes to this file. Although this
11                              *           software has been carefully reviewed and is believed to be reliable, neither
12                              *           Motorola nor the author assumes any liability arising from its use. This soft-
13                              *           ware may be freely used and/or modified at no cost or obligation to the user.
14                              *
15                              *           The following macros may be used to help in managing stack frames in
16                              *           M68HC11 programs. The macros were written for Micro Dialects μASM-HC11
17                              *           assembler that runs on the Macintosh but may be used with other assemblers
18                              *           with some modification. The following discussion of the way parameters are
19                              *           passed and referenced should help in the conversion process.
20                              *
21                              *           Within a macro, parameters are referenced by using a colon (:) followed
22                              *           by a single decimal digit (0-9). Therefore, within the body of the macro
23                              *           the first parameter is referenced by using ':0', the second parameter is
24                              *           referenced by using ':1', and so forth. Parameter substitution is performed
25                              *           strictly on a textual substitution basis.
26                              *
27                              *
28                              *           The link macro may be used to allocate a complete stack frame after entry
29                              *           into a subroutine. The link macro performs the following functions:
30                              *           1) Saves the previous stack frame pointer; 2) Allocates the requested
31                              *           number of bytes of local storage; 3) Initializes a new stack frame pointer.
32                              *
33                              *           Usage:      link <s.f. reg>,<storage bytes>
34                              *
35                              *           The first parameter passed to link is the index register that is being used
36                              *           as the stack frame pointer (either x or y). Although no check is made to
37                              *           ensure that a legal index register name is passed to the macro, the assembler
38                              *           will produce an "Unrecognized Mnemonic" error message when the macro is
39                              *           expanded. The second parameter is the number of bytes of local storage
40                              *           required by the subroutine.
41                              *
42                              *
43                              *
44 M          link      macro
45 M              psh:0          ; Save the previous stack frame pointer.
46 M              ts:0          ; Transfer the stack pointer into :0.
47 M              xgd:0         ; Transfer :0 into D.
48 M              subd #:1      ; subtract the required amount of local storage.
49 M              xgd:0         ; Initialize the new stack frame pointer
50 M              t:0s          ; Update the stack pointer with new value.
51 M          endm
52          *
53          *
54          *
55          *
56          *           The rtd (Return and Deallocate) macro may be used to partially deallocate
57          *           a subroutine stack frame that includes parameters passed on the stack. The
58          *           rtd macro performs the following functions: 1) Deallocates local storage;
59          *           2) Restores the previous stack frame pointer; 3) Returns to the calling
60          *           routine. Rtd DOES NOT remove any parameters from the stack. This function
61          *           must be performed by the calling routine. This macro is useful when
62          *           parameters are passed in registers rather than on the stack.
63          *
64          *           Usage:      rtd <s.f. reg>,<storage bytes>
65          *
66          *           The first parameter passed to link is the index register that is being used
67          *           as the stack frame pointer (either x or y). Although no check is made to
68          *           ensure that a legal index register name is passed to the macro, the assembler
69          *           will produce an "Unrecognized Mnemonic" error message when the macro is
70          *           expanded. The second parameter is the number of bytes of local storage

```

```

71      *      used by the subroutine.
72      *
73      *
74      *
75 M    rtd      macro
76 M      ldab   #:1          ; number of bytes to deallocate.
77 M      ab:0          ; add it to the current stack frame pointer.
78 M      t:0s        ; deallocate storage by updating the stack pointer.
79 M      pul:0       ; restore the previous stack frame pointer.
80 M      rts         ; return to the calling routine.
81 M    endm
82      *
83      *
84      *
85      *
86      *      The frtd (Function Return and Deallocate) macro may be used to partially
87      *      deallocate a subroutine stack frame that includes parameters passed on
88      *      the stack. The frtd macro performs the following functions: 1) Deallocates
89      *      local storage; 2) Restores the previous stack frame pointer; 3) Returns
90      *      to the calling routine. Frtd DOES NOT remove any parameters from the stack.
91      *      This function must be performed by the calling routine. This macro is
92      *      useful when parameters are passed in registers rather than on the stack and
93      *      a value is being returned in the D-accumulator.
94      *
95      *      Usage:      frtd <s.f. reg>,<storage bytes>
96      *
97      *      The first parameter passed to frtd is the index register that is being used
98      *      as the stack frame pointer (either x or y). Although no check is made to
99      *      ensure that a legal index register name is passed to the macro, the assembler
100     *      will produce an "Unrecognized Mnemonic" error message when the macro is
101     *      expanded. The second parameter is the number of bytes of local storage
102     *      used by the subroutine.
103     *
104     *
105     *
106 M    frtd     macro
107 M      pshb          ; save the lower byte of the return value.
108 M      ldab   #:1          ; number of bytes to deallocate.
109 M      ab:0          ; add it to the current stack frame pointer.
110 M      pulb          ; restore the lower byte of the return value.
111 M      t:0s        ; deallocate storage by updating the stack pointer.
112 M      pul:0       ; restore the previous stack frame pointer.
113 M      rts         ; return to the calling routine.
114 M    endm
115     *
116     *
117     *
118     *
119     *      The rtdx and rtdy (Return and Deallocate using x or y) macros may be used
120     *      to completely deallocate a subroutine stack frame including parameters that
121     *      were passed on the stack. The rtdx macro performs the following functions:
122     *      1) Deallocates the entire stack frame including local storage and passed
123     *      parameters; 2) Restores the previous stack frame pointer; and 3) Returns
124     *      to the calling routine.
125     *
126     *      Usage:      rtdx <storage bytes>
127     *      Usage:      rtdy <storage bytes>
128     *
129     *      The only parameter passed to the routines is the number of bytes of local storage
130     *      that were originally allocated upon entry to the subroutine. These macros have
131     *      the advantage that the a and b accumulators are not used during the deallocation
132     *      process. This allows a value to be loaded into the a, b, or d registers before
133     *      the execution of the rtdx or rtdy macro and returned to the calling routine.
134     *
135     *
136     *
137 M    rtdx     macro
138 M      ld y      :0+2,x      ; Load the return address into the y-index register.
139 M      ld x      :0,x        ; restore the previous stack frame pointer.
140 M      txs         ; Update the stack pointer, removing the storage space.
141 M      jmp      0,y        ; Return to the calling routine.
142 M    endm
143     *
144     *

```



```

145 M      rtdy  macro
146 M          ldx   :0+2,y      ; load the return address into the x-index register.
147 M          ldy   :0,y      ; restore the previous stack frame pointer.
148 M          tys           ; Update the stack pointer, removing the storage space.
149 M          jmp   0,x      ; Return to the calling routine.
150 M      endm
151      *
152      *
153      *
154      *
155      *      The pshd macro pushes the 16-bit d-accumulator onto the stack. The
156      *      b-accumulator is pushed first so that the least significant 8-bits of
157      *      the 16-bit number appear on the stack at the higher address. This is
158      *      consistent with the way all 16-bit numbers are stored in memory.
159      *
160      *      Usage:      pshd
161      *
162      *      No parameters are required by the macro.
163      *
164      *
165      *
166 M      pshd  macro
167 M          pshb
168 M          psha
169 M      endm
170      *
171      *
172      *
173      *
174      *      The puld macro pulls the top two bytes from the stack and places them in
175      *      the 16-bit d-accumulator. The first byte pulled from the stack is placed
176      *      in the a-accumulator; the second byte pulled from the stack is placed in
177      *      the b-accumulator. The pull order is consistent with the way all 16-bit
178      *      numbers are stored in memory.
179      *
180      *      Usage:      puld
181      *
182      *      No parameters are required by the macro.
183      *
184      *
185      *
186 M      puld  macro
187 M          pula
188 M          pulb
189 M      endm
190      *
191      *
192      *
193      *
194      *      The clr d macro uses the clra and clrb instructions to clear the 16-bit
195      *      d-accumulator.
196      *
197      *      Usage:      clr d
198      *
199      *      No parameters are required by the macro.
200      *
201      *
202      *
203 M      clr d macro
204 M          clra
205 M          clrb
206 M      endm
207      *
208      *
209      *
210      *

```

```

211 *
212 *
213 *           Written By
214 *           Gordon Doughman
215 *           For
216 *           Motorola Semiconductor
217 *
218 * The author reserves the right to make changes to this file. Although this
219 * software has been carefully reviewed and is believed to be reliable, neither
220 * Motorola nor the author assumes any liability arising from its use. This soft-
221 * ware may be freely used and/or modified at no cost or obligation to the user.
222 *
223 *
224 *-----*
225 *
226 * This subroutine converts a 16-bit binary integer to a null terminated
227 * ASCII string. Three parameters are passed to the subroutine on the
228 * stack. The first parameter is the 16-bit binary number to be converted.
229 * The second parameter is the address of a buffer where the null terminated
230 * ASCII string will be placed. The buffer should be at least 7 bytes long.
231 * The third parameter is a boolean flag indicating whether the number passed
232 * in the first parameter is a signed or unsigned 16-bit number. If the byte
233 * flag is zero, the number is converted as an unsigned number. If the byte
234 * is non-zero, the number will be converted as a 16-bit signed number.
235 * Parameters are pushed onto the stack in the following order: 1) Signed Flag;
236 * 2) Pointer to ASCII buffer; 3) Number to be converted. A typical
237 * calling sequence would be:
238 *
239 *      clrA                ; Do the conversion as an unsigned number.
240 *      pshA                ; put the flag on the stack.
241 *      ldd    #Buffer      ; get the address of the ascii buffer.
242 *      pshD                ; put the address on the stack.
243 *      ldd    Num          ; Get the number to convert.
244 *      pshD                ; Put it on the stack
245 *      jsr   Int2Asc       ; Go convert the number.
246 *      .
247 *      .
248 *      .
249 *
250 * This subroutine has two local variables. The first, zs, is a boolean variable
251 * used to suppress leading zeros when doing a conversion. It is located at an
252 * offset of 0 from the stack frame pointer. The second local, Divisor, is a 16-bit
253 * variable. It is used to divide the number being converted by successively lower
254 * powers of 10. Divisor is located at an offset of 1 from the local stack frame
255 * pointer.
256 *
257 * NOTE: This routine was written assuming that the previous stack frame pointer
258 * is the x-index register. HOWEVER, because the x-index register is required
259 * by the integer divide instruction, the y-index register is used as the
260 * stack frame pointer WITHIN the Int2Asc subroutine.
261 *
262 *
263 *      Declare locals
264 *
265 0000 PCSave    set    *           ; save the current PC value
266 0000                org    0           ; set PC to 0 for offsets to locals
267 0000 zs        rmb    1           ; declare zs variable.
268 0001 Divisor   rmb    2           ; declare Divisor variable.
269 0003 LocSize   set    *           ; number of bytes of local storage.
270 0000                org    PCSave
271 *
272 *      Offsets to parameters
273 *
274 0007 Num       equ    LocSize+4     ; offset to Num parameter.
275 0009 BuffP     equ    LocSize+6     ; offset to BuffP parameter.
276 000B Signed    equ    LocSize+8     ; offset to Signed parameter.
277 *
278 0000 Int2Asc    equ    *
279 0000 3C        [ 4] pshx           ; save the previous stack frame pointer.
280 0001 CC2710   [ 3] ldd    #10000    ; initialize the divisor to 10000.
281 0004                pshd
282 0004 37        [ 3] pshb
283 0005 36        [ 3] psha
284 0006 4F        [ 2] clra           ; initialize zs to 0.

```

```

285 0007 36      [ 3]      psha
286 0008 1830   [ 4]      tsy          ; initialize the new stack frame pointer.
287 000A 18EC07 [ 6]      ldd      num,y      ; get the number to convert. Is it zero?
288 000D 260B   [ 3]      bne      Int2Asc1   ; no go do the conversion.
289 000F CC3000 [ 3]      ldd      #$3000     ; yes.
290 0012 CDEE09 [ 6]      ldx      BuffP,y    ; point to the buffer.
291 0015 18ED00 [ 6]      std      0,y        ; just put an ASCII 0 in the buffer.
292 0018 2050   [ 3]      bra      Int2Asc5    ; then return.
293 001A 186D0B [ 7]      Int2Asc1 tst      Signed,y  ; do the conversion as a signed number?
294 001D 2716   [ 3]      beq      Int2Asc2   ; no.
295 001F 4D     [ 2]      tsta     ; yes. Is the number negative?
296 0020 2A13   [ 3]      bpl      Int2Asc2   ; no. just go do the conversion.
297 0022 43     [ 2]      coma    ; yes. make it a positive number by negation.
298 0023 53     [ 2]      comb
299 0024 C30001 [ 4]      addd    #s1
300 0027 18ED07 [ 6]      std      Num,y      ; save the result.
301 002A 862D   [ 2]      ldaa    #'-'        ; get an ASCII minus sign.
302 002C CDEE09 [ 6]      ldx      BuffP,y    ; point to the buffer.
303 002F A700   [ 4]      staa    0,x        ; put it in the buffer.
304 0031 08     [ 3]      inx     ; point to the next location in the buffer.
305 0032 CDEF09 [ 6]      stx     BuffP,y    ; save the new pointer value.
306 0035 18EC07 [ 6]      Int2Asc2 ldd      Num,y      ; get the remainder to convert.
307 0038 CDEE01 [ 6]      ldx      Divisor,y
308 003B 02     [41]      idiv
309 003C 18ED07 [ 6]      std      Num,y      ; save the remainder.
310 003F 8F     [ 3]      xgdx    ; put the dividend into d.
311 0040 5D     [ 2]      tstb    ; was the dividend 0?
312 0041 2605   [ 3]      bne      Int2Asc3   ; no. go store the number in the buffer.
313 0043 186D00 [ 7]      tst      zs,y       ; are we still suppressing leading zeros?
314 0046 2710   [ 3]      beq      Int2Asc4   ; yes. go setup for the next divide.
315 0048 CB30   [ 2]      Int2Asc3 addb    #'0'    ; make the dividend ASCII.
316 004A 8601   [ 2]      ldaa    #1
317 004C 18A700 [ 5]      staa    zs,y       ; don't suppress leading zeros anymore.
318 004F CDEE09 [ 6]      ldx      BuffP,y    ; get a pointer to the buffer.
319 0052 E700   [ 4]      stab    0,x        ; save the digit.
320 0054 08     [ 3]      inx     ; point to the next location.
321 0055 CDEF09 [ 6]      stx     BuffP,y    ; save the new pointer value.
322 0058 18EC01 [ 6]      Int2Asc4 ldd      Divisor,y  ; get the previous divisor.
323 005B CE000A [ 3]      ldx      #10
324 005E 02     [41]      idiv    ; divide it by 10.
325 005F CDEF01 [ 6]      stx     Divisor,y  ; save the dividend. Is it zero?
326 0062 26D1   [ 3]      bne      Int2Asc2   ; no. continue with the conversion.
327 0064 CDEE09 [ 6]      ldx      BuffP,y    ; get a pointer to the buffer.
328 0067 6F00   [ 6]      clr     0,x        ; null terminate the string.
329 0069 30     [ 3]      tsx     ; this is only needed because we are using y as our
                                     ; sf pointer.
330 006A          Int2Asc5 rtdx    LocSize ; return & deallocate locals & parameters.
331 006A 1AE05   [ 6]      ldy     LocSize+2,x ; Load the return address into the y-index register.
332 006D EE03   [ 5]      ldx     LocSize,x  ; restore the previous stack frame pointer.
333 006F 35     [ 3]      txs     ; Update the stack pointer, removing the storage
                                     ; space.
334 0070 186E00 [ 4]      jmp     0,y        ; Return to the calling routine.
335 *
336 *
337 *
338 *
339 *
340 * This subroutine performs a 16 x 16 bit unsigned multiply and produces a 32-bit
341 * result. Two 16-bit numbers are passed to the subroutine on the stack.
342 * The 32-bit result is returned on the stack in place of the two 16-bit
343 * parameters. This allows the calling routine to easily pull the product
344 * from the stack and store the result. Because multiplication is a
345 * commutative operation, the order in which the parameters are pushed
346 * onto the stack is unimportant. A typical calling sequence would be:
347 *
348 * ldd      Num1
349 * pshd
350 * ldd      Num2
351 * pshd
352 * jsr     Mull6x16
353 * puld
354 * std     Result32
355 * puld
356 * std     Result32+2

```

```

357 * .
358 * .
359 * .
360 *
361 * This subroutine has four local variables. Each variable occupies 1 byte
362 * on the stack. These four bytes are used to hold the partial product as
363 * the final answer is being computed. These four byte variables are
364 * treated as 16-bit variables during the calculation.
365 *
366 * NOTE: This routine was written assuming that the stack frame pointer
367 * is the x-index register.
368 *
369 * Declare locals
370 *
371 0073 PCSave set * ; save the current PC value
372 0000 org 0 ; set PC to 0 for offsets to locals
373 0000 Prd0 rmb 1 ; declare ms byte of partial product variable.
374 0001 Prd1 rmb 1 ; declare next ms byte of partial product variable.
375 0002 Prd2 rmb 1 ; declare next ls byte of partial product variable.
376 0003 Prd3 rmb 1 ; declare ls byte of partial product variable.
377 0004 LocSize set * ; number of bytes of local storage.
378 0073 PCSave org PCSave
379 *
380 * Offsets to parameters
381 *
382 0008 Fact1 equ LocSize+4 ; offset to factor 1 parameter.
383 000A Fact2 equ LocSize+6 ; offset to factor 2 parameter.
384 *
385 * cycles clear
386 *
387 0073 Mull6x16 equ *
388 0073 3C [ 4] pshx clrd ; save the previous stack frame pointer.
389 0074 ; clear the d-accumulator.
390 0074 4F [ 2] clra
391 0075 5F [ 2] clrb
392 0076 pshd ; allocate & initialize the locals prd0 - prd3
393 0076 37 [ 3] pshb
394 0077 36 [ 3] psha
395 0078 pshd
396 0078 37 [ 3] pshb
397 0079 36 [ 3] psha
398 007A 30 [ 3] tsx ; initialize the new stack frame pointer.
399 007B A609 [ 4] ldaa Fact1+1,x ; get the ls byte of factor 1.
400 007D E60B [ 4] ldab Fact2+1,x ; get the ls byte of factor 2.
401 007F 3D [10] mul ; multiply them.
402 0080 ED02 [ 5] std Prd2,x ; save the first term of the partial product.
403 0082 A608 [ 4] ldaa Fact1,x ; get the ms byte of factor 1.
404 0084 E60B [ 4] ldab Fact2+1,x ; get the ls byte of factor 2.
405 0086 3D [10] mul ; multiply them.
406 0087 E301 [ 6] addd Prd1,x ; add the result into the partial product.
407 0089 ED01 [ 5] std Prd1,x ; save the result.
408 008B A609 [ 4] ldaa Fact1+1,x ; get the ls byte of factor 1.
409 008D E60A [ 4] ldab Fact2,x ; get the ms byte of factor 2.
410 008F 3D [10] mul ; multiply them.
411 0090 E301 [ 6] addd Prd1,x ; add the result into the partial product.
412 0092 ED01 [ 5] std Prd1,x ; save the result.
413 0094 2402 [ 3] bcc Mull6 ; Was there a carry into Prd0?
414 0096 6C00 [ 6] inc Prd0,x ; yes. 'add' it in.
415 0098 A608 [ 4] Mull6 ldaa Fact1,x ; get the ms byte of factor 1.
416 009A E60A [ 4] ldab Fact2,x ; get the ms byte of factor 2.
417 009C 3D [10] mul ; multiply them.
418 009D E300 [ 6] addd Prd0,x ; add it to the partial product.
419 009F ED08 [ 5] std Fact1,x ; overwrite the two parameters with the result.
420 00A1 EC02 [ 5] ldd Prd2,x
421 00A3 ED0A [ 5] std Fact2,x
422 00A5 rtd x,LocSize ; return and deallocate the locals.
423 00A5 C604 [ 2] ldab #LocSize ; number of bytes to deallocate.
424 00A7 3A [ 3] abx ; add it to the current stack frame pointer.
425 00A8 35 [ 3] txs ; deallocate storage by updating the stack pointer.
426 00A9 38 [ 5] pulx ; restore the previous stack frame pointer.
427 00AA 39 [ 5] rts ; return to the calling routine.
428 *
429 * cycles total=170 ; Total number of E cycles for a 16 x 16 multiply.
430 *
431 *

```

```

432 *****
433 *
434 * This subroutine performs a 32 by 16 bit unsigned divide and produces a 32-bit
435 * quotient and a 16-bit remainder. Both the divisor and dividend are passed to
436 * the subroutine on the stack. The 32-bit quotient and 16-bit remainder are
437 * returned on the stack in place of the divisor and dividend. This allows the
438 * calling routine to easily pull the answer from the stack and store the result.
439 * The divisor is pushed onto the stack first, followed by the lower 16-bits of
440 * the dividend and finally the upper 16-bits of the dividend. A typical calling
441 * sequence would be:
442 *
443 * ldd Divisor
444 * pshd
445 * ldd Dividend+2
446 * pshd
447 * ldd Dividend
448 * pshd
449 * jsr Div32x16
450 * puld
451 * std Quotient
452 * puld
453 * std Quotient+2
454 * puld
455 * std Remainder
456 *
457 *
458 *
459 *
460 *
461 * This subroutine has two local variables. A 32-bit variable for partial quotient
462 * results that is treated as two 16-bit variables and a 16-bit variable for
463 * intermediate remainder results.
464 *
465 * NOTE: This routine was written assuming that the previous stack frame pointer
466 * is the x-index register. HOWEVER, because the x-index register is required
467 * by the integer and fractional divide instructions, the y-index register is
468 * used as the stack frame pointer WITHIN the Div32x16 subroutine.
469 *
470 * Declare locals
471 *
472 00AB PCSave set * ; save the current PC value.
473 0000 org 0 ; set PC to 0 for offsets to locals.
474 0000 Quo0 rmb 2 ; declare upper 16-bits of quotient.
475 0002 Quo2 rmb 2 ; declare lower 16-bits of quotient.
476 0004 Rem rmb 2 ; declare remainder.
477 0006 LocSize set * ; number of bytes of local storage.
478 00AB org PCSave
479 *
480 * Offsets to parameters
481 *
482 000A Num0 equ LocSize+4 ; upper 16-bits of Dividend.
483 000C Num2 equ LocSize+6 ; lower 16-bits of Dividend.
484 000E Denm equ LocSize+8 ; 16-bit divisor.
485 *
486 cycles clear
487 *
488 00AB Div32x16 equ *
489 00AB 3C [ 4] pshx ; save the previous stack frame pointer.
490 00AC [ 2] clrd ; clear the d-accumulator.
491 00AC 4F [ 2] clra
492 00AD 5F [ 2] clrb
493 00AE pshd ; allocate & initialize the locals.
494 00AE 37 [ 3] pshb
495 00AF 36 [ 3] psha
496 00B0 pshd
497 00B0 37 [ 3] pshb
498 00B1 36 [ 3] psha
499 00B2 pshd
500 00B2 37 [ 3] pshb
501 00B3 36 [ 3] psha
502 00B4 1830 [ 4] tsy ; initialize y as the new stack frame pointer.
503 00B6 18EC0A [ 6] ldd Num0,y ; load the upper 16-bits of the dividend.
504 00B9 CDA30E [ 7] cpd Denm,y ; is the divisor the upper 16-bits of the dividend?
505 00BC 2507 [ 3] blo Div32x16a ; yes. use a fractional divide on the initial value.

```

```

506 00BE CDEE0E [ 6]      ldx  Denm,y      ; load the divisor into x.
507 00C1 02      [41]     idiv             ; divide the upper 16 bits by the divisor.
508 00C2 CDEF00 [ 6]      stx  Quo0,y      ; save the partial quotient.
509 00C5 CDEE0E [ 6] Div32x16a ldx  Denm,y      ; load the divisor into x.
510 00C8 03      [41]     fddiv            ; resolve the remainder into a 16-bit fractional
                    part.
511 00C9 CDEF02 [ 6]      stx  Quo2,y      ; save the partial result.
512 00CC 18ED04 [ 6]      std  Rem,y       ; save the remainder of the fractional divide (par-
                    tial remainder).
513 00CF 18EC0C [ 6]      ldd  Num2,y      ; get the lower 16-bits of the dividend.
514 00D2 CDEE0E [ 6]      ldx  Denm,y      ; get the denominator again.
515 00D5 02      [41]     idiv             ; resolve the remaining quotient.
516 00D6 18E304 [ 7]      addd  Rem,y       ; add the previous remainder to this remainder.
517 00D9 18ED04 [ 6]      std  Rem,y       ; save the total remainder.
518 00DC 8F      [ 3]      xgdx             ; put the last partial quotient into the d-accumula-
                    tor...
519
520 00DD 18E302 [ 7]      addd  Quo2,y      ; & save the total remainder in x.
                    ; add partial quotient to the lower 16-bits of the
                    ; quotient.
521 00E0 18ED0C [ 6]      std  Num2,y      ; save the result.
522 00E3 18EC00 [ 6]      ldd  Quo0,y      ; get the upper 16-bits of the quotient.
523 00E6 C900   [ 2]      adcb  #0         ; add the possible carry to the lower 8-bits.
524 00E8 8900   [ 2]      adca  #0         ; add the possible carry to the upper 8-bits.
525 00EA 18ED0A [ 6]      std  Num0,y      ; save the result.
526 00ED 8F      [ 3]      xgdx             ; get the total remainder back into d.
527 00EE CDA30E [ 7]      cmpd  Denm,y     ; is the total fractional remainder > the divisor?
528 00F1 2519   [ 3]      blo  Div32x16b    ; no. we're finished.
529 00F3 18A30E [ 7]      subd  Denm,y     ; yes. It will be < than 2*Divisor.
530 00F6 18ED04 [ 6]      std  Rem,y       ; save the final remainder.
531 00F9 18EC0C [ 6]      ldd  Num2,y      ; now we must add 1 to the 32-bit quotient.
532 00FC C30001 [ 4]      addd  #1         ; add 1 to the lower 16-bits.
533 00FF 18ED0C [ 6]      std  Num2,y      ; save the result.
534 0102 18EC0A [ 6]      ldd  Num0,y     ; get the upper 16-bits.
535 0105 C900   [ 2]      adcb  #0         ; add the possible carry to the lower 8-bits.
536 0107 8900   [ 2]      adca  #0         ; add the possible carry to the upper 8-bits.
537 0109 18ED0A [ 6]      std  Num0,y     ; save the result.
538 010C 18EC04 [ 6] Div32x16b ldd  Rem,y       ; get the final remainder.
539 010F 18ED0E [ 6]      std  Denm,y     ; overwrite the divisor.
540 0112 30      [ 3]      tsx              ; need to do this for rtd to work correctly. See
                    NOTE.
541 0113
542 0113 C606   [ 2]      ldab  #LocSize   ; deallocate locals & return.
543 0115 3A     [ 3]      abx              ; number of bytes to deallocate.
544 0116 35     [ 3]      txs              ; add it to the current stack frame pointer.
545 0117 38     [ 5]      pulx             ; deallocate storage by updating the stack pointer.
546 0118 39     [ 5]      rts              ; restore the previous stack frame pointer.
547 *
548 *          cycles total=347 ; Total number of E cycles for a 32 x 16 divide.
549 *

```

Errors: None

Labels: 30

Last Program Address: \$0118

Last Storage Address: \$FFFF

Program Bytes: \$0119 281

Storage Bytes: \$000D 13

Use of the MC68HC68T1 Real-Time Clock with Multiple Time Bases

Prepared by: **Jim Carlson**
Field Applications Engineering

INTRODUCTION

The MC68HC68T1 is an HCMOS peripheral chip containing a real-time clock/calendar and a 32 byte static RAM array. A synchronous, serial, three-wire interface (synchronous peripheral interface) is provided for communication with a microcomputer. The MC68HC68T1 can use a crystal or the 50/60 hertz line frequency as its timebase.

Often applications are line powered during normal operation but must continue to maintain the correct time-of-day when powered down. During normal operation the timebase is provided by the power mains (50 or 60 hertz). When the main power is lost, the 50/60 hertz timebase is lost. At this point, the clock must switch to crystal operation to provide correct time keeping. The MC68HC68T1 is not directly capable of this; additional support must be provided by the host microcomputer. This application note makes use of the Motorola MC68HC11A8P1 as the host MCU, as it provides the necessary Serial Peripheral Interface. MCUs that do not provide this functionality may emulate it in software by manipulating parallel I/O.

The power line is chosen as the preferred timebase because of the accuracy provided by the electric utility companies. A 32.768 kHz crystal, on the other hand, is quite sensitive to temperature variations, and to the voltage on which the oscillator runs. Temperature sensitivity often exceeds 0.035 ppm/°C and aging can exceed 5 ppm/year. This may cause errors exceeding twenty seconds per month, if the crystal is subjected to wide temperature variations, such as encountered in remote, outdoor sites.

Contrast this error with that achieved with the power line as a timebase, which may be under one second per month.

This application note discusses the transition from line to crystal timebase operation during line power failure, and back to line power timebase when line power is reestablished.

CIRCUIT OPERATION

The purpose of this circuit is to demonstrate the use of the MC68HC68T1 in a application using both the power line and a crystal as the timebase, depending on the availability of line power. As such, it is just a skeleton of an actual system. No method to set the clock is provided; just a subroutine called "SET_CLOCK" which always sets the clock to 00:00:00, Friday, January 1, 1990 (24 hour time). An actual application would, of course, prompt the operator for the correct time and date.

The demonstration software puts the MCU into a continuous loop, where the time (hours, minutes, and seconds), maintained in the MC68HC68T1 is read, converted into an ASCII representa-

tion, then transmitted through the asynchronous serial port (SCI) to a display device, such as a CRT terminal. No linefeed character is sent, so the time display remains on one line, and appears to simply increment. Power can be removed from the system, and the clock continues to operate on power supplied by the 3.6 volt NiCad battery. When power is reapplied, the displayed time reflects correct operation.

Please refer to Figure 1 for a schematic diagram of this demonstration system. The MCU is operated with an 8 MHz crystal, for an internal bus frequency of 2 MHz. This allows the standard data rate of 9600 baud to be generated. An MC145407 line driver/receiver is used to level shift the asynchronous serial port to RS-232C levels. This device requires only a single 5 volt supply, as on-chip charge pumps generate the ± 10 volt levels required by RS-232C.

The MC68HC68T1 is configured by the host MCU to operate with the line frequency providing the timebase. This is done by serially writing the value %111110100 to the Clock Control Register in the MC68HC68T1.

		D7	D6	D5	D4	D3	D2	D1	D0
CLOCK CONTROL REGISTER	START		LINE	XTAL	XTAL	50 Hz	CLK	CLK	CLK
	STOP		XTAL	SEL 1	SEL 0	60 Hz	OUT 2	OUT 1	OUT 0

Bit D7 = 1 enables the clock counter chain

Bit D6 = 1 selects the line frequency as the timebase

Bits D5:D4 = 1:1 select the 32.768 kHz crystal option

Bit D3 = 1 selects 60 Hz as the timebase frequency

Bits D2:D1:D0 = 1:0:0 disables the CLK OUT pin

Please note that when the line frequency is selected as the timebase, the MC68HC68T1's power fail detect circuit can not be used, as it is disconnected from the LINE pin. This requires that the POWER SENSE bit (bit D5) in the Interrupt Control Register be set to 0. This is the default value at reset time, and does not need to be initialized by the user.

The battery connected to the MC68HC68T1 supplies power to the oscillator at all times, including when the 5 volt system power is available. In a system which uses only the crystal as the timebase, this minimizes frequency variation, as the oscillator supply voltage is relatively constant. In this system, however, this is not important because of the line frequency timebase. The important item here is that current always flows from the battery to the MC68HC68T1. During line powered operation, this current must be supplied to the battery (by R_{charge} in Figure 1), to prevent its discharge, and to provide a trickle charging current. This requires that only rechargeable batteries be used, such as NiCad types, or one of the new, rechargeable lithium cells.

Since the internal power fail detect circuitry is disabled in the MC68HC68T1, the power supply must provide this information to the system. A Motorola MC34160 Power Supply Supervisory Circuit was chosen for this function. This device provides the voltage regulator function required for the 5 volt power supply. The two

outputs of interest here are the Power Warning and the Reset outputs. The Reset output generates a reset to the MCU when power is applied to the system. The Power Warning output generates an interrupt to the MCU when the unregulated voltage drops below a value considered to be dangerously low, but not yet below the minimum operating voltage of the regulator. Please refer to Figure 2.

An uncommitted voltage comparator is also provided by the MC34160, and is used to convert the 60 Hz sine wave from the power transformer into a logic signal to drive LINE on the MC68HC68T1. Since this comparator provides hysteresis, noise immunity is improved.

When the Power Warning output generates an interrupt to the MCU, the MCU completes the current instruction, or series of instructions in the critical code section, and then enters the interrupt handler code. This code simply reconfigures the MC68HC68T1 to crystal timebase operation by writing the value $\%10110100$ to the Clock Control Register. Please note that only bit D6 was changed, as compared to the value previously written. The interrupt handler routine would then save whatever other information was required by the application into the MC68HC68T1's internal RAM. In this example, however, nothing needs to be saved, so the MCU just enters the STOP state.

The code section that reads the time information from the MC68HC68T1 is considered a critical code section, because it accesses MCU resources which are also accessed from within the interrupt handler. If the interrupt were to be accepted while serial transmissions were in progress, errors would occur, because the interrupt handler would not allow the completion of the transmission, but would simply start its own transmission. This could cause a "Write Collision" in the S.P.I., or erroneous data to be sent to the MC68HC68T1. This is prevented by simply bracketing this critical code with instructions that disable interrupts at the beginning, and enable interrupts at the end.

The value of the power supply filter capacitor (C_{filter} in Figure 1) must be chosen so that T_{save} (Figure 2) is long enough to allow the MCU to completely execute the interrupt handler code before the unregulated voltage drops to a point below which the regulator will not operate. The value of C_{filter} (Figure 1) provides approximately 150 ms of operation after the interrupt is generated.

The MC34164P Low Voltage Detect Circuit guarantees that the MCU is held in the reset state when the supply voltage drops below the level that allows the MC34160 to function correctly. This prevents possible program run away, which may modify registers in the MC68HC68T1.

The program source code listing is included. This software was written for Motorola's MC68HC11 cross macro assembler.

REFERENCES

M68HC11 Reference Manual, M68HC11RM/AD Revision 1, Motorola, 1990.

Linear and Interface Integrated Circuits, DL128 Revision 3, Motorola, 1990.

M68HC11EV B Evaluation Board User's Manual, M68HC11EV B/D1, Motorola, 1986.

M6800 Family Macro Assemblers Reference Manual, M68XRASM/D2, Motorola, 1985.

"CX-1V Crystal Data Sheet," Statek Corporation, 12/87.

"MC68HC68T1 Advance Information Data Sheet," MC68HC68T1/D, Motorola, 1988.

"Linear/Switchmode Voltage Regulator Handbook," HB206 Revision 3, Motorola, 1989.

NOTE: After November, 1990 please refer to the updated MC68HC68T1 data sheet in *Application-Specific Standard ICs* data book, DL130 Revision 1, Motorola, 1990.

This software demonstrates the use of the MC68HC68T1 Real Time Clock in a system that normally derives its timebase from the ac line, but during a power failure, switches to a 32.768 kHz crystal.

This software runs on an MC68HC11A8P1 microcomputer. This MCU contains a debug monitor called "BUFFALO".

This software is stored in the MCU's on-chip EEPROM.

The purpose of this program is to output the time-of-day to a CRT terminal connected to the MCUs SCI (UART) port. When a power failure occurs, the MCU is interrupted with an IRQ, then it switches the real time clock to crystal control. When power is reapplied, the MCU switches the real time clock back to line power timebase.

MC68HC68T1 Register Addresses:

Seconds	EQU	\$20
Minutes	EQU	\$21
Hours	EQU	\$22
Day	EQU	\$23
Date	EQU	\$24
Month	EQU	\$25
Year	EQU	\$26
Sec_Alarm	EQU	\$28
Min_Alarm	EQU	\$29
Hrs_Alarm	EQU	\$2A
Status	EQU	\$30
First_Up	EQU	%00010000
Control	EQU	\$31
Start	EQU	%10000000
Line	EQU	%01000000
XTL_SEL1	EQU	%00100000
XTL_SEL0	EQU	%00010000
Line_50	EQU	%00001000
Interrupt	EQU	\$32
Write	EQU	%10000000

Seconds Register
 Minutes Register
 Hours Register
 Day of Week Register
 Date of Month Register
 Month Register
 Year Register
 Seconds Alarm
 Minutes Alarm
 Hours Alarm
 Status Register
 First-Time-Up Status Bit
 Clock Control Register
 Start = 1 Enables the Divider Chain
 Line = 1 Crystal = 0
 XTL_SEL1:XTL_SEL0 = %11 for 32.768 kHz

Clear for 60 Hz Timebase
 Interrupt Control Register
 Write Enable Bit In Address Word

MC68HC11A8P1 Equates:

REGBASE	EQU	\$1000
IRQ_VEC	EQU	\$EE
SPSR	EQU	\$29
SPIF	EQU	%10000000
SPCR	EQU	\$28
SPDR	EQU	\$2A
BAUD	EQU	\$2B
SCCR1	EQU	\$2C
SCCR2	EQU	\$2D
TE	EQU	%00001000
SCSR	EQU	\$2E
TDRE	EQU	%10000000
SCDR	EQU	\$2F
PORTD	EQU	\$8
SS	EQU	%00100000
DDRD	EQU	\$9

Base Address of I/O Register Set
 Indirect Vector for IRQ (3 Bytes)
 SPI Status Register (Offset from "REGBASE")
 Transfer Complete Flag
 SPI Control Register
 SPI Data Register
 SCI Baud Rate Control
 SCI Control Register #1
 SCI Control Register #2
 Transmitter Enable
 SCI Status Register
 Transmit Data Register Empty
 SCI Data Register
 Port D Data Register
 Slave Select Output to MC68HC68T1
 Port D Data Direction Register

Misc. Equates:

CR	EQU	13
JMP	EQU	\$7E
TOS	EQU	\$35
BSCT		

ASCII Carriage Return
 "JMP \$HHHH" OPcode
 Top of Available RAM (BUFFALO Compatible)

Scratch Pad Storage Locations:

Time	RMB	3
String	RMB	8
	ORG	\$B600
Reset	EQU	*
	LDS	#TOS
	LDX	#REGBASE

Store Time in B.C.D. Here
 Build Time String Here in ASCII
 Beginning of EEPROM

Init Stack Pointer to Top of RAM
 Point to Base of I/O Registers

Initialize the SCI Port

LDAA	#%00110000
STAA	BAUD,X
BSET	SCCR2,X,#TE

9600 Baud (Crystal = 8.0 MHz)
 Enable SCI's Transmitter

Initialize the SPI Port

BSET	DDRD,X,#%00111000	Configure SS, SCK, and MOSI into Outputs
LDAA	##%01010100	Enable, SPI, Bit Rate = 1 MHz, ...
STAA	SPCR,X	... CPOL = 0, CPHA = 1

Initialize the Indirect Vector Table. The MC68HC11A8P1's interrupt vectors are in ROM, and can't be modified. They point to an indirect vector table placed in RAM, so users can set vector addresses at will.

LDAA	#JMP	"JMP \$HHHH" Opcode
STAA	IRQ_VEC	Store a "JMP IRQ" in the Table
LDD	\$IRQ	Address of IRQ Interrupt Handler
STD	IRQ_VEC+1	

The MC68HC68T1 is always reinitialized, because it needs to be set to line timebase. If the first-time-up bit is set in its status register, it must also have its time and date set.

BSET	PORTD,X,#SS	Enable the MC68HC68T1
LDAA	#STATUS	Read the Status Register
JSR	SQUIRT	Send Address to MC68HC68T1
JSR	SQUIRT	Dummy Access to get Returned Value
BCLR	PORTD,X,#SS	Disable the MC68HC68T1
TAB		Save the Status to Check for First-Time-Up
BSET	PORTD,X,#SS	Enable the MC68HC68T1
LDAA	#CONTROL!+WRITE	Write the Control Register
JSR	SQUIRT	Send Address

The MC68HC68T1 is set up for line timebase, but the crystal oscillator is set for 32.768 kHz operation, so it will properly run.

LDAA	#START!+LINE!+XTL_SEL1!+XTL_SELO	Select Line Operation
JSR	SQUIRT	Control Register Value
BCLR	PORTD,X,#SS	Disable the MC68HC68T1

If this was the first time up for the MC68HC68T1, then set the time.

BITB	#FIRST_UP	Check First-Time-Up Bit
BEQ	SKIP_SET	Do Not Set the Time in the MC68HC68T1, if Clear
BSR	SET_CLK	Set the Clock's Time Here

Enable the stop instruction.

Skip_Set	TPA	Get Condition Code Register Value
	ANDA	##%01111111
	TAP	Clear the Stop Mask

Main	EQU	
	SEI	Enter Critical Code Section
	BSET	PORTD,X,#SS
	LDAA	#SECONDS
	JSR	SQUIRT
	JSR	SQUIRT
	STAA	TIME+2
	JSR	SQUIRT
	STAA	TIME+1
	BSR	SQUIRT
	STAA	TIME
	BCLR	PORTD,X,#SS
	CLI	End of Critical Code Section

Convert B.C.D. time to ASCII.

LDAA	TIME	Get Hours
BSR	BCD_ASCII	Convert to ASCII
STD	STRING	Build String
LDAA	TIME+1	Get Minutes
BSR	BCD_ASCII	Convert to ASCII
STD	STRING+3	Save Room for Colon
LDAA	TIME+2	Get Seconds
BSR	BCD_ASCII	Convert to ASCII
STD	STRING+6	Save Room for Colon
LDAA	#:	Now Store the Colons
STAA	STRING+2	
STAA	STRING+5	

Now send the time string to the SCI port.

LDAB	#8	8 Bytes to Send
LDY	#STRING	Point to the String
SEND_CHR	LDAA 0,Y	Get the Character
	BSR OUTCH	Output the Character
	INY	Bump the Pointer
	DECB	Decrement Loop Counter
	BNE SEND_CHR	Continue Until Loop Counter Exhausted
	LDAA #CR	Now Send a Carriage Return
	BSR OUTCH	
	BRA MAIN	Continue in Main Loop

A power failure causes an IRQ to be generated by the MC34160 power supply chip. Enter here for power failure.

IRQ	EQU *	Interrupt Handler
	LDX #REGBASE	Point to I/O Space
	BSET PORTD,X,#SS	Enable the MC68HC68T1
	LDAA #CONTROL!+WRITE	Write the Control Register
	BSR SQUIRT	
	LDAA #START!+XTL_SEL1!+XTL_SEL0	Select Crystal Operation
	BSR SQUIRT	
	BCLR PORTD,X,#SS	Disable the MC68HC68T1
	STOP	Just go to Sleep
	PAGE	

Utility Subroutines:

This subroutine converts the packed BCD number in A into ASCII numbers in Register D.

BCD_ASCII	EQU *	
	PSHA	Save Argument
	ANDA #%00001111	Mask all but L.S.D.
	ORAA #0	Convert to ASCII
	TAB	Place in Low Half of Reg D
	PULA	Restore Argument
	LSRA	Align M.S.D.
	LSRA	
	LSRA	
	ORAA #0	Convert to ASCII
	RTS	Return with 2 ASCII Chars in Reg D

This is only a dummy time setting routine. The clock is set to 00.00:00 January 1, 1990, Friday (24 hour time). A real application would substitute an actual time setting routine here.

SET_CLK	EQU *	
	BSET PORTD,X,#SS	Enable the MC68HC68T1
	LDAA #SECONDS!+WRITE	Write to the Seconds Register First
	BSR SQUIRT	
	CLRA	Seconds = 0
	BSR SQUIRT	
	CLRA	Minutes = 0
	BSR SQUIRT	
	CLRA	Hours = 0
	BSR SQUIRT	
	LDAA #6	Day = Friday
	BSR SQUIRT	
	LDAA #1	Date = 1
	BSR SQUIRT	
	LDAA #1	Month = January
	BSR SQUIRT	
	LDAA #\$90	Year = 1990
	BSR SQUIRT	
	LDAA PORTD,X,#SS	Disable the MC68HC68T1
	RTS	Return to Caller

Output the contents of Register A to the SPI Port. Return with A containing the character received from the MC68HC68T1. No other registers modified upon return. The slave select must be handled by the calling program.

```

SQUIRT    EQU
          PSHX
          LDX    #REGBASE
          STAA   SPDR,X
WAITSPIF  BRCLR  SPDR,X,#SPIF,WAITSPIF
          LDAA   SPDR,X
          PULX
          RTS

```

Save Working Register
Point to I/O Registers
Send Byte
Loop Until Transmission Complete
Read Returned Data to Clear SPIF
Restore Working Register
Return to Calling Program

This subroutine transmits the contents of Register A to the SCI Port. All registers preserved upon return.

```

OUTCH     EQU
          PSHX
          LDX    #REGBASE
WAITTDRE  BRCLR  SCSR,X,#TDRE,WAITTDRE
          STAA   SCDR,X
          PULX
          RTS
          END

```

Save Working Register
Point to I/O Space
Wait for Empty Transmit Buffer
Transmit Character
Restore Working Register
Return to Calling Program

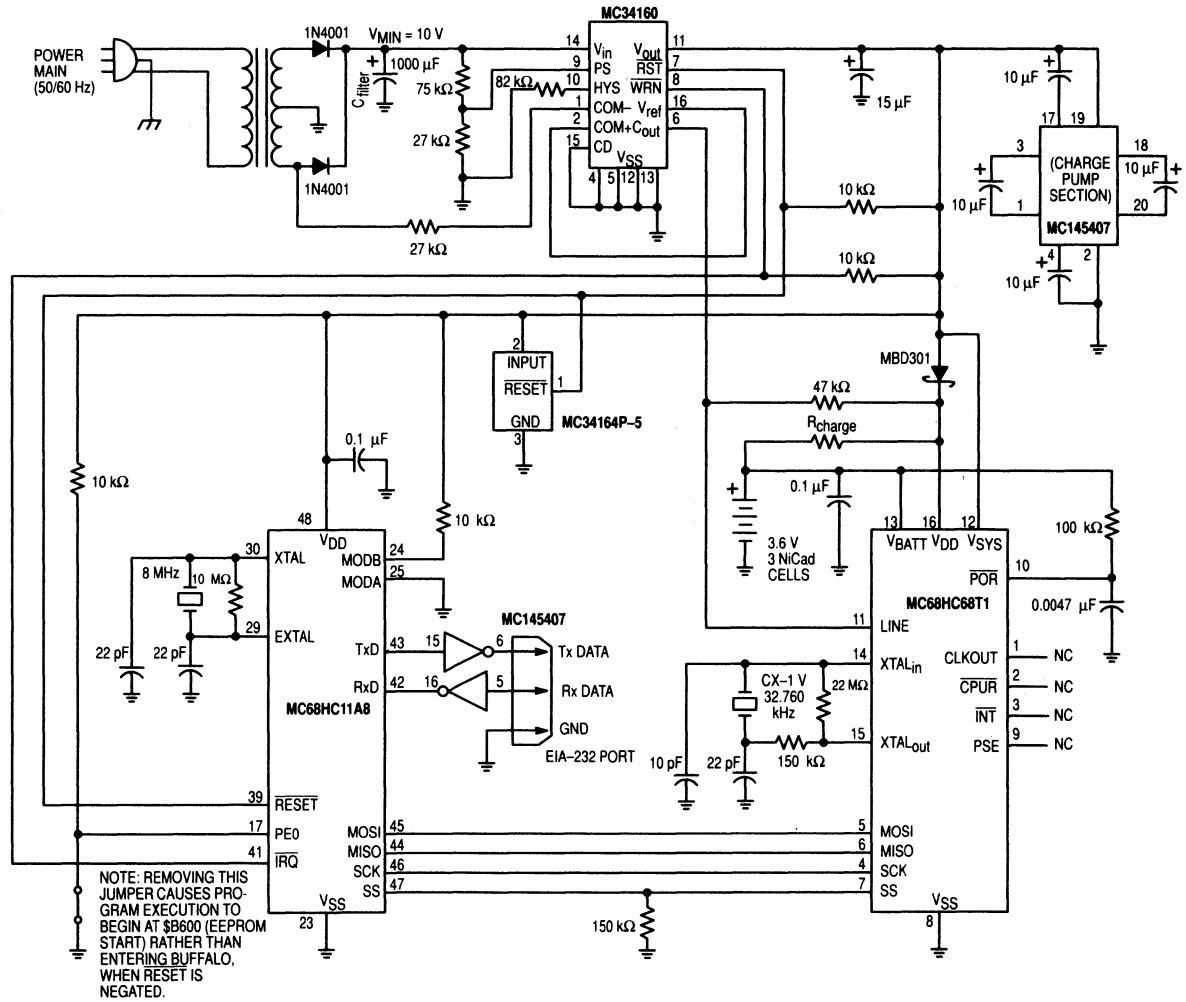


Figure 1. Schematic Diagram

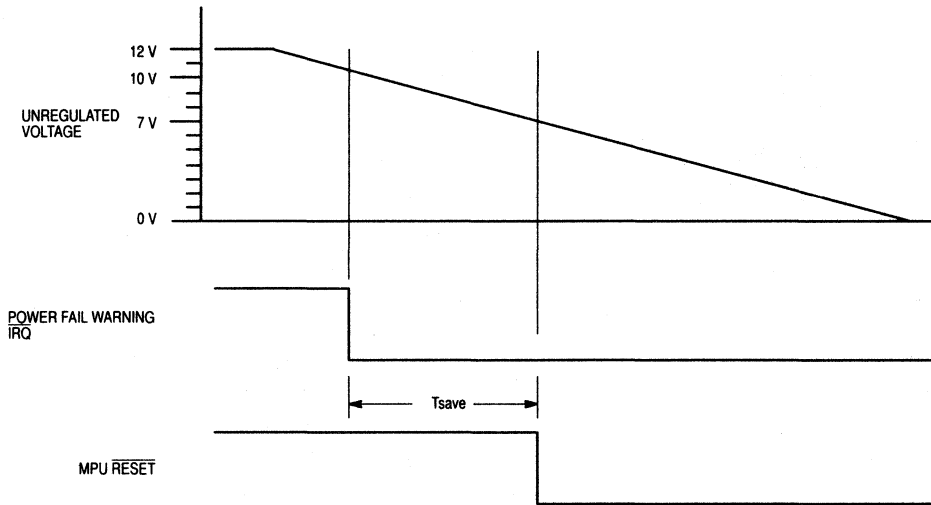


Figure 2. Power Fail Timing

Interfacing the MC68HC05C5 SIOP to an I²C Peripheral

By Naji Naufel

INTRODUCTION

When designing a system based on a standard, non-custom-designed, microcontroller unit (MCU), the user is faced with the problem of not having all the desired peripheral functions on-chip. This problem can be solved by interfacing the MCU to an off-chip set of peripherals. An ideal interface is a synchronous serial communication port. Unfortunately, these peripherals may not have a serial interface that is compatible with the Motorola simple synchronous serial I/O port (SIOP).

This document demonstrates how the SIOP on the MC68HC05C5 can be interfaced to an I²C peripheral, the PCF8573 clock/timer. The MC68HC05C5 was chosen because its SIOP has a programmable clock polarity.

The serial peripheral interface (SPI) on the MC68HC05C4 cannot be used in the interface because the SPI pins cannot be used as output pins when the SPI is off.

SIOP DEFINITION

The SIOP (see Figure 1) is a three-wire master/slave system including serial clock (SCK), serial data input (SDI), and serial data output (SDO). A programmable option determines whether the SIOP handles data most significant bit (MSB) or least significant bit (LSB) first.

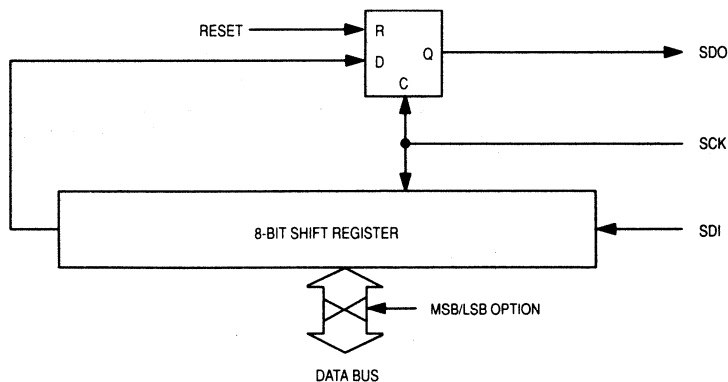


Figure 1. SIOP Block Diagram

SIOP SIGNAL FORMAT

The SCK, SDO, and SDI signals are discussed in the following paragraphs.

SCK

The state of SCK between transmissions can be either a logic one or a logic zero. The first falling edge of SCK signals the beginning of a transmission. At this time, the first bit of received data is presented to the SDI pin, and the first bit of transmitted data is presented at the SDO pin (see Figure 2). When CPOL = 0, the first falling edge occurs internal to the SIOP. Data is captured at the SDI pin on the rising edge of SCK. Subsequent falling edges shift the data and accept or present the next bit. When CPOL = 1, transmission ends at the eighth rising edge of SCK.

Format is the same for master mode and slave mode except that SCK is an internally generated output in master mode and an input in slave mode. The master mode transmission frequency is fixed at E/4.

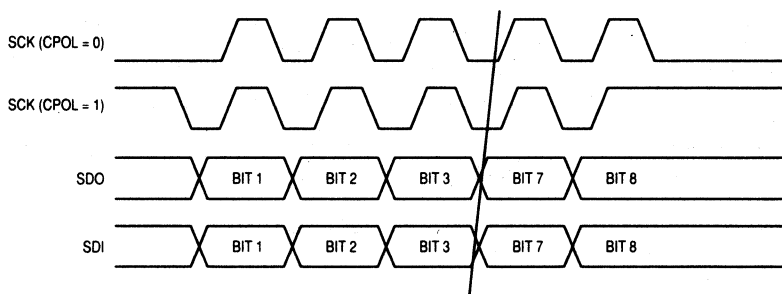


Figure 2. SIOP Timing

SDO

The SDO pin becomes an output when the SIOP is enabled. The state of SDO always reflects the value of the first bit received on the previous transmission, if a transmission occurred. Prior to enabling the SIOP, PB5 can be initialized to determine the beginning state, if necessary. While the SIOP is enabled, PB5 cannot be used as a standard output since that pin is coupled to the last stage of the serial shift register. On the first falling edge of SCK, the first data bit to be shifted out is presented to the output pin.

SDI

The SDI pin becomes an input when the SIOP is enabled. New data may be presented to the SDI pin on the falling edge of SCK. Valid data must be present at least 100 ns before the rising edge of the clock and remain valid 100 ns after that edge.

SIOP REGISTERS

The SIOP contains the following registers: SCR, SSR, and SDR.

SIOP Control Register (SCR)

This register, located at address \$000A, contains three bits (see Figure 3).

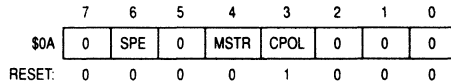


Figure 3. SIOP Control Register

SPE — Serial Peripheral Enable

When set, this bit enables the SIOP and initializes the port B data direction register (DDR) such that PB5 (SDO) is output, PB6 (SDI) is input, and PB7 (SCK) is an input in slave mode and an output in master mode. The port B DDR can be subsequently altered as the application requires, and the port B data register (except for PB5) can be manipulated as usual; however, these actions could affect the transmitted or received data. When SPE is cleared, port B reverts to standard parallel I/O without affecting the port B data register or DDR. SPE is readable and writable any time, but clearing SPE while a transmission is in progress will abort the transmission, reset the bit counter, and return port B to its normal I/O function. Reset clears this bit.

MSTR — Master Mode

When set, this bit configures the SIOP for master mode, which means that the transmission is initiated by a write to the data register and SCK becomes an output, providing a synchronous data clock at a fixed rate of the bus clock divided by 4. While the device is in master mode, SDO and SDI do not change function; these pins behave exactly as they would in slave mode. Reset clears this bit and configures the SIOP for slave operation. MSTR may be set at any time regardless of the state of SPE. Clearing MSTR will abort any transmission in progress.

CPOL — Clock Polarity

When CPOL is set, SCK idles high, and the first data bit is seen after the first falling edge. When CPOL is cleared, the SCK idles low, and the first data bit is seen after the first falling edge, which occurs internally (see Figure 2).

SIOP Status Register (SSR)

Located at address \$000B, the SSR contains only two bits (see Figure 4).

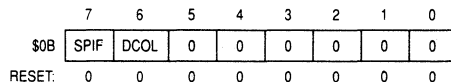


Figure 4. SIOP Status Register

SPIF — Serial Peripheral Interface Flag

This bit is set on the last rising clock edge, indicating that a data transfer has occurred. SPIF has no effect on further transmissions and can be ignored without problem. SPIF is cleared by reading the SSR with SPIF set, followed by a read or write of the SDR. If it is cleared before the last edge of the next byte, it will be set again. Reset also clears this bit.

DCOL — Data Collision

DCOL is a read-only status bit that indicates an invalid access to the data register has been made. A read or write of SDR during a transmission results in invalid data being transmitted or received. DCOL is cleared by reading the SSR with SPIF set, followed by a read or write of the SDR. If the last part of the clearing sequence is done after another transmission has been started, DCOL will be set again. If DCOL is set and SPIF is not set, clearing DCOL requires turning the SIOP off, then turning it back on using the SPE bit in the SCR. Reset also clears this bit.

SIOP Data Register (SDR)

Located at address \$000C, SDR is both the transmit and receive data register (see Figure 5). This system is not double buffered; thus, any write to this register destroys the previous contents. The SDR can be read at any time, but if a transmission is in progress, the results may be ambiguous. Writes to the SDR while a transmission is in progress can cause invalid data to be transmitted and/or received. This register can be read and written only when the SIOP is enabled (SPE = 1).

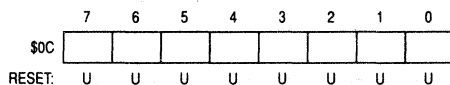


Figure 5. SIOP Data Register

I²C DEFINITION

The inter IC (I²C) is a two-wire half-duplex serial interface with data transmitted/received MSB first. The two wires are a serial data line (SDA) and a serial clock line (SCL).

The protocol consists of a start condition, slave address, n bytes of data, and a stop condition (see Figure 6). Each byte is followed by an acknowledge bit. A start condition is defined as a high-to-low transition on SDA while SCL is high; a stop condition is defined as a low-to-high transition on SDA while SCL is high (see Figure 9). An acknowledge is a low logic level sent by the addressed receiving device during the ninth clock period. A master receiver signals the end of data by not generating an acknowledge after the last byte has left the slave device.

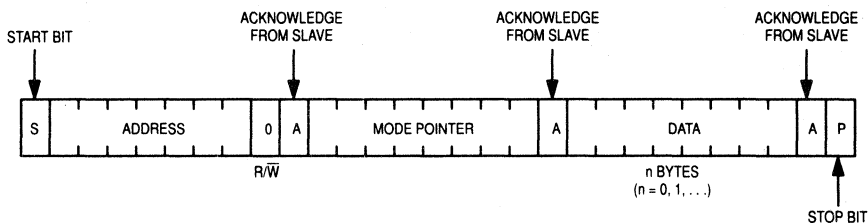


Figure 6. PCF8573 Serial Data Format

INTERFACING THE SIOP TO THE PCF8573

The PCF8573 has an address of 11010 A1 A0, where A1 and A0 give the device a one-of-four address assigned by two hardware pins. Bit 0 of the address byte is the read/write indicator (see Figure 7).

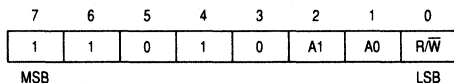


Figure 7. Address Byte

The byte following the address byte is the mode pointer used to control register access inside the PCF8573. Subsequent bytes following the mode pointer contain data read from or written to the clock/timer. Clock data is in binary-coded decimal format with two digits per byte.

HARDWARE DESCRIPTION

The SIOP is used as master by setting the MSTR bit in the SPCR. PB7/SCK is connected to SCL. Since the PCF8573 has a bidirectional data line (SDA) and the SIOP has separate input and output pins, the SDO and the SDI pins need to be connected. A resistor must be used for this connection because port B is not open-drain (see Figure 8). The SEC pin, which goes high every second, is connected to PA7, which is polled by software to keep a seconds count.

When receiving data from the clock timer, an \$FF is transmitted by the SIOP, which makes the resistor (R3), in series with the SDO pin, look like a pullup to V_{DD} ; therefore, SDO will not interfere with data coming from the SDA pin.

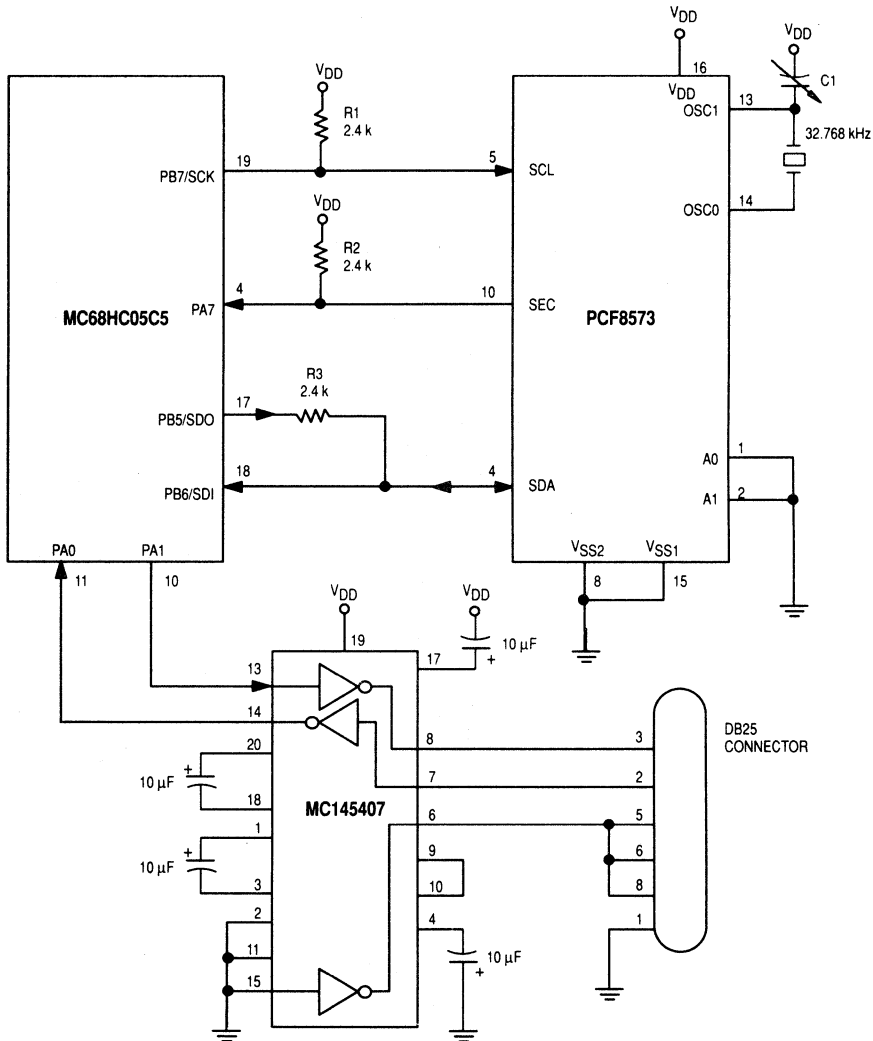


Figure 8. MC68HC05C5 Connection to PCF8573

SOFTWARE DESCRIPTION

To generate the timing required by the I²C, the user has to manipulate the port B pins as I/O and SIOP pins (see Figure 9). Before any data transactions, PB5 and PB7 are initialized high. While the SIOP is off (SPE = 0), PB5 is cleared to zero while PB7 is still high, creating a start condition. The SIOP is then enabled with CPOL = 0 and MSTR = 1 and a byte is transmitted. After transmission is complete, the SIOP is turned off (SPE = 0), and PB7 is toggled high, then low, to generate the acknowledge clock. If the MCU is sending data, PB5 is forced high during the acknowledge pulse; otherwise, it is forced low to let the slave know that the byte has been received. If needed, the stop condition is accomplished by clearing PB5, setting PB7, then setting PB5.

To satisfy the 100-kHz serial clock maximum rating of the PCF8573, the MC68HC05C5 must be slowed to run at a bus speed of 250 kHz, which gives a serial clock rate of 62.5 kHz.

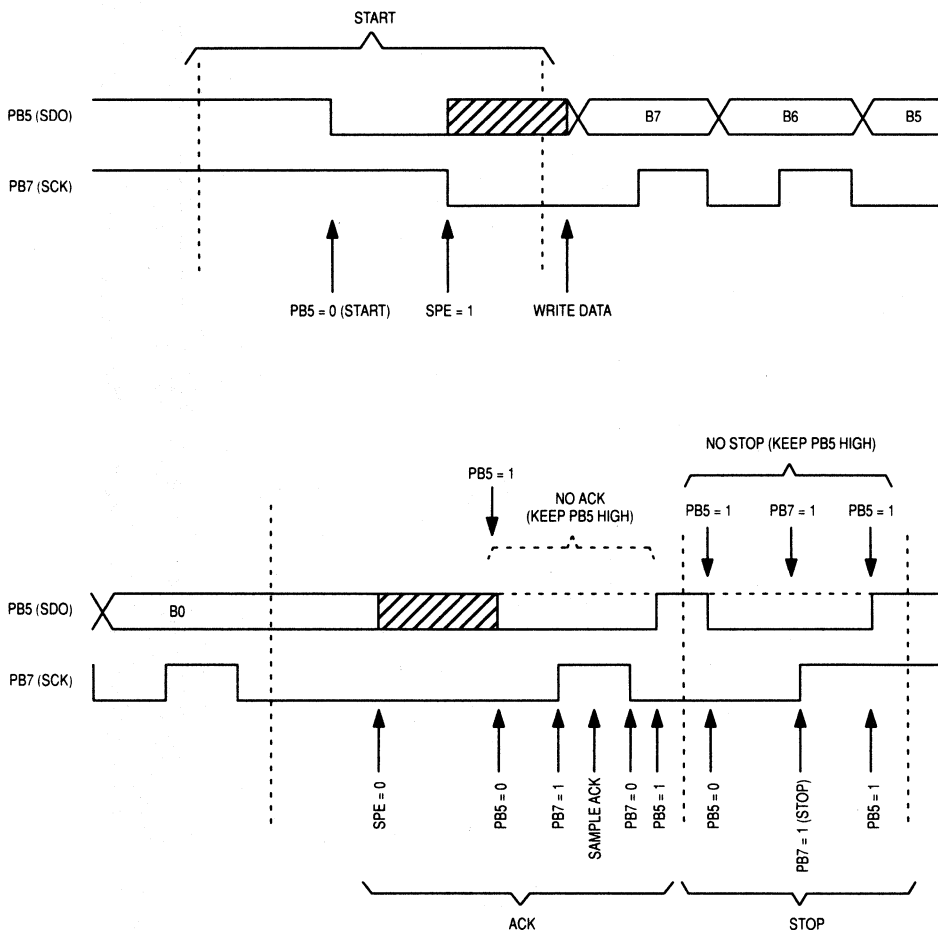


Figure 9. SIOP-Generated Timing

SOFTWARE APPLICATION

In demonstrating how the SIOP is interfaced to an I²C peripheral, the author developed a complex application having time and calendar functions.

This application interfaces serially with a terminal to allow the user to initialize the PCF8573 with the time and date (see Figure 8). After the software prompts the user to enter the date (month, day, hour, and minutes), it starts displaying the information every second (see Figure 10). Every second the SEC pin goes high, telling the software to read the PCF8573 data and display it along with the software-kept seconds.

To initialize clock data, use the following sequence:

- Send \$D0 with start bit (ADDRESS)
- Send \$00 without start bit (CONTROL)
- Send hours data without start bit
- Send minutes data without start bit
- Send day data without start bit
- Send month data without start bit
- Generate stop bit

To read clock data, use the following sequence:

- Send \$D0 with start bit (ADDRESS)
- Send \$00 without start bit (CONTROL)
- Set up for low acknowledge bit transmit
- Send \$D1 with start bit (ADDRESS)
- Send \$FF without start bit to receive hours
- Send \$FF without start bit to receive minutes
- Send \$FF without start bit to receive day
- Send \$FF without start bit to receive month
- Generate stop bit

Since the MC68HC05C5 does not have a universal asynchronous receiver transmitter (UART), the interface to the terminal was implemented in software. See subroutines INCHAR and OUTCHAR in **APPENDIX A PROGRAM LISTING**.

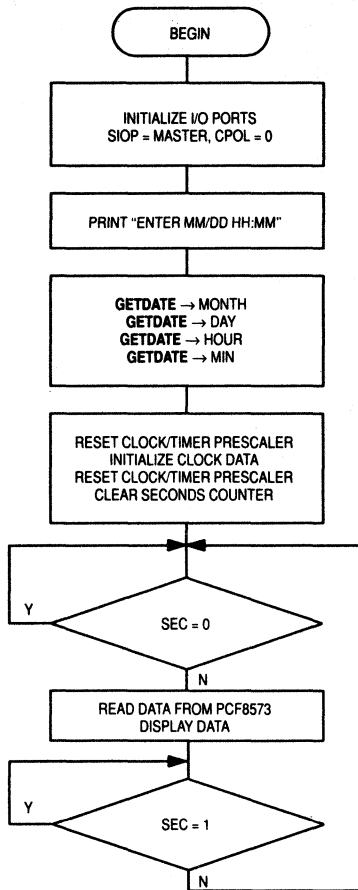


Figure 10. Program Flowchart

APPENDIX A. PROGRAM LISTING

```

0001          *****
0002          *
0003          * This program is written to demonstrate interfacing the MOTROLA
0004          * Simple Serial I/O (SIOP) bus to the SIGNETICS IIC bus.
0005          * The 2 devices used are the MC68HC05C5 MCU and the PCF8573 clock/timer.
0006          * Bus speed is 250 Khz.
0007          * The MCU is used as a master and the clock/timer is used as a
0008          * slave. Some software intervention has to be done so that the
0009          * SIOP meets all IIC specifications.
0010          * The MCU displays clock data on a terminal screen at 2400 baud
0011          *
0012          *       Written by :  Najj Naufel
0013          *                   CSIC MCU Design
0014          *                   Austin, Texas
0015          *
0016          *****
0017
0018 0000          porta      equ    $00          ;port a data register
0019 0001          portb     equ    $01          ;port b data register
0020 0002          portc     equ    $02          ;port c data register
0021 0004          ddra      equ    $04          ;port a data direction register
0022 0005          ddrb      equ    $05          ;port b data direction register
0023 0006          ddrc      equ    $06          ;port c data direction register
0024 000a          spcr      equ    $0a          ;spi control register
0025 000b          spsr      equ    $0b          ;spi status register
0026 000c          spdr      equ    $0c          ;spi data register
0027          *
0028 00d1          raddr     equ    $d1          ;peripheral address for read
0029 00d0          waddr     equ    $d0          ;peripheral address for write
0030
0031 0080          ram       equ    $80          ;start of ram space
0032          *
0033 0080          org       ram
0034 0080          sec       rmb    1          ;seconds byte
0035 0081          control   rmb    1          ;control byte
0036 0082          ack       rmb    1          ;acknowledge polarity
0037 0083          hour      rmb    1
0038 0084          min       rmb    1
0039 0085          month     rmb    1
0040 0086          day       rmb    1
0041 0087          savx      rmb    1
0042 0088          sava      rmb    1
0043 0089          xtemp     rmb    1
0044 008a          count    rmb    1
0045 008b          InChar    rmb    1
0046 008c          OutChar   rmb    1
0047 008d          atemp     rmb    1
0048 008e          cdelay    rmb    1          ;delay variable for serial routines

```



```

0050
0051
0052 *****
0053 * start of program
0054
0055 0200          org      $0200
0056 * all timing is based on a 500 Khz crystal
0057 *
0058
0059 0200          begin    equ      *
0060 0200 a6 02          lda      #00000010
0061 0202 b7 00          sta      porta      ;TX pin high
0062 0204 b7 04          sta      ddra      ;PAL=TX pin, PA0=RX pin
0063 0206 a6 a0          lda      #10100000 ;pb7=pb5=output, pb6=input
0064 0208 b7 01          sta      portb
0065 020a b7 05          sta      ddrb
0066 020c ae 03          ldx      #3
0067 020e d6 04 07       lda      delays,x
0068 0211 b7 8e          sta      cdelay      ;2400 baud
0069 0213 a6 10          lda      #00010000 ;mstr=1, cpol=0, siop still off
0070 0215 b7 0a          sta      spcr
0071 0217 b7 82          sta      ack          ;ack flag non-zero, high acknowledge
0072
0073 0219 cd 03 85          jsr      crlf
0074 021c cd 03 76          jsr      outmsg      ;print "ENTER MM/DD HH:MM"
0075 021f cd 02 85          jsr      getdate     ;get month
0076 0222 b7 85          sta      month
0077 0224 cd 03 90          jsr      inchar      ; dummy char '/'
0078 0227 cd 02 85          jsr      getdate     ;get day
0079 022a b7 86          sta      day
0080 022c cd 03 90          jsr      inchar      ; dummy space
0081 022f cd 02 85          jsr      getdate     ;get hours
0082 0232 b7 83          sta      hour
0083 0234 cd 03 90          jsr      inchar      ;dummy ':'
0084 0237 cd 02 85          jsr      getdate     ;get minutes
0085 023a b7 84          sta      min
0086 023c cd 03 90          again   jsr      inchar      ;wait for <CR>
0087 023f a1 0d          cmp      #0d
0088 0241 26 f9          bne      again
0089 0243 cd 03 85          jsr      crlf
0090
0091 *
0092 * issue a reset prescaler command
0093 *
0093 0246 a6 20          lda      #$20
0094 0248 b7 81          sta      control
0095 024a cd 02 d2          jsr      addrctl
0096 024d cd 02 cb          jsr      stop
0097
0098 * initialize the clock
0099 *
0100 0250 a6 00          lda      #$00
0101 0252 b7 81          sta      control
0102 0254 cd 02 d2          jsr      addrctl     ;send address/control bytes
0103 0257 cd 02 db          jsr      senddta     ;send 4 data bytes
0104
0105 * issue a reset prescaler command
0106 *
0107 025a a6 20          lda      #$20
0108 025c b7 81          sta      control
0109 025e cd 02 d2          jsr      addrctl
0110 0261 cd 02 cb          jsr      stop
0111
0112 0264 3f 80          clr      sec          ;clear seconds counter
0113 0266 0f 00 fd          sec_pin brclr    7,porta,* ;wait for SEC pin to go high
0114 0269 cd 03 31          jsr      dispdata    ;display clock data
0115 026c 0e 00 fd          brset   7,porta,* ;wait until pin goes low
0116 026f 20 f5          bra      sec_pin
0117
0118 0271 45 4e 54 45     msg      fcc      "ENTER MM/DD HH:MM"
0119 0282 0d 0a 04          fcb      $0d,$0a,$04
0120

```

iicc5

page 3

0122

0123

0124

0125

0126

0127

0128 0285

0129 0285 cd 03 90

0130 0288 a0 30

0131 028a 48

0132 028b 48

0133 028c 48

0134 028d 48

0135 028e b7 88

0136 0290 cd 03 90

0137 0293 a0 30

0138 0295 bb 88

0139 0297 81

0140

0141

0142

0143

0144

0145

0146 0298

0147 0298 5f

0148 0299 a0 0a

0149 029b 2b 03

0150 029d 5c

0151 029e 20 f9

0152 02a0 ab 0a

0153 02a2 58

0154 02a3 58

0155 02a4 58

0156 02a5 58

0157 02a6 bf 88

0158 02a8 bb 88

0159 02aa ad 42

0160 02ac 81

```
*****
* This routine reads 2 ASCII characters and converts them into
* 2 BCD digits in Acc. A.
*****
```

```
getdate equ *
        jsr inchar      ;get character
        sub #30         ;convert to binary
        lsla
        lsla
        lsla
        lsla           ;make it upper nibble
        sta sava
        jsr inchar      ;get second ASCII char.
        sub #30
        add sava        ;2 BCD digit is in Acc. A now
        rts
```

```
*****
* Convert a binary byte in Acc. A into a 2 digit BCD number
* in Acc. A and display it as 2 ASCII chars.
*****
```

```
bin_dec equ *
        clrX           ;clear number of subtraction counter
sub_more sub #10       ;see how many times it is divisible
        bmi no_tens   ;by 10
        incX           ;increment counter
        bra sub_more  ;subtract more tens
no_tens add #10        ;restore number to positive
        lslX           ;put 10's digit in upper nibble of X
        lslX
        lslX
        lslX
        stx sava
        add sava       ;merge both nibbles in Acc. A
        bsr bcd        ;display the 2 digits
        rts
```

iicc5

page 4

```
0162
0163 *****
0164 *
0165 * this subroutine transfers a byte from the hc05's spi to the iic
0166 * peripheral. data is in reg X upon entry.
0167 * w_start is the entry point for sending a start bit.
0168 * start is the entry point for transferring data without a start condition.
0169 *
0170 *****
0171 *
0172 02ad w_start equ *
0173 02ad 1e 01 bset 7,portb ;take SCL line high
0174 02af 1b 01 bclr 5,portb ;start condition
0175 02b1 nostart equ *
0176 02b1 1c 0a bset 6,spcr ;enable spi, SPE=1
0177 02b3 bf 0c stx spdr ;send data
0178 02b5 0f 0b fd wait brclr 7,spcr,wait ;wait for end of transmission
0179 *
0180 02b8 1d 0a bclr 6,spcr ;clear SPE, disable spi
0181 *
0182 02ba 3d 82 tst ack ;test acknowledge flag
0183 02bc 26 04 bna hi_ack ;keep ack high
0184 02be 1b 01 lo_ack bclr 5,portb ;else, clear ack bit
0185 02c0 20 02 bra hi_ack1 ;generate ack clock
0186 *
0187 02c2 1a 01 hi_ack bset 5,portb ;send high ACK bit
0188 02c4 1e 01 hi_ack1 bset 7,portb ;take pb7 (SCL) high
0189 02c6 1f 01 bclr 7,portb
0190 02c8 1a 01 bset 5,portb ;return data pin high
0191 02ca 81 rts
```

```

iicc5
0193
0194
0195
0196
0197
0198
0199
0200
0201
0202 02cb          stop      equ      *
0203 02cb 1b 01    bclr     5,portb    ;bring sda low
0204 02cd 1e 01    bset     7,portb    ;bring scl high
0205 02cf 1a 01    bset     5,portb    ;bring sda high
0206 02d1 81          rts
0207
0208
0209
0210
0211
0212
0213
0214
0215
0216
0217
0218
0219 02d2          addrctl  equ      *
0220 02d2 ae d0    ldx      #waddr     ;(r/w=0)
0221 02d4 ad d7    bsr      w_start    ;send address with start condition
0222 02d6 be 81    ldx      control    ;send control byte without start
0223 02d8 ad d7    bsr      nostart
0224 02da 81          rts
0225
0226
0227
0228
0229
0230
0231
0232
0233
0234 02db          senddta  equ      *
0235 02db be 83    ldx      hour
0236 02dd ad d2    bsr      nostart    ;send hours
0237 02df be 84    ldx      min
0238 02e1 ad ce    bsr      nostart    ;send minutes
0239 02e3 be 86    ldx      day
0240 02e5 ad ca    bsr      nostart    ;send days
0241 02e7 be 85    ldx      month
0242 02e9 ad c6    bsr      nostart    ;send months
0243 02eb ad de    bsr      stop       ;stop condition
0244 02ed 81          rts

```

iicc5

page 6

0246

0247

0248

0249

0250

0251 02ee

0252 02ee b7 88

0253 02f0 cd 04 0b

0254 02f3 b6 88

0255 02f5 cd 04 0f

0256 02f8 81

0257

0258

0259

0260

0261

0262

0263

0264

0265 02f9

0266 02f9 a6 00

0267 02fb b7 81

0268 02fd 4c

0269 02fe b7 82

0270 0300 cd 02 d2

0271 0303 ae d1

0272 0305 ad a6

0273 0307 3f 82

0274 0309 ae ff

0275 030b ad a4

0276 030d b6 0c

0277 030f b7 83

0278 0311 ae ff

0279 0313 ad 9c

0280 0315 b6 0c

0281 0317 b7 84

0282 0319 ae ff

0283 031b cd 02 b1

0284 031e b6 0c

0285 0320 b7 86

0286 0322 3c 82

0287 0324 ae ff

0288 0326 cd 02 b1

0289 0329 cd 02 cb

0290 032c b6 0c

0291 032e b7 85

0292 0330 81

* Output 2 BCD digits in A as ASCII chars.

```
bcd      equ      *
          sta      sava      ;save A
          jsr      outlhf     ;output left half
          lda      sava
          jsr      outrhf     ;output right half
          rts
```

*
* the following routine reads a data byte.
*

```
read     equ      *
          lda      #$00
          sta      control
          inca
          sta      ack        ;high ack bit (ack non-zero)
          jsr      addrctl    ;send address/control
          ldx      #raddr     ;(r/v=1)
          bsr      w_start    ;send address with start condition
          clr      ack        ;low ack bit
          ldx      #$ff       ;and read 4 data bytes
          bsr      nostart    ;keep mosi open drain (high)
          lda      spdr       ;get received data
          sta      hour       ;hours
          ldx      #$ff
          bsr      nostart
          lda      spdr
          sta      min        ;minutes
          ldx      #$ff
          jsr      nostart
          lda      spdr
          sta      day        ;days
          inc      ack        ;high ack bit for last bit received
          ldx      #$ff
          jsr      nostart
          jsr      stop       ;end session
          lda      spdr
          sta      month      ;months
          rts
```

iicc5

page 7

0294

0295

0296

0297

0298

0299

0300

0301

0302

0303

0304

0305

0306

0307

0308

0309

0310

0311

0312

0313

0314

0315

0316

0317

0318

0319

0320

0321

0322

0323

0324

0325

0326

0327

0328

0329

0330

0331

0332

0333

0334

0335

0336

0337

0338

```
*****
* This service routine increments the seconds counter
* and displays the clock data on the screen every second.
*****
```

```
dispdata equ *
          lda   #$0d
          jsr   outchar    ;send <CR>
          jsr   read       ;read 4 bytes from clock
          lda   month      ;display month
          and   #$1f
          bsr   bcd        ;output 2 BCD digit
          lda   #'/'
          jsr   outchar    ;outout '/'
          lda   day        ;display day
          and   #$3f
          bsr   bcd
          lda   #$20
          jsr   outchar    ;output space
          lda   hour       ;display hours
          and   #$3f
          bsr   bcd
          lda   #' :
          jsr   outchar    ;output ':'
          lda   min        ;display minutes
          and   #$7f
          jsr   bcd
          lda   #' :
          jsr   outchar    ;output ':'
          lda   sec        ;display seconds
          jsr   bin_dec    ;convert seconds to BCD and display
          lda   sec        ;read seconds byte
          inca    ;increment it
          cmp   #60
          bne   not_sixty ;not 60 yet
          clra
          not_sixty sta   sec    ;update seconds counter
          rts
```

iicc5

page 8

0340

0341

0342

0343

0344

0345

0346 0376

0347 0376 5f

0348 0377 d6 02 71

0349 037a a1 04

0350 037c 27 06

0351 037e cd 03 ca

0352 0381 5c

0353 0382 20 f3

0354 0384 81

0355

0356

0357

0358 0385

0359 0385 a6 0d

0360 0387 cd 03 ca

0361 038a a6 0a

0362 038c cd 03 ca

0363 038f 81

0364

0365

0366

0367

0368

0369

0370

0371

0372

0373

0374

0375

0376 0390

0377 0390 bf 89

0378 0392 a6 08

0379 0394 b7 8a

0380 0396 9d

0381 0397 9b

0382 0398 00 00 fd

0383 039b a6 02

0384 039d ad 63

0385 039f 00 00 f4

0386

0387

0388 03a2 a6 02

0389 03a4 ad 5c

0390 03a6 a6 06

0391 03a8 ad 58

0392 03aa 01 00 00

0393

0394 03ad 36 8b

0395 03af b6 8a

0396 03b1 4a

0397 03b2 c7 00 8a

0398 03b5 26 eb

0399

0400 03b7 9d

0401 03b8 a6 02

0402 03ba ad 46

0403 03bc a6 06

0404 03be ad 42

0405 03c0 b6 8b

0406 03c2 a4 7f

0407 03c4 ad 06

0408 03c6 b6 8b

0409 03c8 9b

0410 03c9 81

* The following are the various routines associated with
* displaying data.

```
outmsg    equ    *                ;print character string
          clr    clr                ;clear
prtmsg    lda    msg,x             ;get message character
          cmp    #04               ;EOT yet?
          beq    finish            ;yes.
          jsr    outchar           ;output character
          incx   prtmsg            ;increment index
          bra    prtmsg
finish    rts
```

```
crlf      equ    *                ;print new line
          lda    #$0d              ;carriage return
          jsr    outchar           ;output character
          lda    #$0a              ;line feed
          jsr    outchar           ;output character
          rts
```

* Register A and location InChar receive the character typed,
* parity stripped and mapped to upper case. X is unchanged.
* For HC05C5 PA1 and PA0 are txd and rxd respectively.
* i.e. transmit from PA1 and receive from PA0

* Interrupts are masked on entry and unmasked on exit.

```
inchar    equ    *                ;input character from terminal
          stx    xtemp             ;save X
          lda    #8                ;number of bits to read
          sta    count
getc4     nop                     ;unmask to allow service, then
          sei                     ;mask while looking for start bit
          brset 0,porta,*         ;wait for hilo transition
          lda    #2
          bsr    delay            ;delay 1/2 bit to middle of start bit
          brset 0,porta,getc4    ;false start bit test
          *
          * main loop for getc
getc7     lda    #2
          bsr    delay            ;6 common delay routine
          lda    #6
          bsr    delay            ;6
          brclr 0,porta,getc6    ;5 test input and set c-bit
getc6     ror    InChar           ;5 add this bit to the byte
          lda    count            ;3 time-wasting way to decr count
          dec    count            ;3
          sta    >count          ;5 extd addr to waste extra cycle
          bne   getc7            ;3 still more bits to get(see?)
          *
          * re-enable interrupts
          nop                     ;2
          lda    #2
          bsr    delay            ;3 wait out the 9th bit
          lda    #6
          bsr    delay            ;3
          lda    InChar          ;get assembled byte
          and    #1111111        ;mask off parity bit
          bsr    putc1            ;echo it back
          lda    InChar          ;get assembled byte
          sei                     ;re-enable interrupts
          rts
```

```

iicc5
0412
0413
0414 03ca outchar equ * ;output character to terminal
0415 03ca bf 89 stx xtemp ;don't forget about X
0416 03cc putc1 equ * ;sneaky entry from getc to avoid clobbering x
0417 03cc b7 8c sta OutChar
0418 03ce b7 8d sta atemp ;save it in both places
0419 03d0 a6 09 lda #9 ;going to put out
0420 03d2 b7 8a sta count ;9 bits this time
0421 03d4 5f clrxc ;for very obscure reasons
0422 03d5 98 clc ;this is the start bit
0423 03d6 9b sei ;mask interrupts while sending
0424 03d7 20 02 bra putc2 ;jump in the middle of things
0425
0426 * main loop for outchar
0427
0428 03d9 36 8c putc5 ror OutChar ;5 get next bit from memory
0429 03db 24 04 putc2 bcc putc3 ;3 now set or clear port bit
0430 03dd 12 00 bset 1,porta ;5
0431 03df 20 04 bra putc4 ;3
0432 03e1 13 00 putc3 bclr 1,porta ;5
0433 03e3 20 00 bra putc4 ;3 equalize timing
0434 03e5 a6 02 putc4 lda #2
0435 03e7 ad 19 bsr delay ;6
0436 03e9 a6 06 lda #6
0437 03eb ad 15 bsr delay ;6
0438 03ed 3a 8a dec count ;5
0439 03ef 26 e8 bne putc5 ;3 still more bits
0440 03f1 12 00 bset 1,porta ;5 send stop bit
0441 03f3 9d nop ;2 re-enable interrupts
0442 *
0443 03f4 a6 02 lda #2
0444 03f6 ad 0a bsr delay ;6 delay for the stop bit
0445 03f8 a6 06 lda #6
0446 03fa ad 06 bsr delay ;6
0447 03fc be 89 ldx xtemp ;3 restore X and
0448 03fe b6 8d lda atemp ;3 of course A
0449 0400 9b sei ;2 re-enable interrupts
0450 0401 81 rts ;6
0451
0452
0453
0454 * delay --- precise 1/2 bit time delay for getc/putc
0455 *****
0456
0457 * caller loop overhead assumes 24 cycles in external loop
0458
0459 0402 delay equ *
0460 0402 4a deca
0461 0403 26 fd bne delay
0462 0405 9d nop
0463 0406 81 rts
0464
0465 * 1/2 bit delay = 24 cycles overhead + (6*A)+8+8, where A=2
0466 * 1 bit delay = 24 cycles overhead + [(6*A)+8+8] + [(6*B)+8+8], A=2, B=6
0467
0468 -----*
0469 * delays for baud rate calculation
0470 -----*
0471 0407 20 delays fcb 32 ;300 baud
0472 0408 08 fcb 8 ;1200 baud
0473 0409 02 fcb 2 ;4800 baud
0474 040a 01 fcb 1 ;9600 baud

```


iicc5

page 10

0476

0477

0478

0479

0480

0481 040b

0482 040b 44

0483 040c 44

0484 040d 44

0485 040e 44

0486 040f a4 0f

0487 0411 ab 30

0488 0413 cd 03 ca

0489 0416 81

0490

0491

0492

0493

0494

0495 1ffa

0496 1ffa 02 00

0497 1ffc 02 00

0498 1ffe 02 00

* Output the left nibble of Acc A as ASCII character.

```
outlhf equ *
        lsra
        lsra
        lsra
        lsra
        and  #0f
        add  #030 ;make ASCII
        jsr  outchar ;send character to terminal
        rts
```

```
org $1ffa
irqv fdb begin
swiv fdb begin
resetv fdb begin
```

page 11

```

iicc5
InChar      008b *0045 0394 0405 0408
OutChar     008c *0046 0417 0428
ack         0082 *0036 0071 0182 0269 0273 0286
addrcntl   02d2 *0219 0095 0102 0109 0270
again       023c *0086 0088
atemp       008d *0047 0418 0448
bcd         02ee *0251 0159 0307 0313 0319 0325
begin       0200 *0059 0496 0497 0498
bin_dec     0298 *0146 0330
cdelay      008e *0048 0068
control     0081 *0035 0094 0101 0108 0222 0267
count       008a *0044 0379 0395 0397 0420 0438
crlf        0385 *0358 0073 0089
day         0086 *0040 0079 0239 0285 0311
ddra        0004 *0021 0062
ddrb        0005 *0022 0065
ddrc        0006 *0023
delay       0402 *0459 0384 0389 0391 0402 0404 0435 0437 0444 0446
            0461
delays      0407 *0471 0067
dispdata    0331 *0300 0114
finish      0384 *0354 0350
getc4       0396 *0380 0385
getc6       03ad *0394 0392
getc7       03a2 *0388 0398
getdate     0285 *0128 0075 0078 0081 0084
hi_ack      02c2 *0187 0183
hi_ackl     02c4 *0188 0185
hour        0083 *0037 0082 0235 0277 0317
inchar      0390 *0376 0077 0080 0083 0086 0129 0136
irqv        1ffa *0496
lo_ack      02be *0184
min         0084 *0038 0085 0237 0281 0323
month       0085 *0039 0076 0241 0291 0305
msg         0271 *0118 0348
no_tens     02a0 *0152 0149
nostart     02b1 *0175 0223 0236 0238 0240 0242 0275 0279 0283 0288
not_sixty   0373 *0337 0335
outchar     03ca *0414 0302 0309 0315 0321 0327 0351 0360 0362 0488
outlhf      040b *0481 0253
outmsg      0376 *0346 0074
outrhf      040f *0486 0255
porta       0000 *0018 0061 0113 0115 0382 0385 0392 0430 0432 0440
portb       0001 *0019 0064 0173 0174 0184 0187 0188 0189 0190 0203
            0204 0205
portc       0002 *0020
prtmsg      0377 *0348 0353
putc1       03cc *0416 0407
putc2       03db *0429 0424
putc3       03e1 *0432 0429
putc4       03e5 *0434 0431 0433
putc5       03d9 *0428 0439
raddr       00d1 *0028 0271
ram         0080 *0031 0033
read        02f9 *0265 0303
resetv      1ffe *0498
sava        0088 *0042 0135 0138 0157 0158 0252 0254
savx        0087 *0041
sec         0080 *0034 0112 0329 0332 0337
sec_pin     0266 *0113 0116
senddta     02db *0234 0103
spcr        000a *0024 0070 0176 0180
spdr        000c *0026 0177 0276 0280 0284 0290
spsr        000b *0025 0178
stop        02cb *0202 0096 0110 0243 0289
sub_more    0299 *0148 0151
swiv        1ffc *0497
w_start     02ad *0172 0221 0272
waddr       00d0 *0029 0220
wait        02b5 *0178 0178
xtemp       0089 *0043 0377 0415 0447

```


Pulse Generation and Detection with Microcontroller Units

By Mike Pauwels

INTRODUCTION

This application note examines two common interfaces between microcontroller units (MCUs) and external circuitry — pulse generation and pulse detection. Several families of Motorola MCUs and a variety of pulse applications are considered. Code segments and listings are also included.

PULSE GENERATION

MCUs are often required to generate timed output pulses — i.e., signals asserted for a specified period of time. The application can be strobing a display latch, transmitting a code, or metering a reagent in a process control system; however, each application has specific requirements for pulse duration and accuracy. This application note examines methods of generating these pulses in relationship to timing accuracy, coding efficiency, and other controller requirements.

The following paragraphs describe the timing of the signals — the start time and duration of the pulse. All pulses can be divided into three basic classifications: short pulses, long pulses, and easy pulses. Each class of pulses is considered using three MCUs with different timer structures.

On the low end of the scale is the MC68HC05J1. The simple timer in this device limits the accuracy of short pulses and requires a larger amount of software investment to produce a given pulse. The second MCU, the MC68HC705C8 and similar devices, has a 16-bit timer that is somewhat more powerful and flexible than the MC68HC05J1 timer. Finally, the MC68HC11A8 offers additional features in the 16-bit timer system, as well as the possibility of producing multiple pulses simultaneously.

Because time measurements are being considered, the clock frequency for the MCU is significant. For this discussion, each MCU is assumed to be operating at a 2.0-MHz internal, implying a 4.0-MHz crystal for the MC68HC05 and an 8.0-MHz crystal for the MC68HC11A8. The maximum speeds of these devices is somewhat higher, but these are commonly used values. Of course, these MCUs can all be operated at much slower clock speeds also. All times should be scaled to the actual clock frequency.

Short Pulses

The classification of short pulses may vary according to the accuracy of the required pulse and the available MCU resources. In general, pulses of a few tens of microseconds and longer are relatively easy to produce. Below this broad limit, the methods used to generate short pulses may vary greatly according to the specific requirements. To produce a strobe pulse whose minimum required duration is in the order of magnitude of the clock period only requires writing a port bit high, then low in consecutive operations:

PULSE	BSET	BIT0, PORTA
	BCLR	BIT0, PORTA

This produces a pulse duration of 2.5 μ s duration in the MC68HC05, and 3.0 μ s in the MC68HC11A8. The longer time in the MC68HC11A8 is a consequence of a longer BSET/BCLR instruction formation — three bytes versus two bytes in the MC68HC05. This is compensated for by the ability to set and clear multiple bits in one instruction. The MC68HC11A8, however, provides for a 0.5 μ s minimum pulse by using the resources of two timer compare registers.

If the requirement for the pulse is longer than 2 μ s, the above pair of instructions can be separated by no operation (NOP) instructions or even by useful instructions to stretch to the desired pulse width. There are two problems with this option. First, padding the instructions with NOPs consumes MCU resources. If there is some task that the MCU can accomplish between the set and clear, this is not too serious. More difficult is the possible requirement that the pulse duration be run-time variable. The flexibility of the busy-wire pulse timing can be extended by adding a loop:

SETUP	LDA	DURATION
PULSE	BSET	BIT0, PORTA
LOOP	DECA	
	BNE	LOOP
	BCLR	BIT0, PORTA

The duration of the pulse is:

$2.5 + 3.0 * \text{DURATION } \mu\text{s}$ for the MC68HC05,
and

$3.0 + 2.5 * \text{DURATION } \mu\text{s}$ for the MC68HC11A8

Of course the previous code could be padded with any number of NOPs at 1.0 μ s or with branch nevers (BRNs) at 1.5 μ s either inside or outside the timing loop for more precise values. However, the variable resolution is 2.5 or 3.0 μ s.

Note that in these cases the on-board timers were not used. In the case of these short pulses, the overhead of setting-up and reading the timers would be about as long as the pulse being driven. When the required pulse width is long enough to use the timer, easy pulses are produced.

Easy Pulses

To produce a 10 ms pulse with the MC68HC11A8 controller, force an output compare pin high and read the timer (in an uninterrupted sequence). Add the 10 ms to the timer value and store the result in the corresponding output compare register. Next write the corresponding output level (OLVL) bit to zero and enable the interrupt (if desired). The pulse completes automatically. Three questions arise: 1) What is the shortest pulse that can be produced in this manner? 2) What considerations must be made in the MC68HC705C8 timer which does not have a force register? 3) What is the equivalent procedure for the MC68HC05J1 timer?

In the MC68HC11A8, two output compare registers can be combined to drive a single output. The elapsed time between the two events can be as little as one clock time; 0.5 μ s if the prescaler is one. The code is as follows:

```

PULSE      LDD      TIMER
           ADDD     #50          Delay start of pulse
*
* The delay is selected according to
* timer prescaler, interrupts, etc.
* (min 33)
*
           STD      TOC1
           ADDD     #1          Pulse Width
           STD      TOC2
           LDAA     #$40       Drive A6 High
           STAA     OC1M
           STAA     OC1D
           LDAA     #$80
           STAA     TCTL1     Drive A6 Low
*
* ENABLE INTERRUPT, ETC
*

```

With the MC68HC705C8 timer system, there is no force bit for compare. The only way to drive the timer compare (TCMP) line high is to set the OLVL bit in the timer control register (TCR) and wait for a match. The exact start time of the pulse is easily obtained from the output compare register (OCR), so pulse accuracy is unaffected for moderate pulses. Often the pulse is started as soon as possible, if for no other reason than to complete the pulse setup routine. The following code segment provides a pulse start in 12 μ s, assuming no interrupts.

```

*
* START THE PULSE
*
           BSET     OLVL, TCR
* OUTPUT_COMPARE: = TIMER + DELAY
           LDX     ACHR          MUST BE READ FIRST
           LDA     ACLR          TIMER = X:A
           ADD     $DELAY
           BCC     OC1          MARK TIME
           INCX
OC1        STX     OCHR          INHIBITS TOC
           STA     OCLR          ENABLES TOC
*
* IF DELAY IS CORRECT, PULSE WILL
* TURN ON IMMEDIATELY

```

Using a value for DELAY of about 21 (cycles) results in an average latency of 12 μ s after the beginning of this routine. Note that loads and stores to the 16-bit registers are always performed high-byte first to take advantage of special hardware that maintains coherency in 16-bit data transfers. The pulse will turn on 1 μ s later when there is a carry out of the low-byte add, which should occur about 1 in 12 times.

The programming of a moderate length pulse is now quite trivial. Simply add the desired pulse width (at 2 μ s per bit) to the value stored in the output compare. Write the new value to the OCR and set the OLVL bit to zero. To finish code segment:

```

*WAIT FOR PULSE TO BE SET
HERE      BRCLR      OCF, TSR, HERE
          LDA        PW_L      PULSE WIDTH LS BYTE
          ADD        OCLR
          TAX
          LDA        PW_H      TEMPORARILY
          ADC        OCHR      PULSE WIDTH MS BYTE
          STA        OCHR      INCLUDES CARRY
          STX        OCLR      INHIBITS TOC
*
*DONE!

```

The interrupt structure is not required to generate pulses. The 16-bit timers on the MC68HC11A8 and the MC68HC705C8 will automatically drive the falling edge of these pulses without software intervention. On the MC68HC05J1, there is no hardware timer interface. To drive moderate length pulses with this device, employ the interrupts so that useful work can be performed while the pulse is being timed. Consider a 10 ms pulse using the MC68HC05J1.

The simple timer of the MC68HC05J1 provides only the capability of being interrupted periodically. The source of interrupt can be a timer overflow or a real-time interrupt (RTI). The choice of interrupt times is given in Table 1:

Table 1. RTI and COP Rates ($f_{op} = 2$ MHz)

RTI/RT0	RTI	COP
0 0	8.2 ms	57.3 ms
0 1	16.4 ms	114.7 ms
1 0	32.8 ms	229.4 ms
1 1	65.5 ms	458.8 ms

Consider the algorithm for the timing of a pulse as counting "ticks" on a clock. Initially, it seems the ticks of the timer must be counted — 5,000 ticks (2 μ s per tick) for the desired period of 10 ms. However, the timer overflow and real-time interrupts of the MC68HC05J1 provide long ticks that sound their completion with interrupts. Instead of 5,000 short ticks, count as follows:

1 RTI tick of 8192 μ s	=	8,192
3 TOF ticks of 512 μ s	=	1,536
544 cycles of 0.5 μ s	=	272
TOTAL		<u>10,000</u>
	=	10.000 ms

For most of this time background tasks can continue processing. The 544 cycles of busy-wait time include necessary work to set up the pulse. The key problem is the required timing of the start of the pulse. If the start time is flexible the design of the pulse could follow the pattern of Figure 1.

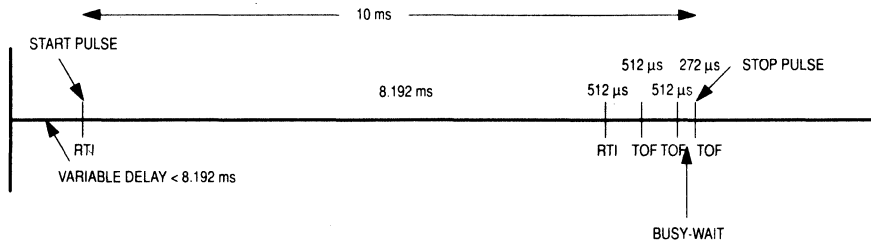


Figure 1. Time Line for a 10 ms Pulse with Flexible Start Time

Start the pulse on the next RTI service routine, then count timer overflow flags (TOFs) after the next RTI until the final sequence, which is timed by a busy-wait counter. Careful calculation of the latencies and instruction cycles produces a pulse with a high degree of accuracy.

When the start time is not as flexible, a different approach is necessary. Since it is now impossible to align the RTI boundaries with the pulse, use only the TOF ticks to time the pulse. To turn the pulse on as soon as possible, read the value of the timer at turn-on. Calculate the time until the next overflow, add the predicted turn-off execution time, and then determine how many full TOF periods are in the remainder. After subtracting these long ticks, the remaining value is the busy-wait. A time line for this approach is given in Figure 2.

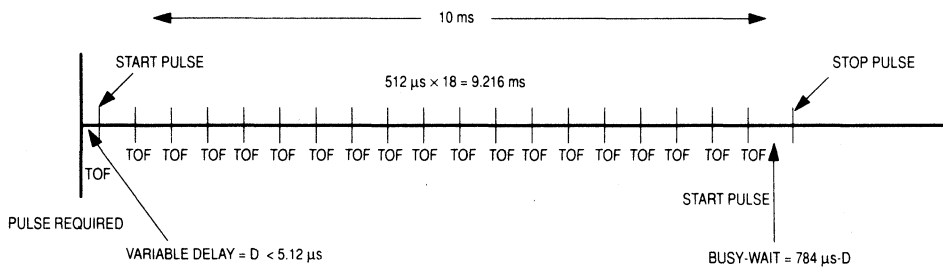


Figure 2. Time Line for a 10 ms Pulse with Immediate Start Time

Since an interrupt occurs every 512 μ s, the performance of the MCU degrades slightly — about 10% versus 1% for the first approach. The following code yields a 10.0 ms pulse on port A1, with a latency of 2.5 μ s after the code is entered:

```

* ASSUME THE DESIRED PULSE WIDTH,
* RESOLVED TO 2  $\mu$ s PER BIT,
* IS STORED IN A TWO-BYTE LOCATION
* LABELED: PW_H:PW_L. FOR A
* PULSE WIDTH OF 10 ms THIS
* VALUE WOULD BE $1388

* TURN ON THE PULSE

START      BCLR      TOF, TCSR
           LDA       TIMER
           BSET     BIT0, PORTA
           COMA
           SBA       PW_L      = TIME REMAINING
           BCC     PW_L      LOW BYTE OF PULSE
           DEC     PW_H      BORROW 1
PW1        LDX     #$AA      RE-SCALE LOW BYTE TO
           MUL     PW_L      ...3  $\mu$ s PER BIT
           STX     PW_L
           BSET     TOFE, TCSR
           CLI

*
* THE TIMER INTERRUPT DOES THE REST
* OF THE WORK:
*
TOFI       DEC     PW_H
           BNE     END_T
           LDA     PW_L
           BEQ     PLS_L
LOOP       DECA
           BNE     LOOP
PLS_L      BCLR     BIT0, PORTA
END_T      RTI

```

There will be some small inaccuracies due to latency of the interrupt and border conditions for the pulse width. The pulse can be refined, but if one-clock precision is required, choose another processor.

Long Pulses

The idea of using the interrupt structure to count long "ticks" can be expanded beyond one byte. If a two-byte decrement is performed in the previous MC68HC05J1 example, pulses up to 30 seconds in length can be generated. The inaccuracies are the same in absolute terms as for the shorter pulses; therefore the percent of error is much smaller.

The same approach is used to expand the pulse width that can be generated by the 16-bit timers in the MC68HC705C8 and the MC68HC11A8 processors. With the help of the output compare function, one-tick accuracy with very long pulses is possible. The accuracy of the output is determined only by the accuracy of the crystal. The code listed in Appendix A has been tested in an MC68HC05C4 and produced pulses in the order of one minute with an accuracy of one part per million. Code to generate

long pulses with an MC68HC11A8 is similar. Since the timer interrupts are used to count the ticks, most of the MCU resources are available to do background tasks. For example, the timer interrupt routine consumes less than 25 μ s every 131 ms. This represents about 0.2% of the processing power of the MCU. The actual code takes about 200 bytes of memory. The pulse will be precise if the interrupts are not masked for more than about 130 ms at a time. Beyond that limit, whole ticks of 131 ms will be added.

Finally, the MC68HC11A8 timer system provides for two timer output functions to drive a single pin. With this MCU, the start time and end time of the pulse can be driven independently with differences of as little as one count between the two pulse edges.

Summary — Pulse Generation

Many MCU systems interface to hardware systems by means of timed pulses. Modern MCUs handle these pulses in three different ways depending on the hardware timing structure available and the length of the pulse.

Short pulses, ranging in length from as short as a microsecond to a few tens of microseconds, are usually timed with “busy-wait” loops. There is simply not enough time to set up a peripheral to control a pulse of short duration. The accuracy and resolution of these pulses is determined in part by the discrete execution time of branch instructions in the controller. The MC68HC11 can drive a pulse as short as a microsecond, resolved to a microsecond, by using the resources of two timer compare registers.

Moderate length pulses are simple to drive automatically using the 16-bit timer available in the MC68HC11A8 and many of the MC68HC05 MCUs. These are set-and-forget systems that run to completion typically in 131 ms. In the simpler MC68HC05 MCUs, there is no 16-bit timer, and the moderate length pulses must use the timer overflow interrupt to count out large chunks of the pulse time while some background task is being performed.

The approach is similar in the MCUs with the 16-bit timer when the desired pulse is greater than 131 ms. Multiple timer overflows can be counted in a few memory locations to produce very long pulses.

For more complex timing functions, a system may require a separate timing processor. In some complex control applications, an MC68HC11A8 or an MC68HC05 is employed as a peripheral timer to a larger computational engine. A variation on this theme is the time processing unit (TPU) in the MC68332. This complex timing system can perform several different functions on 16 different channels simultaneously, independent of the main processor. Information on the MC68332 is available from your Motorola Sales Office.

PULSE DETECTION

Another system problem encountered when applying an MCU to a physical system is the detection and measurement of pulses. These can range from the actuation of a pushbutton to pulse codes detection, detection of the period of rotation of an engine, and accumulation of the ‘on’ time of a process control valve. The periods can range from microseconds to minutes, hours, or more.

There are several parameters that characterize a pulse, as Figure 3 illustrates. As far as a digital system is concerned, most of these parameters cannot be measured directly by a digital device such as an MCU. Indeed, some parameters such as the voltage level must be modified before the pulse

is applied to the MCU. If the values of these parameters are interesting to the system, then an external device such as an analog-to-digital converter is required. Other parameters may not be measurable by the MCU, including the signal rise and fall times and the presence of noise on the signal.

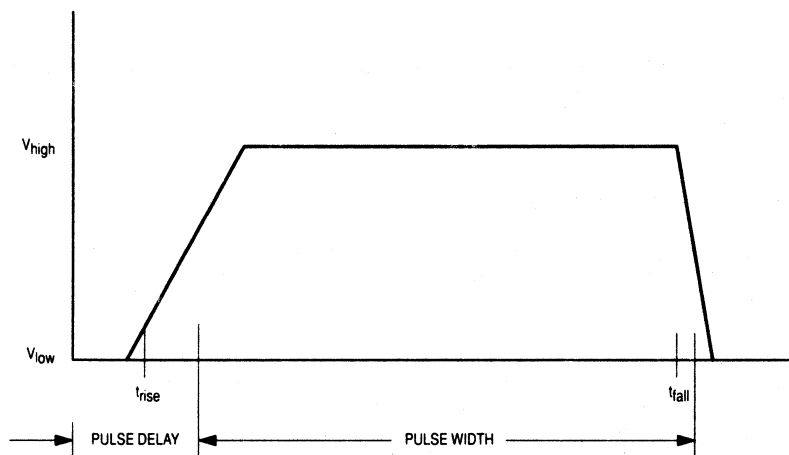


Figure 3. Characteristics of a Pulse

Digital pulses convey information only in the timing of their signals, assuming that all voltage signals vary between V_{SS} and V_{DD} , and that rise and fall times are sufficiently fast to be unambiguous to the processor. The parameters of interest are the start-time and duration of the pulse. Noise, if it exists, is interesting only to the extent that it can be seen by the controller, and in that case, provision must be made to reduce its effects.

There remains one significant question to address before software design can commence. What is the expected duration range of the pulse? There are no effective maximum limits on the duration which can be measured; but very short pulses may require the support of on-chip or off-chip hardware. A related characteristic is the start-time of the pulse, measured from some reference. This can be thought of as the measurement of a pulse off-time, and hence is not significantly different from the duration measurement. Also important is the required accuracy of the measurement, specified in absolute or relative terms.

One important choice the designer makes in addressing system problems is the type of MCU that will be used. Most MCUs have some sort of timing device on-chip. Within the five basic families of Motorola processors are several timer variations. These range from the simple counter in the MC68HC05J1 to the sophisticated TPU of the MC68332. The former is useful only for the simplest requirements, while the latter can measure pulses accurately without intervention of the CPU. The choice for most applications is usually between an MC68HC11 and one of the large family of MC68HC05 devices.

The timer on the MC68HC11A8 provides as many as four hardware input signals with several hardware registers to measure input events. By combining two input capture functions, or by using the clock gate input of the MC68HC11A8, many pulse measurement problems are easily solved. It is more difficult to address the problems with the 16-bit timer system found on the most popular MCU family, the MC68HC05.

Consider the accuracy limitations of the MC68HC(7)05Cx 16-bit timer. The timer counter itself is incremented once for every eight cycles of the MCU crystal frequency. A 4.0-MHz crystal provides a count resolution of 2 μ s. With short pulses, this resolution may be a contributor to accuracy limitations. For example, measuring a 50 μ s pulse, this resolution will produce a count of 25 with a 1-bit quantizing error, an uncertainty of 4%. However, in measuring a one minute pulse, the quantizing error is 0.0000033%.

In the case of the longer pulses, the accuracy of the crystal can contribute far more to the loss of precision. A limited sampling of clock frequencies on MC68HC05 Evaluation Modules indicates that typical crystals may produce errors of 0.001%. While crystals can be selected or trimmed to much higher accuracy, it is important not to specify accuracies from the software that cannot be supported by the hardware.

Consider four general classes of pulses to detect: 1) very fast pulse, say 20 μ s or less; 2) longer pulses up to 130 ms; 3) long pulses; and 4) noisy pulses. The second class is almost trivial with the TCAP feature of the MC68HC05. Indeed, these are the most common class of pulses, and the hardware does almost all of the work. These are considered a special case of the third class of pulses. The other three classes require a bit more study.

Short Pulses

To measure very fast pulses with the MC68HC05, it is necessary to deal with interrupt latency which can be as much as 10 μ s. If an IRQ is triggered on the start of the pulse, the pulse may have ended by the time code is executed in the interrupt routine. Accuracy is limited by the latency of the system. An example of the code necessary to measure these pulses is given below:

```

INTRUPT:
        CLRA
        BIL          END_PULS
T_LOOP:  INCA          COUNT_LOOPS
A:       BIH          T_LOOP

END_PULS:
*
*
```

After the pulse is driven low on the IRQ line, the timed wait is executed for the rising edge which enables detection of a very short pulse. At END_PULS, the Accumulator has a measurement of the length of the pulse resolved to 3 μ s per bit. Assuming the interrupts are not masked the worst case time to get to point A the first time is 13.4 μ s (11.5 μ s if MUL is not used in the background). The fastest time is 9 μ s. Any pulse shorter than this will result in a zero time value. If the pulse value is greater than zero, the pulse width is 3 μ s times the accumulator value plus a latency time of 9 – 13.5 μ s. Finally, the longest pulse time that this routine can reasonably measure before the accumulator will overflow is about 770 μ s. The interpretation of the result is left to the user.

If a short pulse is brought in on the TCAP line, there is additional latency to consider. If there is sufficient time to reverse the IEDG bit and clear the ICF (minimum about 20 μ s), this is a class 2 pulse. If the pulse is shorter than this, the input capture function may miss the second edge. Unlike the MC68HC11A8 input capture functions, the MC68HC05 timer pin (TCAP) is not directly detectable. Precautions must be taken in the hardware design if very short pulses are possible. For example, a port line could be wired to the TCAP pin and the state of the pin could be tested with a BRSET/BRCLR. The minimum resolvable pulse length is still no better than the IRQ driven case. However, using the

TCAP input offers capability to measure pulses of either polarity up to 131 ms in length and with a resolution of 2 μ s.

Of course, if the pulse is expected to be short and the start time can be predicted, a busy-wait can be executed for both ends of the pulse. In this case it is necessary to continually test the state of the input pin and branch accordingly. For example, if the expected pulse length is between 5 and 100 μ s, execute a string of tests as shown below:

```

PULSE :
      CLRA
P0     BRCLR   PIN, PORT, P0      WAIT FOR THE FIRST EDGE
      BRSET   PIN, PORT, P1      ACTUALLY USING THE CODE TO
                                  MEASURE
      BRSET   PIN, PORT, P2
      .
      .
      BRSET   PIN, PORT, PN
PN     INCA
      INCA
      .
      .
P2     INCA
P1     INCA
*
* ACC CONTAINS PULSE WIDTH OF 2.5  $\mu$ s
* PER BIT

```

This code yields a resolution of 2.5 μ s for any pulse down to 2.5 μ s. Below that, the pulse may be missed. As the expected pulse length gets larger, this code becomes unwieldy and finally impossible.

The addition of an instruction loop shortens the code at the expense of resolution:

```

PULSE :
      CLRA
P0     BRCLR   PIN, PORT, P0
P1     INCA
      BRSET   PIN, PORT, P1
*
* ACC CONTAINS PULSE AT 4  $\mu$ s PER BIT
*

```

For longer (class 2) pulses, use the input capture register of the timer to do all the work. Where the pulse is more than a few tens of microseconds long, the interrupt structure works well to measure the pulse within the accuracy of the crystal. The rising edge of the pulse triggers a first interrupt, and the service routine enables the interrupt on the falling edge. By reading the input capture register on each edge, the exact pulse length can be measured. This class of pulses is included in a special case of the long pulses below.

Longer Pulses

What if the pulse length exceeds the rollover time of the timer? By counting the rollovers, a pulse of arbitrary length can be measured. Consider the possibility of a 60 second pulse that must be detected

and measured accurately. If the timer counts 2 μ s per bit, 30 million counts must be accumulated. To store this information, $\log_2(30,000,000) = 25$ bits, or 4 bytes are needed. To be precise, a value of \$1 C9 C3 80 is expected.

The 16-bit timer will automatically record the edges of the pulse. Ignoring the overflow, if the start time is subtracted from the end time the result will yield the two least significant bytes of the pulse width. In the 60 second example, if the pulse is exactly correct, the difference between the output compare value at the start of the pulse and the value at the end of the pulse will be \$C380. Between those two pulse events, the timer will roll over \$1C9 times (= 457). Counting those rollovers exactly will determine the pulse length. The timer overflow facility will allow a count of the rollovers under interrupt control. Some problems remain in arbitrating the interrupts and protecting for boundary conditions, which will be discussed below.

The general approach taken for the MC68HC05Cx TCAP works as well for the MC68HC11 family when a single input capture function is used for measurement.

Appendix B is a listing of an MC68HC05 program which can measure very long pulses with single tick accuracy. The program was tested with the pulse generation problem listed above and appears to work within the accuracy of the crystal. Some adjustment may be necessary when this software is integrated into the user's program, particularly insofar as the interrupt latency is affected, but the basic structure of the routines will perform the measurement function. Note that class 2 pulses can be measured with this routine as it stands, although some code savings can be realized if the pulse to be measured is known to be contained in less than 4 bytes.

Three particular areas should be attended to when incorporating this software in a larger project. The measurement routine uses mutually exclusive interrupts and no subroutines, therefore its contribution to stack push is seven bytes. Add this to any other subroutine and interrupt stack usage to determine the maximum stack depth and therefore the available RAM.

If other interrupts are used, remember that the interrupt mask is automatically set when the interrupt routine is entered. If the mask cannot be cleared, the execution time of the other interrupt, plus its latency, must be kept somewhat less than 500 μ s (or the pulse width, whichever is smaller) to preserve the accuracy of the measurement routine. The same is true if critical code sections must be preserved with SEI...CLI instructions.

Within these limitations, the automatic timing features of the TCAP will provide accurate measurement of the pulse. The 500 μ s limitation is necessary to assure the correct handling of the boundary conditions when an overflow coincides with a pulse edge. If the interrupts must be masked for longer periods, the boundary conditions handling can be modified.

The third area to consider is the effect of the interrupts on execution speed of the processor. The pulse measurement routines take less than 0.02% of the clock cycles when measuring long pulses, so the routine will not significantly affect the throughput of most programs, however, each timer overflow interrupt takes about 24 μ s, so software timing loops and critical sections can be affected.

Noisy Pulses

The important thing to remember about noisy pulses is that a noise edge often cannot be distinguished from a pulse edge. This is particularly true when the input capture register is used to detect the edge. But even when the edge is polled, a momentary change in the signal level can be erroneously interpreted.

In general, it is difficult to measure any true pulse that contains noise pulse durations in order of magnitude of the measurement resolution. This means that signals must be free of 1 μ s noise pulses for most MCU detection and measurement algorithms. The MC68HC11A8 pulse accumulator function in gated mode can be used to measure the total asserted time of a very noisy pulse.

Often, the easiest way to eliminate the ambiguity of minor noise is with some low pass hardware filtering. Remember that low pass filtering will also round and delay the edges of the pulse. The delay will contribute more or less to the accuracy of the measurement. In addition, sampled edges can be double-checked in our busy-wait algorithms with the addition of a single instruction per edge:

```

PULSE:
          CLR A
P0        BRCLR  PIN, PORT, P0
          BRCLR  PIN, PORT, P0
P1        INCA
          BRSET  PIN, PORT, P1
          BRSET  PIN, PORT, P1
*
* ACC HAS PULSE AT 6.5  $\mu$ s PER BIT

```

Sophisticated digital filter algorithms can be used to extract a pulse from very noisy conditions, but these are beyond the scope of this application note. Consider a simple method of determining the approximate pulse-width of an input signal corrupted by a lot of noise.

Consider the signal of Figure 4. Is this one noisy 5 ms pulse or a number of smaller pulses? Taken at face value, this would translate into a number of various length disjoint pulses. However, if this were part of a pulse-width modulated code that had been transmitted on an r.f. carrier, the range of reception of the pulse could be significantly improved if the intelligence could be unambiguously extracted from this waveform. Much of the success of decoding algorithms depends on the knowledge of the expected signal. If, for example, the above waveform is expected to be either a 6 ms pulse or a 2 ms pulse, it is expected that this algorithm would more often choose the former. If there were some independent cross check on the validity of the code detection, such as a cyclic redundancy check, the detection could be made with a fair degree of certainty.

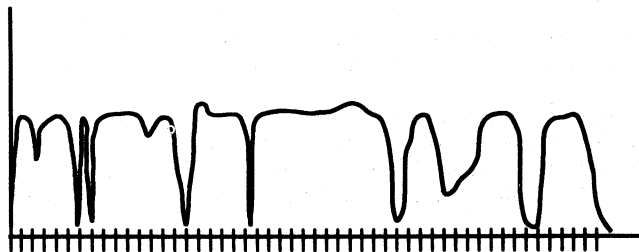


Figure 4. Noisy Pulse

It is beyond the scope of this note to present a detailed discussion of pulse train encoding and decoding, but the following paragraphs offer a few ideas about developing an effective method for capturing noisy pulses with an MCU.

The detection of the above signal with any of the earlier methods is unlikely to yield the correct data. With the MC68HC11A8 pulse accumulator, the pulse can be determined to be more likely 6 ms than 2 ms, but without the pulse accumulator, the MC68HC05 MCUs require more software intensive methods.

The basic strategy used to measure the pulse is to take periodic samples of the signal and employ some heuristic process to discover the signal in the noise. Most commonly, the selected algorithm is simple voting. Additionally, some cross check of the data such as a check-sum may be employed. If, for example, a 100 μ s sample of the above pulse is taken, marked by the tick marks on the drawing, the findings show that the signal is high for 37 to 50 samples. This is more consistent with a 'wide' pulse than a 'narrow' one. If a cross check agrees with this conclusion, there is some confidence in the conclusion. If the cross check disagrees, the error could be guessed based on the lowest probability detection; or a re-transmission might be required. If no cross check is possible, a decision can be made on a minimum probability required to accept the data.

Below is a sampling routine that uses the output compare interrupt to time the samples. Fifty samples are accumulated before testing the vote.

```

TOCI          BRCLR          TOF, TSR, NO_TOC  CLEAR FLAG
              LDX            OCHR
              LDA            OCLR
              ADD            #50                INT IN 100
              BCC            SMP1                ..  $\mu$ s
              INCX
SMP1          STX            OCHR
              STA            OCLR
*
*              NEXT SAMPLE IN 100  $\mu$ s
*
              BRCLR          PIN, PORT, SMP2
              INC            VOTE
              BRA            SMP3
SMP2          DEC            VOTE
              DEC            COUNT
              BNE            SMP9
*
* HERE AFTER 50 SAMPLES
* PUT VOTING ROUTINE HERE
*
NO_TOC:
*
* DO OTHER INTERRUPT HERE
*
              RTI

```

Note that this interrupt routine consumes only about 25 – 30% of the processor cycles. This number is directly related to the sample rate — sampling of 1/2 the rate reduces usage to less than 15% of the processor.

The choice of voting algorithm is application dependent. However, synchronization of the signal must also be considered. Depending on the type of coding used, a signal can be assumed to be self-synced. That is, the measurement of any pulse after a quiet period causes the receiving processor to try to wake up to a wide pulse or a narrow pulse. This causes the voting algorithm to reject pulses that vary widely from one of the expected widths.

With crystal-controlled oscillators in both the transmitting processor and the receiving one, this does not present a problem. If one or both of the controller clocks is not tightly regulated, however, the receiver will require time base as well as start time synchronization. In general, the more information that must be transmitted, the greater potential for error due to noise. The information transmitted is the code, the start time, and the time base.

Summary of Pulse Detection

MCU systems often read information from a hardware device by means of timed pulses. When these pulses fall in the range of a few tens of milliseconds, most MCUs can measure the pulse width easily with a high degree of accuracy. When the pulses are very short, very long, or noisy, the accurate detection and measurement of them is more difficult.

The most important decision to be made in system design for pulse measurement is the choice of MCU, specifically the timer subsystem. The least sophisticated timers such as found on the MC68HC05J1 lose some resolution and accuracy, particularly for short pulses, but these simple timers are often found on the low-cost chips. As the complexity and cost of the timer is increased, so is the performance of the MCU in this task. The very complex timer system in the MC68332 provides the greatest resolution and performance of any MCU available. For information, call your local Motorola sales office.

APPENDIX A

TTL LONG PULSE GENERATION

```

*
*
*           TIC-TOC ROUTINES FOR 68HC05CX
*
*   WRITTEN 11/11/89   BY MIKE PAUWELS
*
*
* PULSE GENERATION
*
* THIS ROUTINE GENERATES PULSES FROM A MC68HC05CX MICROCONTROLLER USING
* THE TIMER OUTPUT COMPARE FUNCTION.  THE LENGTH OF PULSES GENERATED
* RANGE FROM A FEW MICROSECONDS TO MORE THAN TWO HOURS.
*
* THIS SOFTWARE IS INTENDED AS A SUBSYSTEM TO BE INCLUDED IN A LARGER
* PROGRAM.  ETC.
*
* CONSTANTS:
* SYSTEM CONSTANTS:
* ADDRESSES:
    OPT      EQU      no1
PORTA      EQU      0           PORT A
DDRA      EQU      4           DATA DIRECTION REGISTER FOR PORT A
TCR       EQU      $12        TIMER CONTROL REGISTER
ICIE      EQU      7           INPUT CAPTURE INTERRUPT ENABLE
OCIE      EQU      6           OUTPUT COMPARE INTERRUPT ENABLE
TOIE      EQU      5           TIMER OVERFLOW INTERRUPT ENABLE
IEDG      EQU      1           INPUT EDGE
OLVL      EQU      0           OUTPUT LEVEL
TSR       EQU      $13        TIMER STATUS REGISTER
ICF       EQU      7           INPUT CAPTURE FLAG
OCF       EQU      6           OUTPUT COMPARE FLAG
TOF       EQU      5           TIMER OVERFLOW FLAG
ICHR      EQU      $14        INPUT CAPTURE REGISTER HIGH BYTE
ICLR      EQU      $15        INPUT CAPTURE REGISTER LOW BYTE
OCHR      EQU      $16        OUTPUT COMPARE REGISTER HIGH BYTE
OCLR      EQU      $17        OUTPUT COMPARE REGISTER LOW BYTE
CHR       EQU      $18        TIMER/COUNTER HIGH BYTE
CLR       EQU      $19        TIMER/COUNTER LOW BYTE
ACHR      EQU      $1A        ALTERNATE TIMER/COUNTER HIGH BYTE
ACLR      EQU      $1B        ALTERNATE TIMER/COUNTER LOW BYTE
    OPT      EQU      1
* PROGRAM CONSTANTS
    ORG      EQU      $20
DELAY     FCB      6           DELAY FOR START OF PULSE
MIN_PLS   FCB      5           MINIMUM PULSE WIDTH IN CLOCK COUNTS
DO_PLS    FCB      $01,$C9,$C3,$80
* VARIABLES
    ORG      EQU      $BA           OR CONCATENATE WITH USER MEMORY
PULSE     RMB      4           MAX TIME = 143.1655765 MINUTES!
*

```

```

* ASSUMING A 4 MHZ CRYSTAL, FOUR BYTES WILL AUTOMATICALLY TIME
* 2^33 MICROSECONDS (ABOUT 2.4 HOURS) WITHIN THE ACCURACY OF THE
* CRYSTAL. EACH BIT IS 2 MICROSECONDS. FOR LONG TIME PERIODS,
* CONSIDER THAT A SLOWER CLOCK WILL SAVE POWER AND A 32KHZ WATCH
* CRYSTAL IS INEXPENSIVE, BUT REMEMBER THAT THE PROCESSOR EXECUTION
* WILL SLOW BY 122 TIMES! IF YOU HAVE A LOT OF PROCESSING TO DO
* BETWEEN UPDATES, YOU MAY FIND THE PROCESSOR TOO SLOW!
*
* SOME OTHER TIME OPTIONS:
*     5 BYTES WILL TIME UP TO 25.45 DAYS
*     6 BYTES WILL TIME UP TO 17.83 YEARS
*     7 BYTES WILL TIME 4,566 YEARS!
*
* NO RESET INITIALIZATION IS REQUIRED. THE TIMED PULSE WILL BE
* DRIVEN ON THE TCMP PIN WHICH IS AUTOMATICALLY INITIALIZED AS
* AN OUTPUT. THE TIMER OUTPUT COMPARE AND THE TIMER OVERFLOW
* INTERRUPTS ARE INITIALIZED BY THE START PULSE SUBROUTINE (STRT_PLS).
*
FLAGS    RMB    1        STORE A FLAG
FIRE     EQU    7        INDICATES PULSE HAS STARTED
LAST     EQU    6        INDICATES LAST INTERRUPT HAS OCCURED
*
* MAIN PROGRAM GOES HERE. THE LENGTH OF THE DESIRED PULSE IS
* DETERMINED AND STORED IN 'PULSE' AT 2 MICROSECONDS PER BIT.
* THE PULSE WILL START AFTER 'STRT_PLS' IS CALLED WITH THE
* LATENCY AND ACCURACY NOTED BELOW.

```

```

*****
*                               RESET ROUTINE                               *
*****

```

```

RST_INT:  ORG      $100
          CLR      TCR
          CLR      FLAGS          RESET ALL FLAGS
          LDA      #$FF
          STA      DDRA
          LDA      #$02
          STA      PORTA

```

```

*****
*                               MAIN PROGRAM                               *
*****

```

```

*
* HERE IS THE MAIN LOOP. IF WE HAVEN'T FIRED, CALL STRT_PLS
*
MAIN:
          BRSET    FIRE, FLAGS, FIRED
*
* HERE ONCE AFTER RESET WHILE FIRE FLAG IS CLEARED
*
LOAD     LDX      #3            LOAD FOUR BYTES
          LDA      DO_PLS, X
          STA      PULSE, X

```

```

        DECX
        BPL      LOAD
        JSR      STRT_PLS
*
* DURING THE INTERRUPTS, THE 'LAST' FLAG IS CLEAR, JUMP TO MAIN
*
FIRED   BRCLR   LAST,FLAGS,MAIN
*
* HERE AFTER THE INTERRUPTS
*
        NOP                REPRESENTS OTHER INSTRUCTIONS
        BRA      MAIN

*****
*          START PULSE SUBROUTINE          *
*****
*
* CALL THIS ROUTINE WITH THE DESIRED PULSE LENGTH IN 'PULSE'.
* THE MOST SIGNIFICANT BYTE IS STORED FIRST. FOR LONG PULSES,
* THE 'FRACTIONAL' PART, THAT STORED IN THE TWO LEAST SIGNIFICANT
* BYTES, ARE TIMED FIRST. THEN THE EXTENSIONS ARE TIMED OUT ONE
* AT A TIME UNTIL, ON THE LAST PERIOD THE OUTPUT LEVEL BIT IS
* CLEARED AND THE PULSE STOPS AUTOMATICALLY.
*
* NOTE THAT THE VARIABLE PULSE IS MODIFIED BY THE PULSE GENERATION
* FUNCTION, AND THAT THAT VARIABLE REFLECTS (ROUGHLY) THE AMOUNT
* OF PULSE REMAINING. OVERWRITING THE PULSE WIDTH CAN HAVE
* UNDESIREABLE RESULTS, BUT SHOULD USUALLY RESULT IN CHANGING THE
* TERMINATION TIME.
*
* PROCEDURE START_PULSE (PULSE_WIDTH: LONG_INT);

STRT_PLS:
        SEI      DON'T INTERRUPT
* IF PULSE_WIDTH > $FFFF THEN INTERRUPT:=ENABLE ELSE INTERRUPT:=DISABLE
        BSET    7,PORTA          TURNS ON INDICATOR LED, NOT TRUE PULSE

        BSET    OCIE,TCR ENABLE TOC INTERRUPT
        LDA     PULSE+1
        SUB     #1
        STA     PULSE+1
        LDA     PULSE
        SBC     #0
        STA     PULSE
        BCC     SP1
*
* HERE IF PULSE WAS LESS THAN $FFFF--FIX THE DAMAGE
*
        CLR     PULSE+1
        CLR     PULSE
        BCLR    OCIE,TCR
*
* IF 0 < PULSE_WIDTH < MINIMUM THEN PULSE_WIDTH := MINIMUM;
*
SP1     TST     PULSE+2

```

```

        BNE     LONG_PLS
        LDA     #PULSE+3
        BEQ     LONG_PLS
        CMP     MIN_PLS
        BHI     LONG_PLS
        LDA     MIN_PLS
        STA     PULSE+3

LONG_PLS:
* HERE WHEN THE PULSE WIDTH FRACTIONAL PART IS ZERO OR >= MIN_PLS
*
* FIRST START THE PULSE
*
* NEXT LEVEL := TRUE;
        BSET    OLVL,TCR

*
* ONE OF THE TRICKIEST OPERATIONS IS TURNING ON THE PULSE. SINCE
* THE 'HC05 DOES NOT HAVE THE FACILITY TO SWITCH THE TCMPLINE
* DIRECTLY, WE SETUP A TURN ON TO OCCUR IMMEDIATELY. WE HAVE TO
* ADJUST TO THE TIME NEEDED FOR THE SETUP. THIS IS THE VALUE 'DELAY'.
*
* OUTPUT_COMPARE := TIMER + DELAY
        LDX     ACHR     MUST BE READ FIRST
        LDA     ACLR     TIMER = X:A
        ADD     DELAY
        BCC     MARK_1   MARK TIME
        INCX
        BRA     OC1
MARK_1  NOP             TO BALANCE EXECUTION TIMES
        BRA     OC1
OC1     STX     OCHR     INHIBITS TOC
        STA     OCLR     ENABLES TOC
*
* IF DELAY IS CORRECT, PULSE WILL TURN ON IMMEDIATELY
*
*
* TOC := TURN_ON + PULSE_WIDTH MOD $10000
*
        ADD     PULSE+3
        STA     PULSE+3
        TXA
        ADC     PULSE+2
        TAX
        LDA     PULSE+3
        CLR     PULSE+3
        CLR     PULSE+2
*
*
* IF INTERRUPT=ENABLED THEN OLVL := 1 ELSE OLVL := 0 ;...AND PULSE
* WILL TERMINATE
*
        BRSET  OCIE,TCR,OC2
        BCLR   OLVL,TCR IF INTERRUPT = DISABLED
OC2     STX     OCHR
        TST     TSR             WILL CLEAR OCF...
        STA     OCLR           ...WHEN EXECUTED

```

```

        BSET      FIRE,FLAGS      INDICATE PULSE HAS FIRED
*
* AT THIS TIME, THE MINIMUM PULSE CAN EXPIRE.  IN THAT CASE
* WHEN WE ENABLE THE INTERRUPT, WE WILL IMMEDIATELY BEGIN
* SERVICING.
*
        CLI
        RTS

```

```

*****
*                TIMER INTERRUPT ROUTINE                *
*****

```

```

TCMP_INT:
*
* WE WILL INTERRUPT WITH A TOC ONLY IF THERE ARE A WHOLE NUMBER OF
* $10000 PERIODS TO COMPLETE.  WE NEED ONLY DECREMENT THE 'INTEGER'
* PART OF THE PULSE WIDTH AND IF THIS IS THE LAST TIME, WE CLEAR
* THE INTERRUPTS AND SET THE OUTPUT LEVEL TO '0'.  THE TOC REGISTER
* IS NOT CHANGED.
*
* IF THERE ARE OTHER POSSIBLE TIMER INTERRUPT SOURCES (INPUT CAPTURE
* AND/OR TIMER OVERFLOW) THEN WE SHOULD ARBITRATE THE SOURCE AT THIS
* TIME.  NOTE THAT THERE WILL ALWAYS BE PLENTY OF TIME TO SERVICE THIS
* ROUTINE, SO THE PRIORITY COULD BE SET TO THE LOWEST LEVEL.
*
*
* ARBITRATION...
*
* IF PULSE_WIDTH > $10000 THEN
* PULSE_WIDTH := PULSE_WIDTH - $10000
*
        LDA      PORTA
        EOR      #$03
        STA      PORTA          TOGGLE 2 PORT LINES (DIAGNOSTICS)
*
        LDA      PULSE+1
        SUB      #1
        STA      PULSE+1
        LDA      PULSE
        SBC      #0
        STA      PULSE
        BCC      NOT_LAST
* ...ELSE INTERRUPT := DISABLE; OLVL := 0;
*
* HERE IF PULSE WAS ON LAST COUNT, CLEAR INTERRUPT AND OLVL
*
        CLR      PULSE+1
        CLR      PULSE
        BCLR     7,PORTA
        BCLR     OCIE,TCR
        BCLR     OLVL,TCR
        BSET     LAST,FLAGS

```

*
* HERE IF NOT ON LAST PULSE
*

* CLEAR(OCF);
*

NOT_LAST:

LDA	TSR	NECESSARY ACCESS
LDA	OCLR	... NEXT INTERRUPT WILL HAPPEN IN \$10000
RTI		

* DUMMY INTERRUPT ROUTINES *

SPI_INT RTI
SCI_INT RTI
IRQ_INT RTI
SWI_INT RTI

* INTERRUPT VECTORS *

	ORG	\$1FF4
SPI_VEC	FDB	SPI_INT
SCI_VEC	FDB	SCI_INT
TIM_VEC	FDB	TCMP_INT
IRQ_VEC	FDB	IRQ_INT
SWI_VEC	FDB	SWI_INT
RST_VEC	FDB	RST_INT

* ASSUMING A 4 MHZ CRYSTAL, TWO BYTES CAN ACCUMULATE UP TO
 * 2^33 MICROSECONDS (ABOUT 2.4 HOURS) WITHIN THE ACCURACY OF THE
 * CRYSTAL. EACH BIT IS 2 MICROSECONDS. FOR LONG TIME PERIODS,
 * CONSIDER A SLOWER CLOCK.

* SOME OTHER TIME OPTIONS:

* 3 BYTES WILL TIME UP TO 25.45 DAYS
 * 4 BYTES WILL TIME UP TO 17.83 YEARS
 * 5 BYTES WILL TIME 4,566 YEARS!

FLAGS	RMB	1	BOOLEAN VARIABLES
ARM	EQU	7	SET WHEN PROCESSOR IS READY
GOT	EQU	6	SET WHEN PULSE IS CAPTURED
	ORG	\$100	

 * RESET INTERRUPT ROUTINE *

RST_INT:

* NO RESET INITIALIZATION IS REQUIRED. TO MEASURE A PULSE INCIDENT
 * ON THE INPUT CAPTURE PIN, ARM THE PROCEDURE BY CALLING 'GET_PLS'.
 * AFTER THE PULSE IS TERMINATED, ADDITIONAL USER CODE (E.G. TO SET
 * A FLAG) CAN BE ADDED AS INDICATED IN THE INTERRUPT ROUTINE. NOTE
 * THAT THIS FUNCTION REQUIRES THE INTERRUPT STRUCTURE TO SERVICE
 * TIMER OVERFLOWS AND FINAL PULSE TERMINATION. THIS IS NOT ESSENTIAL
 * AND THE INTERRUPT STRUCTURE COULD BE REPLACED BY POLLING IN THE
 * USER'S MAIN LOOP, AS LONG AS THE POLLING PERIOD WAS LESS THAN THE
 * OVERFLOW TIME OF THE COUNTER/TIMER.

```

BSET 7,DDRA
BSET 7,PORTA
CLR  FLAGS
  
```

* DO OTHER INIT STUFF. THE FOLLOWING DELAY REPRESENTS OTHER CODE,
 * AND GIVES THE LED A MOMENTARY FLASH.

```

LDA  #100
JSR  DELAY          FOR 1 SECOND
  
```

* CONTINUE

 * MAIN LOOP *

MAIN:

```

BRSET ARM, FLAGS, ARMED
BSR  GET_PLS
ARMED  NOP
BRSET GOT, FLAGS, GOT_IT
NOP
GOT_IT  NOP
BRA  MAIN
  
```

```
*****
*                               ARM CAPTURE SUBROUTINE                               *
*****
```

```
*
* CALL THIS ROUTINE TO ARM THE PULSE MEASUREMENT. NOTE THAT THE
* LENGTH OF PULSE THAT CAN BE MEASURED IS LIMITED BY SIZE OF THE
* OVERFLOW ACCUMULATOR. POSITIVE GOING PULSE IS ASSUMED; THE
* MODIFICATIONS FOR NEGATIVE GOING PULSE ARE SIMPLY THE INVERSION
* OF THE IEDG. SYSTEM IS ARMED 22 MICROCYCLES AFTER THE ROUTINE
* IS CALLED.
```

```
*
GET_PLS:
      BSET      IEDG,TCR
      LDA       TSR                TWO STEPS REQUIRED...
      LDA       ICLR                ...TO CLEAR OLD FLAGS
      BSET      ICIE,TCR
      BSET      TOIE,TCR START COUNTING OVERFLOWS
      BCLR      7,PORTA
      BSET      ARM,FLAGS
      CLI
      RTS
```

```
*****
*                               TIME DELAY SUBROUTINE                               *
*****
```

```
* CALLED FOR A BUSY DELAY. IF NOT INTERRUPTED, WILL RETURN AFTER
* A DELAY OF 5 MILLISECONDS TIMES THE CONTENTS OF 'A' ACCUMULATOR.
*
```

```
DELAY:
      LDX       #249
DLA1  DECK
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      BNE      DLA1
      DECA
      BNE      DELAY
      RTS
```

```
*****
*                               TIMER INTERRUPT ROUTINE                               *
*****
```

```
* HERE ON TIMER INTERRUPT. WE ASSUME THAT TIMER OUTPUT ROUTINES
* DO NOT HAVE TO BE ARBITRATED. IF TOC IS NEEDED, THE ARBITRATION
* MUST BE CALCULATED. SINCE THE ONLY STICKY PROBLEM OCCURS ON
* SIMULTANEOUS OR NEAR-SIMULTANEOUS INTERRUPTS, THE TIMING OF THIS
```

```

* ROUTINE IS CAREFULLY CALCULATED.
*
TIM_INT:
*       THE FOLLOWING INSTRUCTION IS NEEDED IF ANY OTHER TIMER
*       INTERRUPTS ARE ENABLED:
*BRCLR  ICF,TSR,NO_TIC   BR IF NO INPUT CAPTURE
*
* HERE ON INPUT CAPTURE.  IS THIS FIRST EDGE OR LAST EDGE?
*
      BRCLR  IEDG,TCR, LAST_EDG
      BCLR   IEDG,TCR PREPARE FOR TRAILING EDGE
*
* HERE ON THE FIRST (RISING) EDGE
*
      LDA    ICHR
      LDX    ICLR          <<<< point A
      STA    START_T  START TIME HIGH BYTE
      STX    START_T+1  "      "  LOW      "
*
* WE NOW HAVE THE CAPTURED START TIME IN MEMORY.
*
      CLI
      RTI
*
* HERE ON THE TRAILING EDGE OF THE MEASURED PULSE.  THE TIC REGISTER
* HAS THE TWO LEAST SIGNIFICANT BYTES OF THE STOP TIME.  SUBTRACT
* THE START TIME; IF NECESSARY BORROW FROM THE AC_OVFL.  NO CHECK IS
* MADE FOR OVERFLOW OF THE MAXIMUM PULSE.
*
LAST_EDG:
      BSET   GOT,FLAGS
      BSET   7,PORTA
      BCLR   ICIE,TCR
      LDX    ICHR
      LDA    ICLR
      STX    PULSE_W
      STA    PULSE_W+1
*
* HERE THE PROBLEM IS TOO MANY OVERFLOWS.  IF ICHR = $FF AND ACHR = 0
* AND THE OVERFLOW FLAG HAS BEEN CLEARED, WE ACCUMULATED ONE TOO
* MANY OVERFLOW.
*
      INCA          TEST FOR = $FF
      BNE    CALC_PW
      TST    ACHR
      BNE    CLEAR_A1
      BRSET  TOF,TSR,CLEAR_A1
      LDA    AC_OVFL+1
      SUB    #1
      STA    AC_OVFL+1
      BCC    CLEAR_A1
      DEC    AC_OVFL
CLEAR_A1:
      LDA    ACLR          TO CLEAR LATCH
CALC_PW:
      LDA    PULSE_W+1

```

```

SUB     START_T+1
STA     PULSE_W+1
LDA     PULSE_W
SBC     START_T
STA     PULSE_W
LDA     AC_OVFL+1
SBC     #0
STA     AC_OVFL+1
BCC     TIM_EXIT
DEC     AC_OVFL

```

NO_TIC:

```

*
*  COULD BE A TOC.  OTHER TIC OR TOC OR OVERFLOW STUFF CAN BE DONE HERE
*
*

```

```

TIM_EXIT:
    RTI

```

```

*****
*                DUMMY INTERRUPT ROUTINES                *
*****
SPI_INT RTI
SCI_INT RTI
IRQ_INT RTI
SWI_INT RTI

```

```

*****
*                INTERRUPT VECTORS                        *
*****

```

```

        ORG     $1FF4
* Interrupt Vectors
SPI_VEC FDB     SPI_INT
SCI_VEC FDB     SCI_INT
TIM_VEC FDB     TIM_INT
IRQ_VEC FDB     IRQ_INT
SWI_VEC FDB     SWI_INT
RST_VEC FDB     RST_INT

```


Low Skew Clock Drivers and their System Design Considerations

Prepared by Chris Hanke, CMOS Design Engineer
Gary Tharalson, CMOS/TTL Product Planning Manager

ABSTRACT

Several varieties of clock drivers with 1 ns or less skew from output-to-output are available from Motorola. Microprocessor-based systems are now running at 33 MHz and beyond, and system clock distribution at these frequencies mandate the use of low skew clock drivers. Unfortunately, just plugging a high performance clock driver into a system does not guarantee trouble free operation. Only careful board layout and consideration of system noise issues can guarantee reliable clock distribution. This application note addresses these system design issues to help ensure that Motorola's low skew clock drivers are used effectively in a system environment.

INTRODUCTION

With frequencies regularly reaching 33 MHz and approaching 40–50 MHz in today's CISC and RISC microprocessor systems, well controlled and precise clock signals are required to maintain a synchronous system. Many microprocessors also require input clock duty cycles very close to 50%. These stringent timing requirements mandate the use of specially designed, low skew clock distribution circuits or 'clock drivers.' However, just plugging one of these parts into your board does not ensure a trouble free system. Careful system and board design techniques must be used in conjunction with a low skew clock driver to meet system timing requirements and provide clean clock signals.

Why are Low Skew Clock Drivers Necessary

An MPU system designer wants to utilize as much of a clock cycle as possible without adding unnecessary timing guardbands. Propagation delays of peripheral logic do not scale with frequency. Therefore, as the clock period decreases, the system designer has less time but the same logic delays to accomplish the function. How can he get more time? A viable option is to use a special clock source that minimizes clock 'uncertainty.'

A simple example illustrates this concept. At 33 MHz, $T_{\text{cycle}} = 30$ ns. An FCT240A, for example, has a High-Low uncertainty of the min/max spread of t_{PLH} to t_{PHL} of approximately 3.3 ns. If 1.7 ns of pin-to-pin skew due to the actual part and PCB trace delays is also considered, then only 25 ns of the clock period is still available. The worst case t_{p} of clock-to-data valid on the 88200 M-Bus is 12 ns, which leaves only 13 ns to accomplish additional functions. In this case

17% of a cycle is required for clock distribution or clock 'uncertainty,' which is an unacceptable penalty from a system designer's point of view. At 50 MHz this penalty becomes 25%. A maximum of 10% of the period allotted for clock distribution is an acceptable standard.

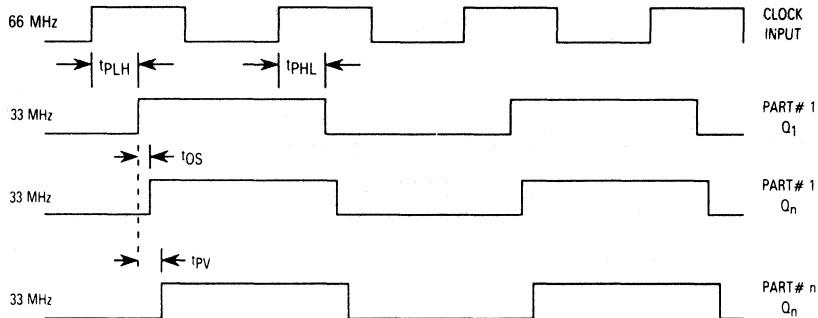
If multiple levels of clock distribution (one clock driver's output feeding the inputs of several other clock drivers) are necessary due to large clock fan-outs, the additional part-to-part skew variations add even more to the clock uncertainty. Standard logic has always been specified with a large (and conservative) delta between the minimum and maximum propagation delays. This delta creates the excessive amount of clock 'uncertainty' which the system designer has been forced to design into his system, even though it is not realistic. When system frequencies were below 16 MHz this large clock penalty could be tolerated, but as the above example points out, not anymore. A clock driver's specs *guarantee* this min/max delta to be a specific, small value. To reduce the clock overhead to manageable levels, a clock driver with minimal variation (<5%) from a 50% duty cycle and guaranteed low output-to-output and part-to-part skew must be used.

DEFINITIONS

A typical clock driver has a single input which is usually driven by a crystal oscillator. The clock driver can have any number of outputs which have a certain frequency relationship to the clock input. Clock driver skew is typically defined by three different specs. These specs are graphically illustrated in Figure 1.

The first spec, t_{OS} , measures the difference between the fastest and slowest propagation delays (any transition) between the outputs of a single part. This number must be 1 ns or less for high-end systems.

The second, t_{PS} , measures the difference between the high-to-low and low-to-high transition for a single output (pin). This spec defines how close to a 50% duty cycle the outputs of the clock driver will be. For example, if this spec is 1 ns (± 0.5 ns), at 33 MHz the output duty cycle is $50\% \pm 3.5\%$. A clock driver which only buffers the crystal input, creating a 1:1 input to output frequency relationship, can be a problem if a very tight tolerance to a 50% duty cycle is required. In this situation the output duty cycle is directly dependent on the input duty cycle, which is not well controlled in most crystal oscillators. The clock driver's outputs switching at half the input frequency ($\div 2$) is a common relationship, which means



- Notes: 1) t_{PS} measures $t_{PLH} - t_{PHL}$ for any single output on a part.
 2) t_{OS} measures the maximum difference between any t_{PHL} or t_{PLH} between any output on a single part.
 3) t_{PV} measures the maximum difference between any t_{PHL} or t_{PLH} between any output on any part.

Figure 1. Timing Diagram Depicting Clock Skew Specs Within One Part and Between any Two Parts

that the outputs switch on only one edge of the oscillator, eliminating the output's dependence on the duty cycle of the input (crystal oscillator frequency is very stable).

The third spec, t_{PV} , measures the maximum propagation delay delta between any given pin on any part. This spec defines the part to part variation between any clock driver (of the same device type) which is ever shipped. This number reflects the process variation inherent in any technology. For CMOS, this spec is usually 3 ns or less. High performance ECL technologies can bring this number down into the 1–2 ns range. Another way to minimize the part-to-part variation is to use a phase-locked loop clock driver, which are just now becoming available.

An important consideration when designing a clock driver into a system is that the skew specs described above are usually specified at a fixed, lumped capacitive load. In a real system environment the clock lines usually have various loads distributed over several inches of PCB trace which can contribute additional delay and sometimes act like transmission lines, so the system designer must use careful board layout techniques to minimize the total system skew. In other words, just plugging a low skew clock driver into a board will not solve all your timing problems.

DESIGN CONSIDERATIONS

Figure 2 is a scale replication of a section of an actual 88000 RISC system board layout. The section shown in the figure includes the MC88100 MPU and the MC88200 CMMU devices and the MC88914 CMOS clock driver. The only PCB traces shown are the clock output traces from the MC88914 to the various loads. For this clock driver the output-to-output skew (t_{OS}) is guaranteed to be less than 1 ns at any given temperature, supply voltage, and fixed load up to 50 pF.

In calculating the total system skew, the difference in clock PCB trace length and loading must be taken into account. For an unloaded PCB trace, the signal delay per unit length,

t_{pd} , is dependent only on the dielectric constant, ϵ_r , of the board material. The characteristic impedance, Z_0 , of the line is dependent upon ϵ_r and the geometry of the trace. These relationships are depicted in Figure 3 for a microstrip line.¹ The formulas for t_{pd} and Z_0 are slightly different for other types of strip lines, but for simplicity's sake all calculations in this article will assume a microstrip line.

The equations in Figure 3 are valid only for an unloaded trace; loading down a line will increase its delay and lower its impedance. The signal propagation delay (t_{pd}') and characteristic impedance (Z_0') due to a loaded trace are calculated by the following formulas:

$$t_{pd}' = t_{pd} \sqrt{1 + \frac{C_d}{C_0}}$$

$$Z_0' = \frac{Z_0}{\sqrt{1 + \frac{C_d}{C_0}}}$$

C_d is the distributed load capacitance per unit length, which is the total input capacitance of the receiving devices divided by the length of the trace. C_0 is the intrinsic capacitance of the trace, which is defined as:

$$C_0 = \frac{t_{pd}}{Z_0}$$

Assuming typical microstrip dimensions and characteristics as $w = 0.01$ in., $t = 0.002$ in., $h = 0.012$ in., and $\epsilon_r = 4.7$, the equations of Figure 3 yield $Z_0 = 69.4 \Omega$ and $t_{pd} = 0.144$ ns/in. C_0 is then calculated as 2.075 pF/in. If it is assumed that an MC88100 or 88200 clock input load is 15 pF, and that two of these loads, in addition to a 7 pF FAST TTL load, are distributed along a 9.6 in. clock trace, $C_d = (2 \times 15 + 7)$ pF/9.6 in. = 3.85 pF/in. The loaded trace propagation delay and characteristic impedance are then calculated as $t_{pd}' = 0.243$ ns/in. and $Z_0' = 41 \Omega$.

Looking at trace c in Figure 2, the two MC88200's are approximately 3 inches apart. Using the calculated value of

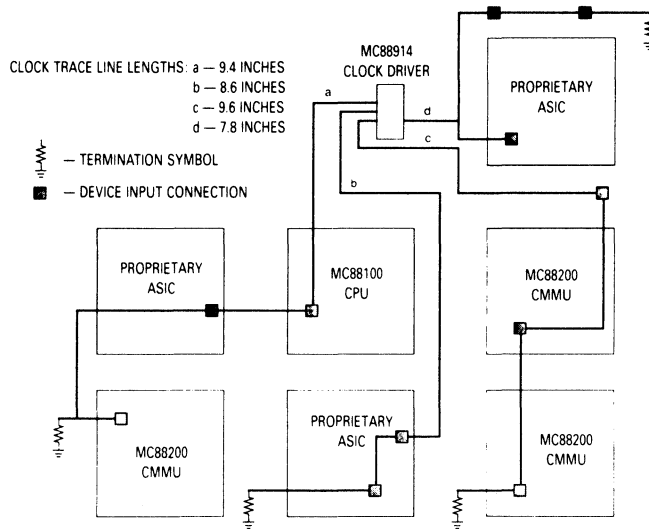


Figure 2. Scale Representation of an Actual 88000 System PCB Layout (only sections of the board related to the clock driver outputs are shown).

t_{pd} , the clock signal skew due to the trace is about 0.7 ns. Since these two devices are on the same trace, this is the total clock skew between these devices. Upon careful inspection of all the clock traces, it can be seen that clock signal skew was accounted for and minimized on this board layout. The longest distance between any 88K devices on a single clock trace is about 4.5 inches, which translates to approximately 1.1 ns of skew. The two 88K devices farthest away from the clock driver (traces a and c), are located at almost

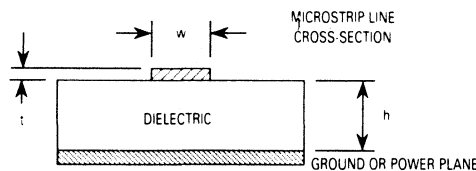
exactly the same distance along their respective traces, making the clock skew between them the 1 ns guaranteed from output to output of the clock driver. This means that the worst case clock skew between any two devices on this board is approximately 2.1 ns, which at 33 MHz is 7% of the period. Without careful attention to matching the clock traces on the board, this number could easily exceed 3 ns and the 10% cut-off point, even if a low skew clock driver is used.

CLOCK SIGNAL TERMINATIONS

Transmission line effects occur when a large mismatch is present between the characteristic impedance of the line and the input or output impedances of the receiving or driving device. The basic guidelines used to determine if a PCB trace needs to be examined for transmission line effects is that if the smaller of the driving device's rise or fall time is less than three times the propagation delay of a switching wave through a trace, the transmission line effects will be present.² This relationship can be stated in equation form as:³

$$3 \times t_{pd} \times \text{trace length} \leq t_{RISE} \text{ or } t_{FALL}$$

For the MC88914 CMOS clock driver described in this article, rise and fall times are typically 1.5 ns or less (from 20% to 80% of V_{CC}). Analyzing the clock trace characteristics presented earlier for transmission line effects, $3 \times 0.243 \text{ ns/in.} \times \text{trace length} \leq 1 \text{ ns}$ (1 ns is used as 'fastest' rise or fall time). Therefore the trace length must be less than 1.5 inches for the transmission line effects to be masked by the rise and fall times.



$$Z_0 = \frac{87}{\sqrt{\epsilon_r - 1.41}} \ln \left(\frac{5.98h}{0.8w - t} \right)$$

$$t_{pd} = 1.017 \sqrt{0.475 \epsilon_r - 0.67} \text{ ns/ft.}$$

WHERE:

ϵ_r = RELATIVE DIELECTRIC CONSTANT OF THE BOARD MATERIAL
 w, h, t = DIMENSIONS INDICATED IN A MICROSTRIP DIAGRAM.

Figure 3. Formulas for the Characteristic Impedance and Propagation Delay of a Microstrip Line. (Ref. 1)

Figure 4 shows the clock signal waveform seen at the receiver end of an unterminated 0.5 inch trace and an unterminated 9 inch trace. These results were obtained using SPICE simulations, which may not be exact, but are adequate to predict trends and for comparison purposes. The 9 inch trace, which is well beyond the 1.5 inch limit where transmission line effects come into play, exhibits unacceptable switching characteristics caused by reflections going back and forth on the trace. Even the 0.5 inch line exhibits substantial overshoot and undershoot. Any unterminated line will exhibit some overshoot and undershoot at these edge rates.

Clock lines shorter than 1–1.5 inches are unrealistic on a practical board layout, therefore it is recommended that CMOS clock lines be terminated if the driver has 1–2 ns edge rates. Termination, which is used to more closely match the line to the load or source impedances, has been a fact of life in the ECL world for many years (reference 1 is an excellent source for transmission line theory and practice in ECL systems), but CMOS and TTL devices have only recently reached the speeds and edge rates which require termination. CMOS outputs further complicate the issue by driving

from rail to rail (5 V), with slew rates exceeding those of high performance ECL devices.

Since clock lines are only driven from a single location, they lend themselves to termination more easily than bus lines which are commonly driven from multiple locations. Termination of bus lines with multiple drivers is a complicated matter which will not be addressed in this article. The most common types of termination in digital systems are shown in Figure 5. Since no single termination scheme is optimal in all cases, the tradeoffs involving the use of each will be discussed, and recommendations specific to clock drivers will be made. Reference 2 is a comprehensive and practical treatment of transmission line theory and analysis of CMOS signals, and is recommended reading for those who want to gain a better understanding of transmission lines. Figure 6 shows SPICE simulated waveforms of the different termination schemes to be discussed. The driving device in the simulations was the MC88914 output buffer; in all simulations it drove a 9 inch 41 Ω transmission line. The simulations were run using typical model parameters at 25°C and $V_{CC} = 5$ V.

Series termination, depicted in Figure 5b, is recommended if the load is lumped at the end of the trace and the output impedance of the driving device is less than the loaded characteristic impedance of the trace, or when a minimum number of components is required. The main problem with series termination occurs when the driving device has different output impedance values in the low and high states, which is a problem in TTL and some CMOS devices. A well designed CMOS clock driver should have nearly equal output impedances in the high and low states, avoiding this problem. An additional advantage is that series termination does not create a DC current path, thus the V_{OL} and V_{OH} levels are not degraded. The SPICE generated waveforms of series termination in Figure 6a show that series termination effectively masks the transmission line effects exhibited in Figure 4. If each clock output is driving only one device, series termination would be recommended, but this is not a realistic case in most systems, so series termination is not generally recommended for termination of clock lines.

Parallel termination utilizes a single resistor tied to ground or V_{CC} whose value is equal to the characteristic impedance of the line. Its major disadvantage is the DC current path it creates when the driver is in the high state (if the resistor is tied to ground). This causes excessive power dissipation and V_{OH} level degradation. Since a clock driver output is always switching, the DC current draw argument loses some credibility at higher frequencies because the AC switching current becomes a major component of the overall current. Therefore the main consideration in parallel termination is how much V_{OH} degradation can be tolerated by the receiving devices. Figure 6b demonstrates that this termination technique is effective in minimizing the switching noise, but Thevenin termination has some advantages over parallel termination.

Thevenin termination utilizes one resistor tied to ground and a second tied to V_{CC} . An important consideration when using this type of termination is choosing the resistor values to avoid settling of the voltage between the high and low logic levels of the receiving device.² TTL designers commonly use a 220/330 resistor value ratio, but CMOS is a little tricky because the switch point is at $V_{CC}/2$. With a 1:1 resistor ratio a failure at the driver output would cause the line to settle at

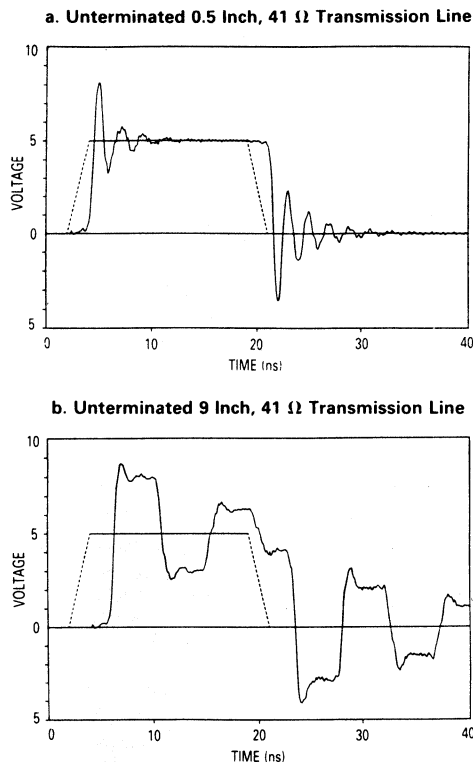


Figure 4. SPICE Simulation Results of 'Short' and 'Long' Transmission Lines. Simulations were Run with Typical Parameters (α 25°C and $V_{CC} = 5$ V).

2.5 V, causing system debug problems and also potential damage to the receiving devices.

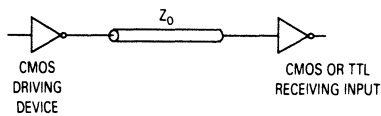
In Thevenin termination, the parallel equivalent value of the two resistors should be equal to the characteristic impedance of the line. A DC path does exist in both the high and low states, but it is not as bad as parallel termination because the resistance in the Thevenin DC path is at least 2 times greater. Figure 6c shows the termination waveforms, which exhibit characteristics similar to parallel termination, but with less V_{OH} degradation. The only real advantage of parallel over Thevenin is less resistors (1/2 as many) and less space taken up on the board by the resistors. If this is not a factor, Thevenin termination is recommended over parallel.

AC termination, shown in Figure 5e, normally utilizes a resistor and capacitor in series to ground. The capacitor blocks DC current flow, but allows the AC signal to flow to ground during switching. The RC time constant of the resistor and capacitor must be greater than twice the loaded line delay. AC termination is recommended because of its low power dissipation and also because of the availability of the resistor and capacitor in single-in-line packages (SIP). A pull-up resistor to V_{CC} is sometimes added to set the DC level at a certain point because of the failure condition described in regards to Thevenin termination. As discussed earlier, the argument of lower DC current is less convincing at high frequencies. The AC terminated waveform walks out slightly toward the end of a high-to-low or low-to-high transition as seen in Figure 6d, making it slightly less desirable than Thevenin termination.

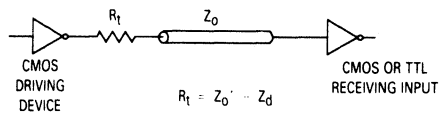
Thevenin and AC termination are the two recommended termination schemes for clock lines, but it depends on what frequency the clock is running at when making a decision between these types of termination. Although hard data is not provided to back this statement up, it is a safe assumption that at frequencies of 25 MHz and below AC is the best choice. If the system frequency could reach 40 MHz and beyond, Thevenin becomes the better choice.

Additional Considerations when Terminating Clock Lines

The results presented might imply that terminating the clock lines will completely solve noise problems, but termination can cause secondary problems with some logic devices. Termination acts to reduce the noise seen at the receiver, but that noise actually is seen as additional current and noise at the output of the driving device. If the internal and input logic on the source device is not sufficiently decoupled on chip from the high current outputs, internal threshold problems can occur. This phenomenon is commonly known as 'dynamic threshold.' It is usually evidenced by glitches appearing on the outputs of a fast, high current drive logic device as it switches high or low. This is most severe on 'ACT' devices which have high current and high slew rate CMOS outputs along with TTL inputs which have low noise immunity. This problem can be minimized by decoupling the internal ground and V_{CC} supplies on-chip and in the package. This decoupling is accomplished by having separate 'quiet' ground and V_{CC} pads on chip which supply the input circuitry's ground

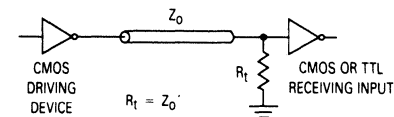


a. Transmission Line with No Termination

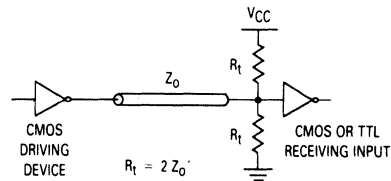


WHERE, Z_d = DRIVING DEVICE OUTPUT IMPEDANCE

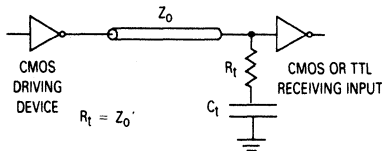
b. Transmission Line with Series Termination



c. Transmission Line with Parallel Termination



d. Transmission Line with Thevenin Termination



e. Transmission Line with AC Termination

Figure 5. Schematic Representations of Common Termination Techniques

and V_{CC} references. These pads are then tied to extra 'quiet' ground and 'quiet' V_{CC} pins on the package, or to special 'split leads' which resemble a tuning fork and utilize the lead-frame inductance to accomplish the decoupling. When choosing a clock source, make sure that the part has one of these decoupling schemes.

References

1. Blood, William R., *MECL System Design Handbook*, Motorola Inc., 1983.
2. Appl. Note AN1051, *Transmission Line Effects in PCB Applications*, Motorola Inc., 1990.
3. *Motorola FACT Data Book DL138*, Motorola Inc., 1990

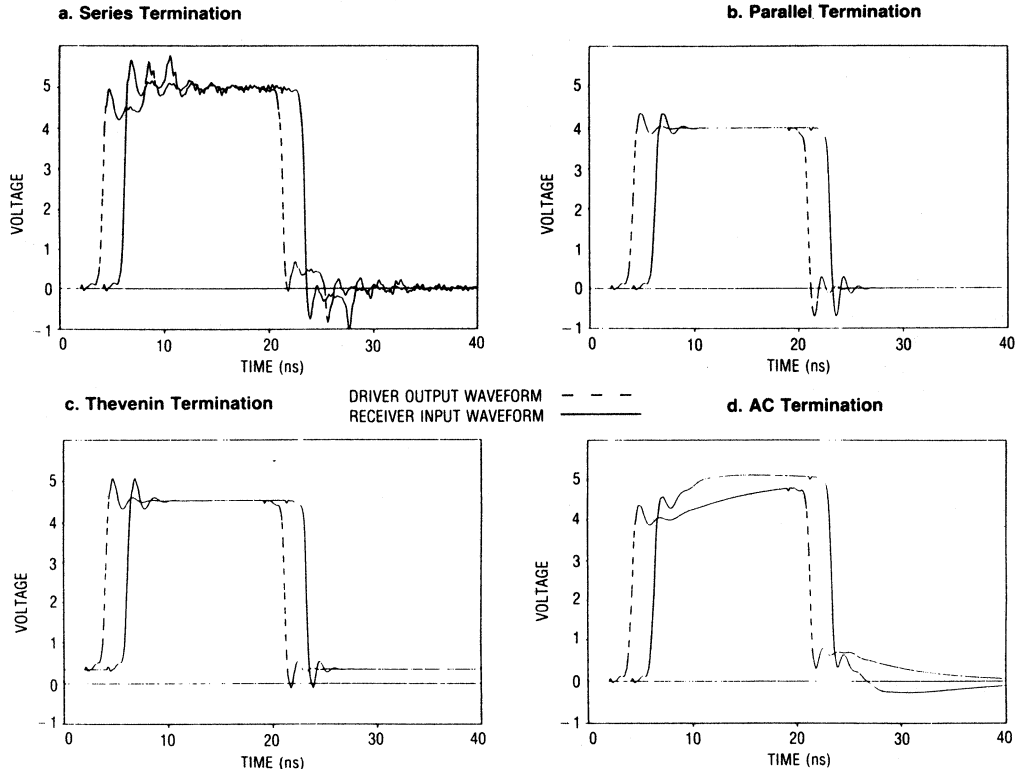


Figure 6. SPICE Simulation Results for Various Terminations of a 9 Inch, 41 Ω Transmission Line. Simulations were Run With Typical Model Parameters @ 25°C and $V_{CC} = 5.0$ V.

Calibration-Free Pressure Sensor System

Prepared by
Michel Burri, Senior System Engineer
 Geneva, Switzerland

INTRODUCTION

The MPX2000 Series of pressure transducers are semiconductor devices which give an electrical output signal proportional to the applied pressure. The sensors are a single monolithic silicon diaphragm with strain gage and thin-film resistor networks on the chip. Each chip is laser trimmed for full scale output, offset and temperature compensation.

The purpose of this document is to describe another method of measurement which should facilitate the life of the designer. The MPX2000 Series sensors are available as unported elements and as ported assemblies suitable for pressure, vacuum and differential pressure measurements in the range of 10 kPa through 200 kPa.

The use of the on-chip A/D converter of Motorola's MC68HC05B6 HCMOS MCU makes possible the design of an accurate and reliable pressure measurement system.

SYSTEM ANALYSIS

The measurement system is made up of the pressure sensor, the amplifiers and the MCU. Each element in the chain has their own device-to-device variations and temperature effects which should be analyzed separately. For instance, the 8-bit A/D converter has a quantization error of about $\pm 0.2\%$. This error should be subtracted from the maximum error specified for the system to find the available error for the rest of elements in the chain. The MPX2000 Series pressure sensors are designed to provide an output sensitivity of 4.0 mV/V excitation voltage with full-scale pressure applied or 20 mV at the excitation voltage of 5.0 Vdc.

An interesting property must be considered to define the configuration of the system, the ratiometric function of both the A/D converter and the pressure sensor device. The ratiometric function of these elements make all voltage variations from the power supply rejected by the system. With this advantage, it is possible to design a chain of amplification where the signal is conditioned in a different way.

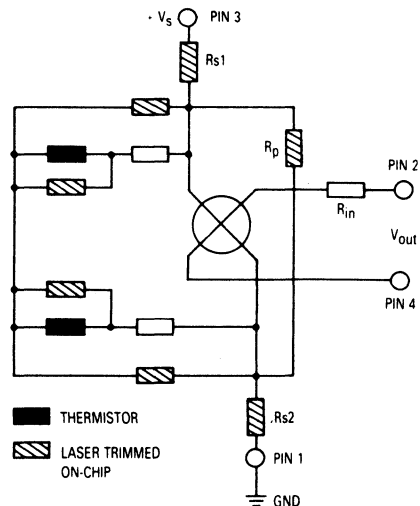


Figure 1. Seven Laser-Trimmed Resistors and Two Thermistors Calibrate the Sensor for Offset, Span, Symmetry and Temperature Compensation.

The OP-AMP configuration should have a good common-mode rejection ratio to cancel the DC component voltage of the pressure sensor element which is about half the excitation voltage value V_S . Also, the OP-AMP configuration is important when the designer's objective is to minimize the calibration procedures which cost time and money and often don't allow the unit-to-unit replacement of devices or modules.

One other aspect is that most of the applications are not affected by inaccuracy in the region 0 kPa thru 40 kPa. Therefore, the goal is to obtain an acceptable tolerance of the system from 40 kPa thru 100 kPa thus minimizing the inherent offset voltage of the pressure sensor.

PRESSURE SENSOR CHARACTERISTIC

Figure 2 shows the differential output voltage of the MPX2100 series at +25°C. The dispersion of the output voltage determines the best tolerance that the system may achieve without undertaking a calibration procedure, if any other elements or parameters in the chain do not introduce additional errors.

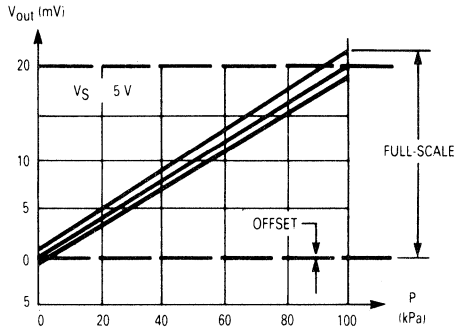


Figure 2. Spread of the Output Voltage versus the Applied Pressure at 25°C

The effects of temperature on the full scale output and offset are shown in Figure 3. It is interesting to notice that the offset variation is greater than the full scale output and both have a positive temperature coefficient respectively of $+8.0 \mu\text{V}/\text{degree}$ and $+5.0 \mu\text{V}/\text{degree}$ at 5.0 V excitation voltage. That means that the full scale variation may be compensated by modifying the gain somewhere in the chain amplifier by components arranged to produce a negative T_C of 250 PPM/°C. The dark area of Figure 3 shows the trend of the compensation which improves the full scale value over the temperature range. In the area of 40 kPa, the compensation acts in the ratio of 40/100 of the value of the offset temperature coefficient.

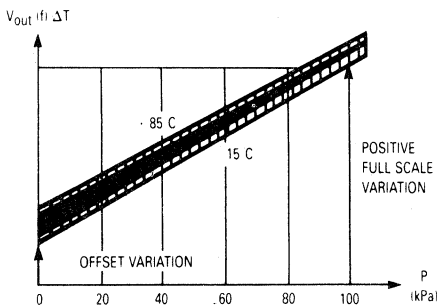


Figure 3. Output Voltage versus Temperature. The Dark Area Shows the Trend of the Compensation.

OP-AMP CHARACTERISTICS

For systems with only one power supply, the instrument amplifier configuration shown in Figure 4 is a good solution to monitor the output of a resistive transducer bridge.

The instrument amplifier does provide an excellent CMRR and a symmetrical buffered high input impedance at both non-inverting and inverting terminals. It minimizes the number of the external passive components used to set the gain of the amplifier. Also, it is easy to compensate the temperature variation of the Full Scale Output of the Pressure Sensor by implementing resistors "Rf" having a negative coefficient temperature of $-250 \text{ PPM}/\text{°C}$.

The differential-mode voltage gain of the instrument amplifier is:

$$A_{vd} = \frac{V1-V2}{Vs2-Vs4} = \left(1 + \frac{2R_f}{R_g}\right) \quad (1)$$

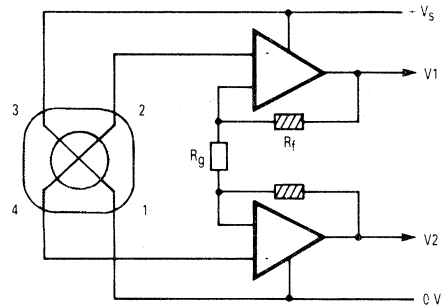


Figure 4. One Power Supply to Excite the Bridge and to Develop a Differential Output Voltage

The major source of errors introduced by the OP-AMP are offset voltages which may be positive or negative and the input bias current which develops a drop voltage ΔV through the feedback resistance R_f . When the OP-AMP input is composed of PNP transistors, the whole characteristic of the transfer function is shifted below the DC component voltage value set by the Pressure Sensor as shown in Figure 5.

The gain of the instrument amplifier is calculated carefully to avoid a saturation of the output voltage and to provide the maximum of differential output voltage available for the A/D Converter. The maximum output swing voltage of the amplifiers is also dependent on the bias current which creates a ΔV voltage on the feedback resistance R_f and on the Full Scale output voltage of the pressure sensor.

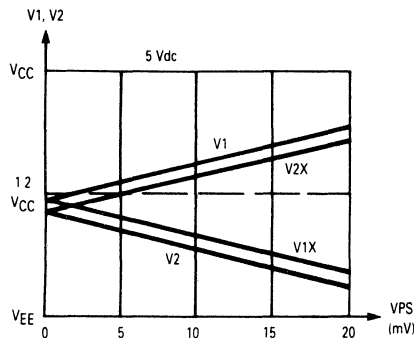


Figure 5. Instrument Amplifier Transfer Function with Spread of the Device to Device Offset Variation

Figure 5 also shows the transfer function of different instrument amplifiers used in the same application. The same sort of random errors are generated by crossing the inputs of the instrument amplifier. The spread of the differential output voltage ($V1-V2$) and ($V2x-V1x$) is due to the unsigned voltage offset and its absolute value. Figures 6 and 7 show the unit-to-unit variations of both the offset and the bias current of the dual OP-AMP MC33078.

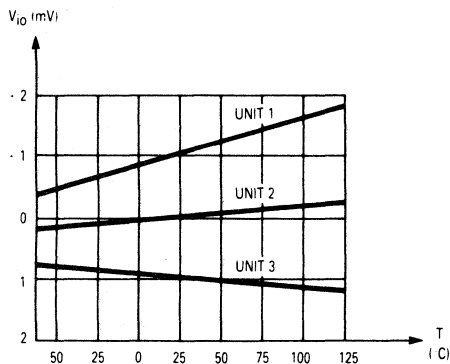


Figure 6. Input Offset Voltage versus Temperature

To realize such a system, the designer must provide a calibration procedure which is very time consuming. Some extra potentiometers must be implemented for setting both the offset and the Full Scale Output with a complex temperature compensation network circuit.

The new proposed solution will reduce or eliminate any calibration procedure.

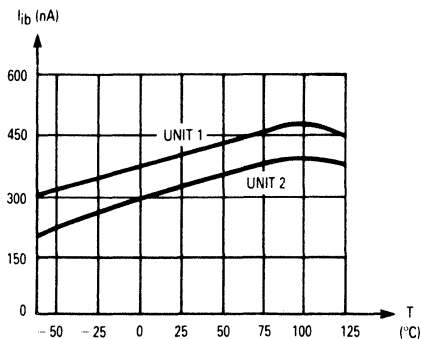


Figure 7. Input Bias Current versus Temperature

MCU CONTRIBUTION

As shown in Figure 5, crossing the instrument amplifier inputs generated their mutual differences which can be computed by the MCU.

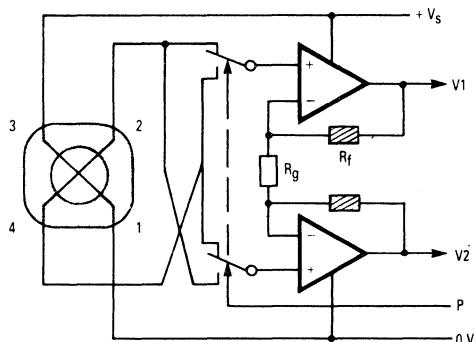


Figure 8. Crossing of the Instrument Amplifier Inputs Using a Port of the MCU

Figure 8 shows the analog switches on the front of the instrument amplifier and the total symmetry of the chain. The residual resistance $R_{DS(on)}$ of the switches does not introduce errors due to the high input impedance of the instrument amplifier.

With the aid of two analog switches, the MCU successively converts the output signals $V1$, $V2$.

Four conversions are necessary to compute the final result. First, two conversions of $V1$ and $V2$ are executed and stored in the registers $R1$, $R2$. Then, the analog switches are commuted in the opposite position and the two last conversions of $V2x$ and $V1x$ are executed and stored in the registers $R2x$ and $R1x$. Then, the MCU computes the following equation:

$$\text{RESULT} = (R1-R2) + (R2x-R1x) \quad (2)$$

The result is twice a differential conversion. As demonstrated below, all errors from the instrument amplifier are cancelled. Other averaging techniques may be used

to improve the result, but the appropriated algorithm is always determined by the maximum bandwidth of the input signal and the required accuracy of the system.

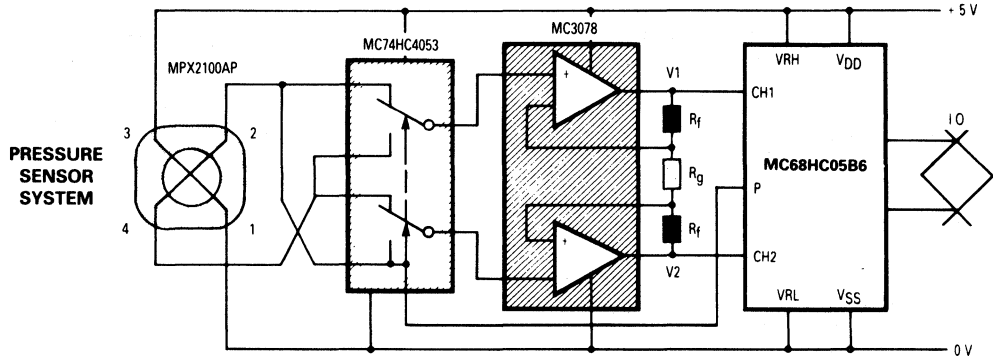


Figure 9. Two Channel Inputs and One Output Port are Used by the MCU

SYSTEM CALCULATION

$$\begin{array}{ll} \text{Sensor out 2} & \text{Sensor out 4} \\ V_{s2} = a(P) + of2 & V_{s4} = b(P) + of4 \end{array}$$

$$\begin{array}{ll} \text{Amplifier out 1} & \text{Amplifier out 2} \\ V_1 = A_{vd}(V_{s2} + OF1) & V_2 = A_{vd}(V_{s4} + OF2) \end{array}$$

$$\begin{array}{ll} \text{Inverting of the amplifier input} & \\ V_{1x} = A_{vd}(V_{s4} + OF1) & V_{2x} = A_{vd}(V_{s2} + OF2) \end{array}$$

$$\begin{array}{ll} \Delta = V_1 - V_2 & \text{1st differential result} \\ = A_{vd} * (V_{s2} + OF1) - A_{vd} * (V_{s4} + OF2) & \end{array}$$

$$\begin{array}{ll} \Delta_{2x} = V_{2x} - V_{1x} & \text{2nd differential result} \\ = A_{vd} * (V_{s2} + OF2) - A_{vd} * (V_{s4} + OF1) & \end{array}$$

Adding of the two differential results

$$\begin{aligned} V_{outV} &= \Delta + \Delta_{2x} \\ &= A_{vd} * V_{s2} + A_{vd} * V_{s2} - A_{vd} * V_{s4} - A_{vd} * V_{s4} \\ &\quad + A_{vd} * OF1 - A_{vd} * OF2 + A_{vd} * OF2 - A_{vd} * OF1 \\ &= 2 * A_{vd} * (V_{s2} - V_{s4}) \\ &= 2 * A_{vd} * [(a(P) + of2) - (b(P) + of4)] \\ &= 2 * A_{vd} * [V(P) + V_{offset}] \end{aligned}$$

There is a full cancellation of the amplifier offset OF1 and OF2. The addition of the two differential results V1-V2 and V2x-V1x produce a virtual output voltage VoutV which becomes the applied input voltage to the A/D converter. The result of the conversion is expressed in the number of counts or bits by the ratiometric formula shows below:

$$\text{count} = V_{outV} * \frac{255}{V_{RH} - V_{RL}}$$

255 is the maximum number of counts provided by the A/D converter and VRH-VRL is the reference voltage of the ratiometric A/D converter which is commonly tied to the 5.0 V supply voltage of the MCU.

When the tolerance of the full scale pressure has to be in the range of $\pm 2.5\%$, the offset of the pressure sensor may be neglected. That means the system does not require any calibration procedure.

The equation of the system transfer is then:

$$\text{count} = 2 * A_{vd} * V(P) * 51V \quad \text{where:}$$

A_{vd} is the differential-mode gain of the instrument amplifier which is calculated using the equation (1). Then with R_f = 510 k Ω and R_g = 9.1 k Ω A_{vd} = 113.

The maximum counts available in the MCU register at the Full Scale Pressure is:

$$\text{count (Full Scale)} = 2 * 113 * 0.02 V * 51 V = 230$$

knowing that the MPX2100AP pressure sensor provides 20 mV at 5.0 V excitation voltage and 100 kPa full scale pressure.

The system resolution is 100 kPa 230 that give 0.43 kPa per count.

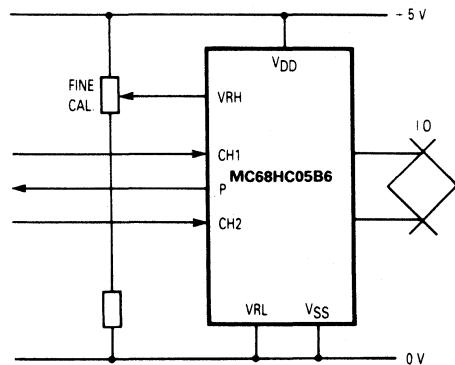


Figure 10. Full Scale Output Calibration Using the Reference Voltage VRH-VRL

When the tolerance of the system has to be in the range of $\pm 1\%$, the designer should provide only one calibration procedure which sets the Full Scale Output (counts) at

25°C 100 kPa or under the local atmospheric pressure conditions.

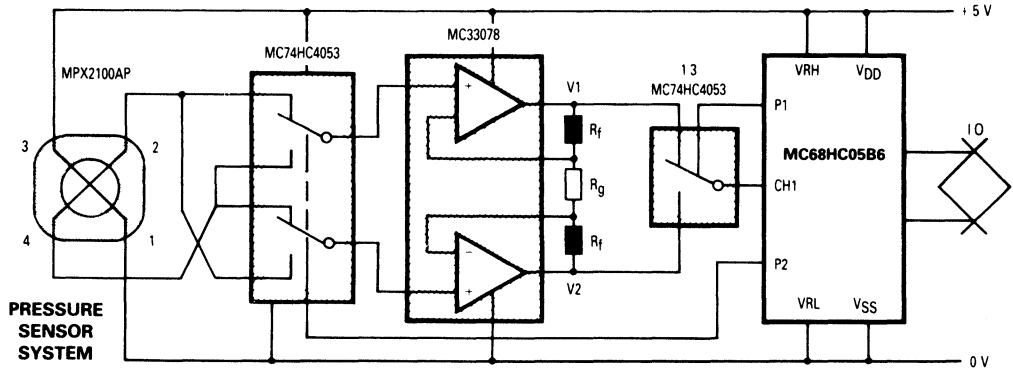


Figure 11. One Channel Input and Two Output Ports are used by the MCU

Due to the high impedance input of the A/D converter of the MC68HC05B6 MCU, another configuration may be implemented which uses only one channel input as shown in Figure 11. It is interesting to notice that practically any dual OP-AMP may be used to do the job but a global consideration must be made to optimize the total cost of the system according to the requested specification.

When the Full Scale Pressure has to be set with accuracy, the calibration procedure may be executed in different ways.

For instance, the module may be calibrated directly using Up Down push buttons.

The gain of the chain is set by changing the VRH voltage of the ratiometric A/D converter with the R/2R ladder network circuit which is directly driven by the ports of the MCU. (See Figure 12.)

Using a communication bus, the calibration procedure may be executed from a host computer. In both cases, the setting value is stored in the EEROM of the MCU.

The gain may be also set using a potentiometer in place of the resistor R_f . But, this component is expensive, taking into account that it must be stable over the temperature range at long term.

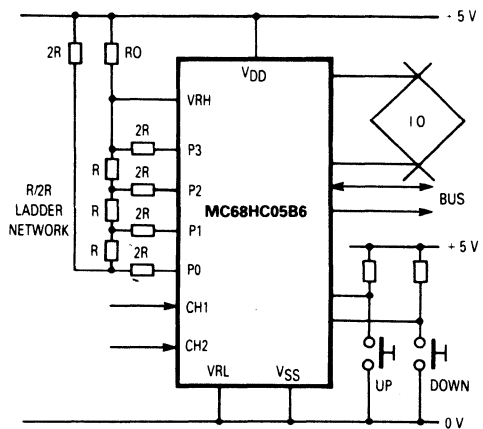


Figure 12.

PRESSURE CONVERSION TABLE

Unity	Pa	mbar	Torr	atm	at = kp/cm ²	mWS	psi
1 N m ⁻² = 1 Pascal	1	0.01	7.5 · 10 ⁻³	—	—	—	—
1 mbar	100	1	0.75	—	—	0.0102	0.014
1 Torr = 1 mmHg	133.32	1.333	.1	—	—	—	0.019
1 atm (1)	101325	1013.2	760	1	1.033	10.33	14.69
1 at = 1 kp·cm ⁻² (2)	98066.5	981	735.6	0.97	1	10	14.22
1 m of water	9806.65	98.1	73.56	0.097	0.1	1	1.422
1 lb/sq.in = 1 psi	6894.8	68.95	51.71	0.068	—	—	1

(1) Normal atmosphere (2) Technical atmosphere

Interfacing Power MOSFETs to Logic Devices

Prepared by Ken Berringer
Motorola Discrete Applications

POWER MOSFET DRIVE CHARACTERISTICS

Power MOSFETs are commonly used in switching applications due to their fast switching speeds and low static losses. When driven with sufficient gate voltage, a power MOSFET will turn on and have a very low on-resistance. If the gate voltage is insufficient to bias the Power MOSFET fully on, or excessive drain currents are applied, the power MOSFET will operate in the saturation (pinch-off) region. In other words, a certain gate voltage will support only a limited amount of drain current.

Most of the current crop of fourth generation power MOSFETs require 10 volts of gate drive to support their maximum continuous drain current. This means that 5 volt logic will not provide enough voltage to drive a standard power MOSFET. A new family of Logic Level power MOSFETs are now available that can support their rated drain current with a gate voltage of 5 volts. With the proper considerations, these power MOSFETs may be easily interfaced to most logic families.

Design of the MOSFET's gate drive is dependent on the MOSFET's input capacitance, which is strongly affected by die size. Therefore, selecting the correct device for the application not only minimizes component cost, but it also optimizes switching performance. Static, or DC, losses are determined by the power MOSFET's on-resistance $R_{DS(on)}$, which is a function of junction temperature (T_J), gate voltage (V_{GS}), and drain current (I_D). $R_{DS(on)}$ is typically specified at I_D equal to half the rated drain current, a V_{GS} of 10 volts, and junction temperatures of 25 and 100°C.

The power MOSFET's static losses can be easily calculated in DC or pulsed applications. First, correct the rated $R_{DS(on)}$ for your drain current and estimated operating temperature with the help of the manufacturers' data sheet curves. Then multiply this value times the RMS load current squared [$P_{static} = I_{rms}^2 R_{DS(on)}$]. You should choose a power MOSFET with a current rating (I_D) and voltage rating (V_{DSS}) well above your worst case load conditions. A good rule of thumb is to select a device with twice your worst case RMS drain current and a voltage rating 25% above your worst case drain voltage.

In high frequency applications switching losses are often more significant than static losses. To minimize switching losses you must decrease the switching times. When a power MOSFET is used in switching applications, the gate cannot be modeled as a simple capacitor due to sizable displacement currents in C_{RSS} , the drain-to-gate capacitor, brought on by large swings in drain-to-gate voltage. As a result, the total input capacitance, C_{ISS} , varies greatly over the power MOSFET's operating range. C_{ISS} can be piecewise modeled as a linear

capacitor in order to find first order approximations of switching times.

A better method of calculating switching times is to use gate charge data from the manufacturers' data sheet. Although a power MOSFET is usually thought of as a voltage controlled device, it can be accurately modeled as a charge controlled device. The charge required for a power MOSFET to handle a given current is relatively constant even though its drain-to-gate capacitance (C_{RSS}) varies drastically with drain-to-gate voltage. The value of C_{RSS} may increase 1000% or more over the operating range.

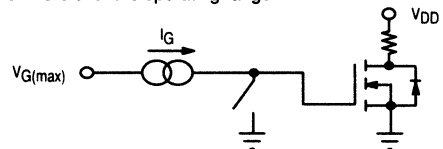


Figure 1. Driving a Power MOSFET with a Constant Current Source

When a power MOSFET is driven by a current source as in Figure 1, its gate voltage will be nearly piecewise linear as shown in Figure 2. The three distinct regions are turn on delay (t_0 to t_1), rise time (t_1 to t_2), and excess charge time (t_2 to t_3). At the end of the turn on delay (t_1) the power MOSFET begins to conduct but the drain current is still very small. During the rise time the power MOSFET actually turns on and the drain voltage drops to almost zero. The resistive switching rise time t_{rise} is actually measured as the time it takes for the drain voltage to drop from 90% to 10% of its highest value. It is called rise time referring to the drain current rise time although the voltage is what's usually measured. This time corresponds to the time that V_{GS} remains in the plateau region of Figure 2.

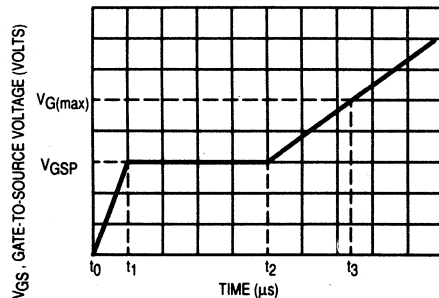


Figure 2. Gate-to-Source Voltage versus Time for a Current Source Turning On a Power MOSFET

During the excess charge time (t_2 to t_3) $R_{DS(on)}$ continues to decrease. This excess charge must be removed during the turn off delay, so driving the gate to an unnecessarily high voltage will increase the total turn off time.

Unlike bipolar transistors, power MOSFETs are majority carrier devices. Without minority injection, power MOSFETs can be turned off just as easily as they are turned on. For identical gate drive currents, rise time will equal fall time. The turn off waveform for a constant gate current will be a mirror image of Figure 2. Note that the turn off delay does not equal turn on delay, it instead corresponds to the turn on excess charge time.

Since the gate current in Figure 2 is constant and equal to the charge per unit time, the horizontal axis can be labeled time or charge. Gate charge data is usually measured using a 1 mA current source which means it will provide 1 nC (nano-Coulomb) of charge in 1 μ s. Manufacturers' data sheets usually include a gate charge chart of V_{GS} vs Q_g with Q_g labeled in nC as in Figure 3. It is important to note that the value of V_{GS} during rise time, also called the plateau voltage, increases with I_D and therefore so does the turn on delay. Also, the amount of charge needed for rise time will vary with the drain supply voltage. This is usually indicated on the gate charge chart by multiple lines for the excess charge region labeled with the corresponding V_{DS} .

To determine the switching times using a current source to drive a Power MOSFET, find the charge required for each region using the gate charge chart, Figure 3, and then use the simple equation:

$$t = Q_g / I_G \quad (1)$$

First find the charge required during the turn on delay region, $Q_{d(on)}$, by drawing a line down from the first inflection point to the horizontal axis of Figure 3. This is the gate charge for the rated or tested I_D . If your actual drain current is different than the rated current you may improve accuracy by linearly scaling $Q_{d(on)}$. Now calculate the turn on delay using Equation 1. Next find the gate charge required for rise time (Q_{rise}) from the gate charge chart as the distance between the first inflection point and the intersection of the plateau with the line for your expected V_{DS} . A typical value is sometimes listed as Q_{gd} . This value may be used to calculate both rise and fall times. Next find the intersection point of your maximum V_{GS} and the line corresponding to your V_{DS} . This is the total gate charge $Q_{g(total)}$. To find the charge required for turn off delay $Q_{d(off)}$ (and turn on excess charge), subtract $Q_{d(on)}$ and Q_{rise} from $Q_{g(total)}$. A maximum total gate charge $Q_{g(max)}$ is often specified to facilitate worst case design, however this figure sometimes includes a substantial guard-band.

When driving a power MOSFET with a voltage source with a series resistance (Thevenin source), the calculations are a little more complex. During the rise and fall times V_{GS} is relatively constant since all the gate current is used to charge the gate-to-drain capacitor. By Ohm's law, I_G is therefore also constant and the gate charge chart can be used with Equation 1 to find rise and fall times. During turn on the voltage across the series resistance is the effective source voltage (usually the supply voltage) minus the gate-to-source plateau voltage, V_{GSP} . During turn off the voltage across the resistor is the plateau voltage minus the effective sink voltage (usually ground). Rise and fall times will therefore typically be different.

Using this information with Equation 1, we can obtain equations for rise and fall time.

$$t_{rise} = \frac{Q_g}{I_G} = \frac{Q_{gd} R_{eff(ON)}}{V_{SOURCE} - V_{GSP}} \quad (2)$$

and fall time:

$$t_{fall} = \frac{Q_g}{I_G} = \frac{Q_{gd} R_{eff(OFF)}}{V_{GSP} - V_{SINK}} \quad (3)$$

V_{GSP} is the Power MOSFET's gate-to-source plateau voltage, V_{source} is the gate driver's effective source voltage, V_{sink} is the gate driver's effective sink voltage, and R_{eff} is the gate driver's effective resistance (output resistance). During turn off V_{sink} may be near zero volts or even a negative voltage.

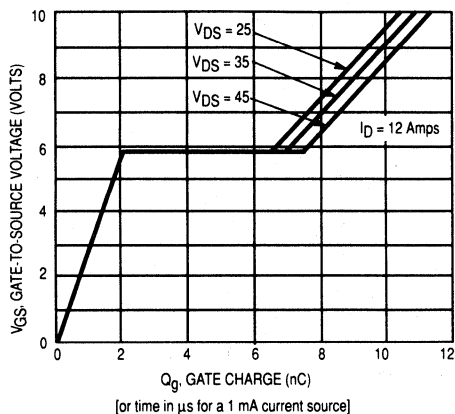


Figure 3. Gate Charge Chart for the MTP3055E

During the turn on and turn off delays gate current is not constant and gate charge data cannot be used to determine switching speeds. The series resistance and the gate capacitance form a simple RC network; however, the capacitance varies greatly over the operating range. To find the switching times you must determine the capacitance for each region from a capacitance chart like Figure 4. During the turn on delay V_{DS} is near its maximum value, V_{GS} is near zero, and the input capacitance is low. Find the value of C_{iss} in the capacitance curve for your maximum value of V_{DS} and use this capacitance, Point A in Figure 4, to calculate the turn on delay. You can use Equation 4 to approximate the turn on delay time.

$$t_{d(ON)} = R_{eff(ON)} C_{iss(MIN)} \ln \left[\frac{V_{SOURCE}}{V_{SOURCE} - V_{GSP}} \right] \quad (4)$$

During the turn off delay V_{DS} will be low and C_{iss} will have a larger value. Find the value of C_{iss} corresponding to minimum V_{DS} and maximum V_{GS} , Point B on the capacitance chart. Then use Equation 5 to approximate turn off delay time.

$$t_{d(OFF)} = R_{eff(OFF)} C_{iss(MAX)} \ln \left[\frac{V_{G(MAX)} - V_{SINK}}{V_{GSP} - V_{SINK}} \right] \quad (5)$$

$V_{G(max)}$ is the initial gate voltage prior to turn off (usually the supply voltage), $R_{eff(off)}$ is the effective series resistance during turn off, and V_{sink} is the effective sink voltage. If V_{sink} is at ground, then the V_{sink} terms will drop out of Equation 5.

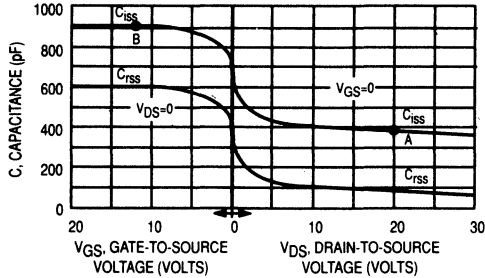


Figure 4. Capacitance Chart for the MTP3055E

Note that the gate charge chart and capacitance curves are related. The slope of the line in the gate charge chart is in volts per nano-Coulomb. A Farad of capacitance is equal to a Coulomb per volt.

$$I(\text{coulomb/sec}) = C \, dV/dt(\text{Farad-volts/sec}).$$

From this equation, you find that

$$\text{Farad} = \text{coulomb/volt}.$$

Therefore, the reciprocal of the slope is the input capacitance in nano-Farads (1000 pF). However, you should use both charts. The gate charge chart is most useful when the input capacitance varies and the gate current is constant (rise and fall times). The capacitance curve is most useful when the input capacitance is constant and the gate current varies (delay times).

DIRECT INTERFACE TO STANDARD POWER MOSFETs

Standard power MOSFETs can be interfaced directly with standard CMOS devices, such as the MC14000 family. This family uses complementary N and P channel FETs for the output stage. Although standard outputs are rated at ± 10 mA and buffer outputs are rated at ± 45 mA, saturation currents for short circuit conditions are much higher. While a CMOS gate should not be short circuited for long periods of time, it may be safely operated in the saturation region when switching large capacitive loads. A 14049UB inverter buffer can typically source 30 mA and sink 120 mA using a 12 volt supply. If the

output current is not limited, the CMOS gate's output will act like a current source. If the output current is limited to less than the saturation currents, the CMOS gate's output will act like a voltage source with a finite output resistance. The MC14000 series family will operate from 3 to 18 volts. The common 12 or 15 volt V_{DD} supply will drive Power MOSFETs nicely.

The 14049UB can be connected directly to a standard power MOSFET such as the MTP3055E as in Figure 5. The MTP3055E is a rugged 12 amp, 60 volt power MOSFET that is very popular in the industry. The gate drive current is not limited by a series resistor and therefore the gate drive current will be equal to the 14049's output saturation currents of $+30/-120$ mA. Using the gate charge data with Equation 1, we can predict the following switching times.

$$t_{d(ON)} = 2 \text{ nC}/30 \text{ mA} = 67 \text{ nsec}$$

$$t_{rise} = 4 \text{ nC}/30 \text{ mA} = 133 \text{ nsec}$$

$$t_{d(OFF)} = 6 \text{ nC}/120 \text{ mA} = 50 \text{ nsec}$$

$$t_{fall} = 4 \text{ nC}/120 \text{ mA} = 33 \text{ nsec}$$

The switching times were measured using the circuit in Figure 5. The actual scope waveforms are shown in Figure 6, and the measured switching times are shown in Table 1.

Table 1. Switching Times for Standard CMOS Devices Driving an MTP3055E
 $I_D = 6$ Amps One gate used unless noted

Driver	V_{CC} (Volts)	R_G (Ω)	$t_{d(on)}$ (ns)	t_{rise} (ns)	$t_{d(off)}$ (ns)	t_{fall} (ns)
4049UB	12	0	50	150	60	50
4049UB	12	220	60	300	200	150
4049UB	12	470	100	400	400	300
4049UB	15	0	40	100	70	40
4049UB	15	220	50	200	280	120
4049UB	15	470	75	330	500	420
4050B	12	0	50	150	60	50
4050B	12	220	60	300	200	150
4050B	12	470	100	400	400	300
4069UB	12	0	100	350	340	250
4069UB	12	220	115	500	380	370
4069UB	12	470	150	680	530	580
4069UB x2	12	0	70	260	170	130

The calculated values were fairly accurate for first order approximations considering that the speeds are high enough that circuit parasitics can affect performance. The saturation currents of the '4049 vary from device to device and with the supply voltage V_{DD} and junction temperature. Driving directly from the logic IC will provide the quickest rise and fall times, but these times will vary greatly.

By adding a resistor between the CMOS buffer's output and the gate of the power MOSFET in Figure 5 we can control switching times by limiting gate drive current. However, increasing the gate resistor also increases the power MOSFET's susceptibility to noise and accidental dv/dt turn on. A rapid change in the power MOSFET's drain voltage will cause a voltage to appear on the gate, which may be sufficient to turn

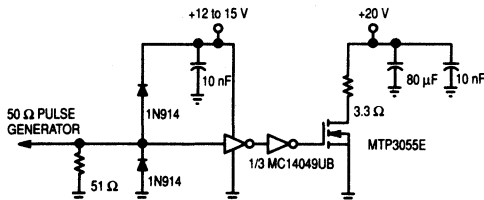


Figure 5. Standard CMOS Interface Circuit

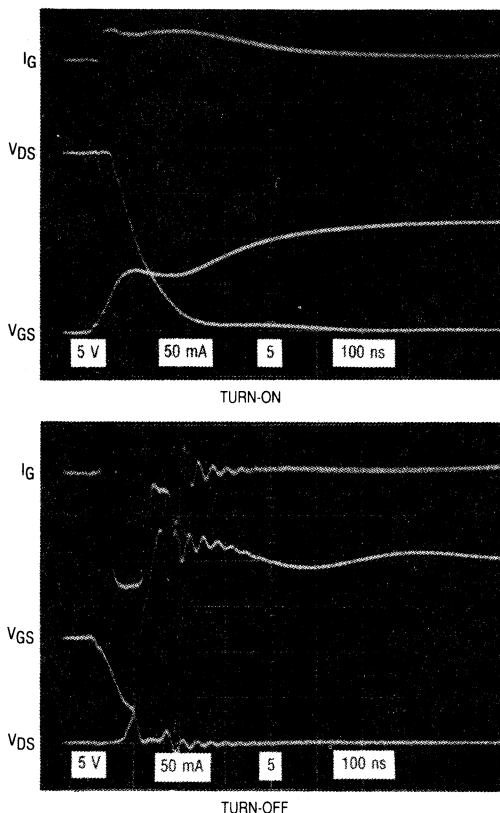


Figure 6. Scope Waveforms for an MC14049 Driving an MTP3055E

it on. Keeping the driver impedance low will minimize or eliminate this phenomenon.

To find the switching times using a gate resistor, use Equations 2 and 3 to find rise and fall times. Then use Equations 4 and 5 to find the delay times. Here $R_{eff(on/off)}$ equals the gate resistor, R_g , plus the CMOS buffer's output resistance, R_o . The approximate output resistance of the '4049 is 200 Ω for turn on and 50 Ω for turn off. Let V_{source} equal V_{DD} and V_{sink} equal zero. Switching times for several gate resistors are summarized in Table 1.

The UB in the MC14049UB stands for "un-buffered". This means that it consists of a single complementary inverter. The additional gate in Figure 5 is used to ensure the power MOSFET driver is itself driven to V_{DD} . The input voltage will greatly affect saturation currents and therefore switching times. The MC14050B is a "buffered" non-inverting buffer and consists of two cascaded inverters. It therefore does not invert the signal, and is less susceptible to soft drive conditions. The diodes on the input in Figure 5 clamp the input voltage to ground and V_{DD} . Excessive voltage applied to a CMOS input may damage its internal static protection diodes. Voltage in excess of the supply voltage, V_{DD} , applied to the output of a CMOS device may cause it to latch-up and destroy itself. Remember

to decouple the logic device, as it is drawing substantial currents.

Open collector TTL gates can also be used to drive standard power MOSFETs. However, most open collector output stages were designed for 5 volt operation. Low power Schottky (LS) gates such as the 74LS05 typically have a collector-emitter breakdown voltage of 10 to 15 volts. This makes them unsuitable for operation using a 12 or 15 volt supply. They can be operated from an 8 to 10 volt supply or with an 8 to 10 volt zener clamp on the output; however, long-term reliability of the logic device will suffer.

The 74LS26 was designed to interface to 15 volt logic and has a tested CE breakdown greater than 15 volts. This Quad NAND gate can be used to drive a power MOSFET with a single pull-up resistor, as in Figure 7. Using a 1.5K Ω pull-up with a 12 volt supply will limit the steady state sink current to 8 mA. This is necessary to guarantee the 'LS26's rated output low voltage V_{OL} of 0.5 volts. Using a smaller pull-up resistor would increase the V_{OL} of the 'LS26, and consequently increase the drain-to-source leakage current of the power MOSFET in the off state.

During turn on, current is supplied by the pull-up resistor. During turn off the 'LS26 must sink both the gate current and the pull-up resistor current. The pull-down transistor of an LS output will typically sink about 30 mA. Turn on times can be calculated using Equations 2 and 4 with $R_{eff(on)}=R_p$ and $V_{source}=V_p$, where R_p is the pull-up resistor and V_p is the pull-up's supply voltage. Turn off times can be calculated using Equations 3 and 5 with $R_{eff(off)}=R_p$ and $V_{sink}=V_p-I_{sink}R_p$ (V_{sink} may be negative). The equations for $R_{eff(off)}$ and V_{sink} are the Thevenin equivalent of an ideal constant current source working against a pull-up resistor. The V_{sink} equation is only valid when the pull-down transistor may be approximated as a current source. During the turn off delay and fall times, the pull-down transistor provides a nearly constant sink current, since the pull-down transistor's collector-emitter voltage exceeds its $V_{CE(sat)}$ and the base drive current is relatively constant.

The 'LS26 with a 1.5K Ω pull-up was used to drive a MTP3055E as in Figure 7. Oscilloscope waveforms are shown in Figure 8, and the switching times are summarized in Table 2.

This configuration provides minimum rise and fall times; however, fall times will vary greatly, since the 'LS26's sink current will vary with temperature and from device to device. A series gate resistor can be used to slow and control turn off. Switching times can again be calculated using Equations 2 through 5. For large gate resistors you may use the following approximations: $R_{eff(on)}=R_p+R_g$, $V_{source}=V_p$, $R_{eff(off)}=R_g$, and $V_{sink}=0.5$ volts. Switching times for several gate resistors are summarized in Table 2.

Table 2. Switching Times for the 74LS26 Driving an MTP3055E
 $I_D = 6$ Amps Only one gate used

VCC (Volts)	R_g (Ω)	R_p (Ω)	$t_d(on)$ (ns)	t_{rise} (ns)	$t_d(off)$ (ns)	t_{fall} (ns)
12	0	1500	200	850	240	175
15	0	1800	200	750	300	175
12	1500	1500	450	2000	1300	1450
12	3000	3000	930	3900	2500	2900

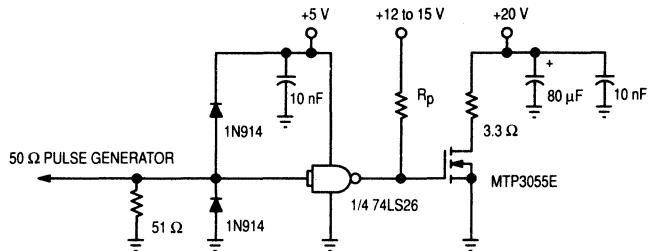


Figure 7. Low Power Schottky Interface Circuit

DIRECT INTERFACE TO LOGIC LEVEL POWER MOSFETS

Logic level Power MOSFETs are designed to be easily interfaced to 5 volt logic devices. They have a larger transconductance and a lower threshold voltage than their conventional counterparts. More importantly, $R_{DS(on)}$ is specified at $V_{GS}=5$ volts. Unfortunately most 5 volt logic families do not have 5 volt high output (V_{OH}) capability. Fast Schottky (FAST) and Low power Schottky (LS) logic have a minimum rated V_{OH} of 2.7 volts. This means that a pull-up resistor to 5 volts is required to drive Logic Level Power MOSFETs. High speed CMOS (HC) has a V_{OH} rating of 4.95 volts, and therefore does not need a pull-up resistor.

Figure 9 shows the output stages of HC and LS logic devices. The HC output stage in Figure 9a is identical to the standard CMOS output stage, except that the complementary MOSFETs have been optimized for 5 volt operation. Most HC devices are buffered by additional complementary stages. The LS output stage in Figure 9b uses a totem pole output. The pull-down transistor is biased on by about 500 μ A and has a current gain of about 60. This means it can sink a maximum of 30 mA. The 110 Ω resistor limits the pull-up transistor's sink current to about 30 mA when the output is shorted.

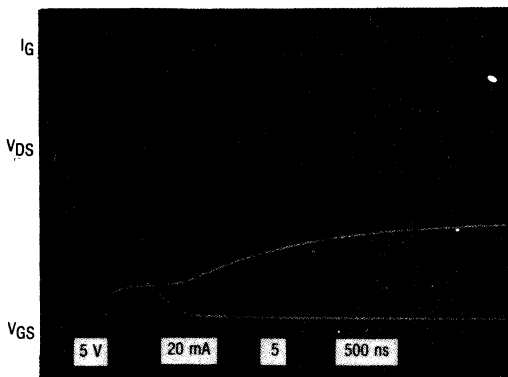
Figure 10 shows how to interface HC, LS, and FAST logic to Logic Level Power MOSFETs. Note the input termination and protection circuitry. This is necessary to drive the logic devices with a pulse generator. It is best to drive the Logic Level Power MOSFET driver with a device from the same logic family. When connecting an HC (or any CMOS) device to a off board connector, the diodes should be used for ESD protection.

Figure 11 shows the switching waveforms for the three logic families driving a Logic Level Power MOSFET using the circuits in Figure 10. The measured switching times are in Table 3.

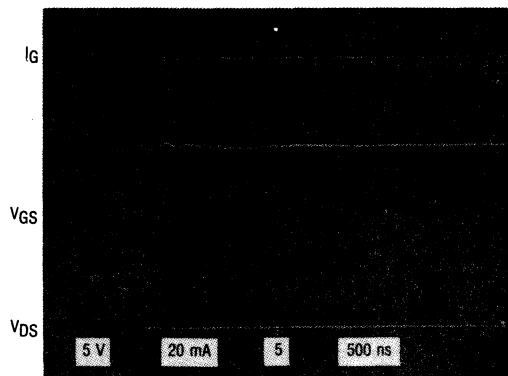
Table 3. Switching Times for Logic Devices Driving a Logic Level MTP3055EL

$I_D = 6$ Amps unless noted. One gate used unless noted.

Driver	R_p (Ω)	$t_d(on)$ (ns)	t_{rise} (ns)	$t_d(off)$ (ns)	t_{fall} (ns)	Comment
74HC04		25	120	85	75	
74LS04	560	45	450	120	130	
74F04	220	15	170	18	21	
74HC04		10	65	30	30	2 gates
74HC04		10	125	35	45	12A 50°C



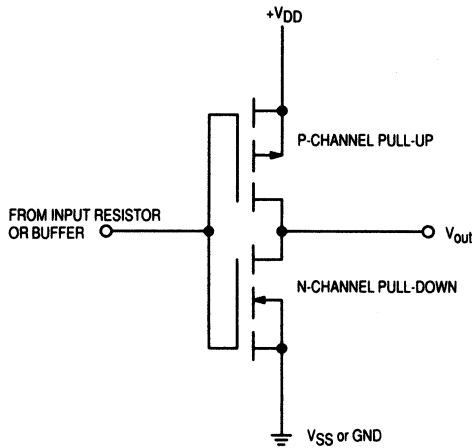
TURN-ON



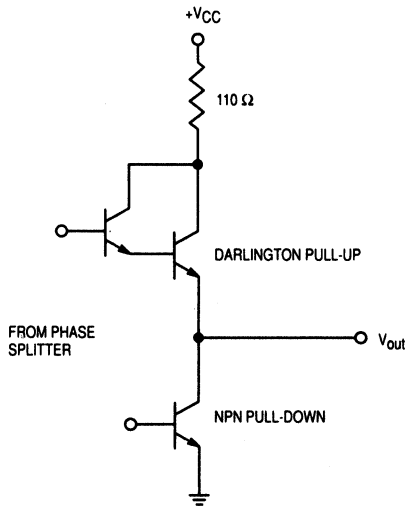
TURN-OFF

Figure 8. Scope Waveforms for a 74LS26 Driving an MTP3055E

$V_{CC} = 5$ Volts, $V_p = 12$ Volts, $R_p = 1.5K\Omega$



(a) CMOS Output Stage



(b) LS TTL Output Stage

Figure 9. Logic Output Stages

A 74HC04 hex inverter can be connected directly to a Logic Level Power MOSFET. The switching times can be calculated the same way as the CMOS inverter buffer. The 'HC04 will source and sink about 50 mA with a 5 volt supply.

The HC family has an operating supply range of 2 to 6 volts. An HC device will drive the Logic Level Power MOSFETs gate to within 50 mV of V_{DD} . However, if the V_{DD} supply falls below 5 volts the switching times and $R_{DS(on)}$ will increase dramatically. A 10% reduction in V_{DD} (to 4.5 volts) will increase the rise time by about 50% and fall time roughly 15%. $R_{DS(on)}$ will increase from 10 to 100% or more depending on the drain

current and junction temperature. If low voltage operation is a real possibility you should choose the Logic Level Power MOSFET and heatsink to handle this worst case condition. Examine the curves for "On-region Characteristics", " $R_{DS(on)}$ versus I_D ", and " $R_{DS(on)}$ versus Temperature" in the manufacturers' data sheet. You may need to use a device with a current rating much larger than your expected load current to attain the desired $R_{DS(on)}$ under low supply conditions. Manufactures are now developing 4 volt logic level power MOSFETs with $R_{DS(on)}$ rated at 4 volts. These devices may be easily interfaced to HC logic devices and operated down to 4 volts. However, the lower threshold voltage makes them more susceptible to noise and increases leakage currents.

The 74LS04 in Figure 10 must have a pull-up resistor to 5 volts. A minimum pull-up resistor of 560 Ω will guarantee the logic device's output low voltage, V_{OL} , of 0.5 volts. During turn on, gate drive current is supplied by the pull-up resistor and the 'LS04's internal pull-up transistor. During turn off the 'LS04 must sink both the gate drive current and the pull-up resistor current. A larger R_p will increase turn on time and decrease turn off time. A smaller R_p would increase the V_{OL} of the 'LS04, increasing the power MOSFET's leakage current. The lower threshold voltage of logic level power MOSFETs makes the V_{OL} rating critical. The threshold voltage of a power MOSFET decreases as temperature increases. Therefore, the V_{OL} of the logic device must be less than the logic level power MOSFET's threshold voltage $V_{GS(th)}$ at its maximum expected junction temperature. For this reason 4 volt logic level power MOSFETs may be incompatible with TTL logic devices.

Switching times can again be estimated by using the Thevenin equivalents of the drive circuit with Equations 2 through 5. During turn on delay, current is supplied by the Darlington pull-up transistor of the 74LS04, and the external pull-up resistor. The Darlington is in saturation with a $V_{CE(sat)}$ of about 1.5 Volts. The 74LS04's output current is then limited by the internal 110 Ω resistor. To calculate turn on delay time, you may use Equation 4 with $V_{source} = V_{CC} [1.5R_p / (R_p + 110\Omega)]$ and $R_{eff(on)} = R_p || 110\Omega$. During rise time nearly all the current is supplied by the pull-up resistor, since V_{GSP} is usually above the V_{OH} of the 'LS04. You may therefore use Equation 2 with $V_{source} = V_{CC}$ and $R_{eff(on)} = R_p$ to estimate rise time.

During turn off the pull-down transistor must sink both the gate current and the pull-up resistor current, just like the open collector 74LS26 in Figure 7. To calculate turn off times, use $V_{sink} = V_{CC} - I_{sink}R_p$ and $R_{eff(off)} = R_p$ with Equations 3 and 5. The pull-down transistor's maximum sink current, I_{sink} , is typically about 30 mA.

The 74LS family's specified supply voltage (V_{CC}) range is from 4.75 to 5.25 volts. The rise time will vary greatly with supply voltage while the fall time only varies by about 5%. The rise time will vary from about +80% to -40% for V_{CC} equals 4.75 and 5.25 volts respectively. This is due to supply voltage affecting both the pull-up resistor current and the pull-up transistor current. Since the operating supply range of LS is less than that of HC logic, $R_{DS(on)}$ will not vary as much, but must be considered.

The FAST logic family can source and sink much more current than the LS family. The 74F04 can source about 50 mA and sink about 200 mA. A minimum pull-up resistor of 220 Ω will guarantee the logic device's output low voltage V_{OL} of

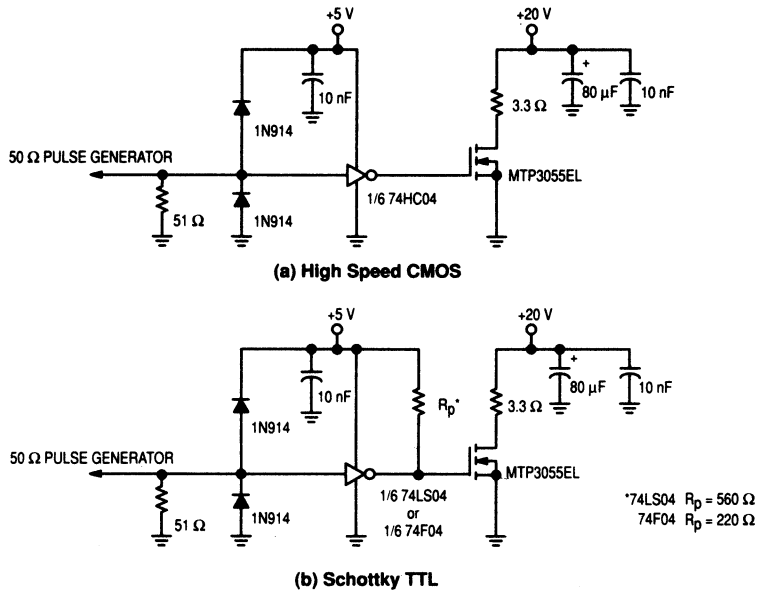


Figure 10. Logic Level Power MOSFET Interface Circuits

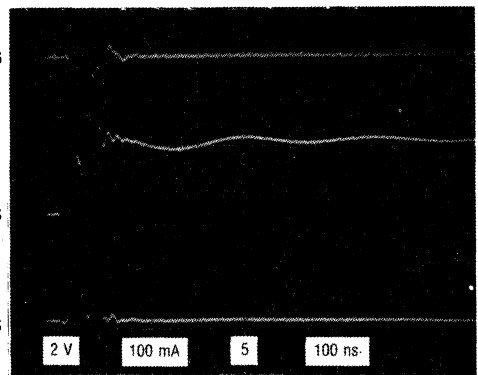
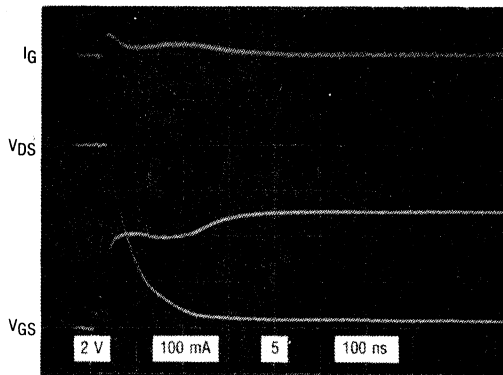
0.5 volts. A larger R_p will increase turn on time and decrease turn off time. The switching times can be calculated as in the LS family. The 74F04 uses an internal 35Ω resistor to limit the pull-up Darlington's output current, instead of the 110Ω resistor. The same supply voltage considerations for LS family also apply to the FAST family.

A series gate resistor may be used with any of the circuits in Figure 10 to slow and control switching times. The switching times for large gate resistors (greater than 200Ω for HC, $5K\Omega$ for LS, and $2K\Omega$ for FAST) can be estimated using $R_{eff}(on/off) = R_g$ with the Equations 2 through 5: When switching loads even slightly inductive, the inductive kick-back during turn off may cause the drain voltage to rise above the load supply. Slowing down the turn off with a gate resistor will reduce this voltage. If this voltage is large enough and sufficient energy is present it may destroy the Power MOSFET. A new family of rugged Power MOSFETs can handle considerable energy under these conditions. You may also want to choose a large R_g value in order to reduce Electromagnetic Interference (EMI). When driving a lamp, you may want to use a very large resistor to limit in-rush current. Long-term reliability of the logic device will also be improved by using a gate resistor and/or a larger pull-up resistor. The gate resistor dissipates most the gate drive power losses, instead of the logic device, reducing stress on the logic output devices. A larger pull-up resistor limits the steady state on current in the pull-down transistors, thereby decreasing their power dissipation. However, using a large gate resistor will also increase the power MOSFET's susceptibility to noise and dv/dt turn on.

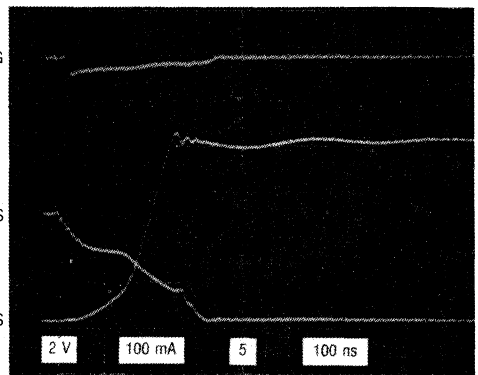
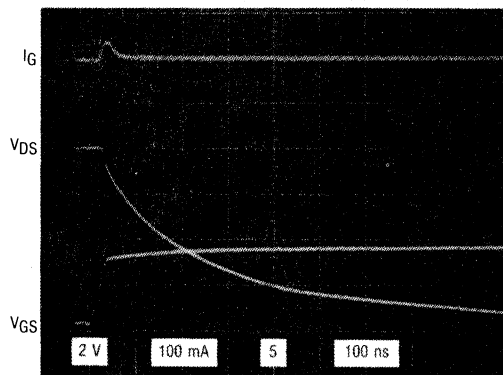
Logic gates on the same chip may be paralleled to increase switching speeds. The output current capability will increase in proportion to the number of gates used. If no gate resistor is used, the switching times will decrease in proportion to the number of gates used. If a gate resistor is used it may be safely decreased, in proportion to the number of gates, to decrease switching times. Paralleling logic gates will not change the total logic package power dissipation, since the output current increases and switching times decrease. When many gates are used, switching times may decrease to the point where they are limited by the stray inductance in the load and in the lay-out. Logic gates on different chips or from different families should not be paralleled because the different propagation delays may cause excessive shoot-through currents which might damage the logic devices.

Spare gates left over from a digital circuit may be used to drive a Logic Level power MOSFET. However, the large currents being used by the driver may cause large amounts of noise on the supply rail. This noise may cause data errors in the other gates on the same IC. Limiting the current with a large gate resistor and carefully decoupling the logic device will reduce the power supply noise. Also the driving logic device must be grounded at same point as the source of the power MOSFET to avoid ground shift problems caused by the large drain currents. If separate logic and analog grounds are used they should be connected only at the source of the power MOSFET.

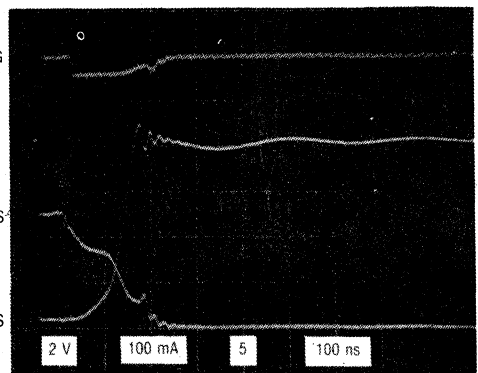
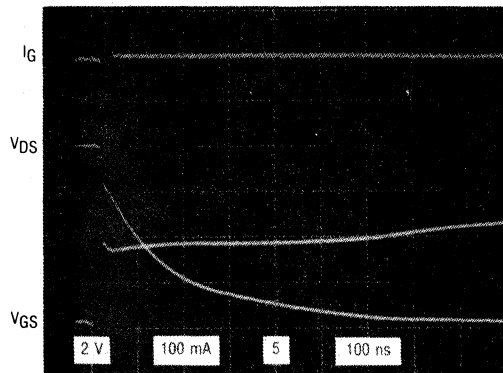
Pay close attention to the power supply scheme. The gate of a power MOSFET should never be left floating with voltage



(a) 74HC04



(b) 74LS04 $R_p = 560 \Omega$



(c) 74F04 $R_p = 220 \Omega$

Figure 11. Logic Devices Driving an MTP3055EL

applied to the drain. When this happens the power MOSFET may turn on and destroy itself if the current is not limited. If separate supplies are used for the load and the logic IC, the logic supply should be powered up first and powered down last. If this is not possible, consider what happens to the logic device output when power is removed. The pull-up resistors in the LS and FAST circuits of Figure 10 will pull the power MOSFET's gate down to V_{CC} when it is low, turning the power MOSFET off. The HC inverter's output, however, will be in a high impedance state when the logic supply voltage is low, allowing the power MOSFET's gate to float. A large resistor to the logic supply voltage or ground, or using a small signal diode to clamp the output to below the logic supply voltage, will solve this problem. Low logic supply voltage may also cause power MOSFET failure due to insufficient gate drive. When a power MOSFET fails the drain voltage will usually appear at its gate, which may take out the entire logic circuit. A gate resistor will also limit the current under this power MOSFET failure condition.

INTERFACING TO A MICROPROCESSOR

Microprocessors can be easily interfaced to a Power MOSFET. Any of the circuits in Figure 10 can be used as a buffer between a microprocessor port and a Logic Level power MOSFET. If you want to use a standard power MOSFET, you will have to use the LS26 circuit in Figure 7 or a level shifter. The MC14504B hex level shifter can be used to interface HC, LS, or FAST to standard CMOS. This level shifter can be used to drive the Power MOSFET directly or with a buffer like the MC14049UB in Figure 5 to decrease switching times. The MC14504B has selectable TTL/CMOS level inputs and standard CMOS outputs. It can source and sink a maximum of about 20 mA using a 12 volt supply.

Be very careful when using bus drivers and latches which have tri-state outputs, like the 74LS240-74HC240 and 74LS373-74HC373, to drive a power MOSFET. The LS tri-state devices require a pull-up resistor to drive the power MOSFET to 5 volts, and will therefore leave the power MOSFET on when the outputs are disabled. The HC devices with tri-state outputs will let the gate float when the outputs are disabled, possibly damaging the power MOSFET. Tri-state devices can be used provided the output enable pin is tied true, low for negative logic enable inputs. HC tri-state devices do not require a pull-up resistor to drive a logic level power MOSFET, and may therefore be used with a pull-down resistor to ground. Note that tri-state outputs should never be pulled above the supply rail or below ground.

When simplicity is important, a single chip microcomputer like the 68HC11 can be used to drive a power MOSFET directly. This microcomputer may be used to perform functions like Pulse Width Modulation, complex motor speed control, and controlling multiple power MOSFETs for bridge applications. When the microcomputer is used in the single chip mode, any one of the 8 pins of parallel output port B can be used to drive a Logic Level power MOSFET. A large gate series resistor should be used to minimize power dissipation and noise on the chip. This means that switching times will be fairly slow. This arrangement also exposes the microprocessor to possible harm from power MOSFET failure. Although all the outputs of port B will be reset to zero on a Power-On Reset (POR), a pull down to ground should be used to ensure the power MOSFET will be off during power down. In some appli-

cations it may be necessary to initialize the power MOSFET gate drive via software before power is supplied to the power MOSFET.

Port B may also be used in a strobed mode by using the STRB signal from the control port D. The STRB signal will go high after the data on port B is valid and may be used to latch or enable a logic device driving a power MOSFET. This mode may be useful when exact synchronization is desired between the microprocessor controlled devices.

When used in the extended memory mode, ports B and C are used for address and data busses. The 68HC24 port replacement unit will replace port B in a software transparent fashion. Thus, a system can be developed using the 68HC11 with a 68HC24 and external memory, while the final product will use only the 68HC11.

CONCLUSION

We have seen that standard Power MOSFETs can be interfaced directly to standard CMOS logic with very good performance, about 50 ns rise and fall times for the MC14049UB driving a 12 Amp power MOSFET. Standard Power MOSFETs may also be interfaced to 5 volt logic using a special interface device such as the 74LS26 open collector NAND gate or the MC14504B hex level shifter. The 74LS26 driving a 12 Amp standard Power MOSFET gives turn on times of about 1 μ s and fast turn off times of less than 200 ns. Switching times may be easily estimated using four simple equations and a series resistor may be selected to give the desired rise and fall times.

Logic Level Power MOSFETs can be driven directly with HC logic, and by LS logic with the addition of a pull-up resistor. Switching speeds using an HC device are very fast, less than 150 ns per gate when driving a 12 Amp power MOSFET. Using an LS device, turn on speed is good, about 0.5 μ s, and turn off speed is excellent, less than 150 ns. Again, switching speeds may be easily estimated and a series resistor may be selected to give the desired performance.

Logic power supply variations are the most important aspect affecting Logic Level Power MOSFET performance. Power supply sequencing and under-voltage protection is necessary to ensure system integrity. Circuit lay-out and power supply decoupling are also important at high speeds.

Finally a Logic Level Power MOSFET may be interfaced directly to a dedicated microprocessor output port when microprocessor control is desired.

Bibliography

- Motorola Power MOSFET Transistor Data, DL135 Rev 2, 1988, Ch 1-4, 6, and pp. 3.711-716.
- Motorola MTP3055EL Designer Data Sheet, MTP3055EL/D, 1988.
- Motorola CMOS Logic Data, DL131 Rev 1, 1988, Ch 5, pp. 6.125-128 and pp. 6.154-155.
- Motorola FAST and TTL Data, DL121 Rev 3, 1988, Ch 2, pp. 4.6-7 and 5.6.
- Motorola High-Speed Logic Data, DL129 Rev 3, 1988, Ch 4 and pp. 5.11-14.
- Paul R. Grey and Robert G. Meyer. Analysis and Design of Analog Integrated Circuits, Second Edition, 1984, Wiley and Sons, pp. 55-75, and Ch 12.
- Adel S. Sedra and Kenneth C. Smith. Microelectronic Circuits; Holt, Rinehart and Winston; 1982; pp. 689-715.

Basic Servo Loop Motor Control Using the MC68HC05B6 MCU

By Jim Gray

This application note describes a basic circuit and software implementing proportional derivative (PD) closed-loop speed control for a brush motor using four integrated circuits (ICs), two opto discretes, and less than 200 bytes of code.

Feedback control systems using digital algorithms implemented on microcontroller units (MCUs) are becoming increasingly commonplace. The use of an MCU in this type of control application is justified when system flexibility is needed, such as varying drive motors or storing wear parameters in electrically erasable programmable read-only memory (EEPROM). Typically, the system would be modeled mathematically in the discrete time domain due to the use of sampled rather than continuous data. The linear difference equations describing the transfer function of the system are solved using z-transforms, allowing, in the case of proportional-integral-derivative (PID) control, the determination of constants for proper system performance and stability. However, this level of analysis is not necessary to illustrate how straightforward the implementation is using the MC68HC05B6 and the MPM3004 TMOS™ H-bridge. The generalized flow of a PD loop is shown in Figure 1. The transfer function of $G_C(s)$ consists of the PD control, and $G_P(s)$ represents the power amplifier, motor, and load. Here s is a complex variable having both real and imaginary parts. The proportional term K_P can be accomplished with shifting operations, at least to the resolution of powers of 2. The derivative term, K_{DS} , of $f(t)$ is approximately

$$\left. \frac{df(t)}{dt} \right|_{t=kT} \cong \frac{1}{T} [f(kT) - f(k-1)T]$$

where $f(kT)$ is the current value of the controlled parameter, and $f(k-1)T$ is the value of the same parameter at the previous sampling time. In this example, k_{DS} is realized as the rate of change of the difference between the measured and the desired period of motor-shaft rotation.

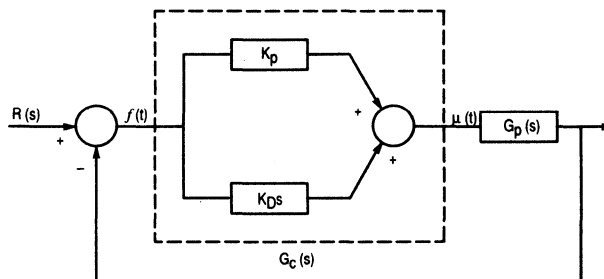


Figure 1. PD Loop Flow

The MC68HC05B6 is an M68HC05 MCU Family member with two channels of programmable pulse-length modulation on-chip. When used with an H-bridge device such as the MPM3004, these channels can control bidirectional currents of up to 10-A continuous (25-A peak) at 60 V (see Figure 2). Two I/O pins and both pulse-length modulation (PLM) channels are used to control the MPM3004. Proper gate drive and level conversion is provided by the MC34151 dual inverting gate drivers. Input to the control loop consists of the MLED71 infrared emitter and MRD750 photo Schmitt trigger detector coupled through a slotted disc on the motor shaft. The TCAP2 pin and associated input capture registers are used to convert the optical index marks into a time measurement. Great care must be taken to ensure an adequate current source for the MPM3004 and to isolate the supply for the MC34151s. Separate circuit runs and 0.1- μ F bypass capacitors on the MC34151 ICs were used in this case.

The justification for adding a derivative term to a proportional controller can be easily understood by examining the reasons for the overshoot and ringing typical of an underdamped proportional-only controller. When proportional control applies additional power to correct an underspeed condition, it does so continuously until the error term is zero, resulting in a power setting that ensures an overspeed condition. The converse occurs when reducing motor speed. The rate of change of the error signal as excessive power is being applied to correct underspeed will be a relatively large negative value (the error term is being rapidly reduced). Thus, the derivative of the error term is of the correct sign to compensate the proportional gain term. One effect of this compensation is to retard the loop's response time, but the proportional gain can be increased to offset this.

The listing (see Figure 3) shows the assembly source code for speed measurement and the PD control of PLMA, which drives the power H-bridge in one direction. The opposite direction of rotation is obtained by complementing bits 0 and 1 of port A and driving the opposite lower leg of the H-bridge with PLMB. Eight-bit arithmetic was used exclusively in this example for space and clarity. Although this approach is functional, 16-bit routines for multiply and divide, given in Reference 2, are better for finer control. Routines to set initial values, control direction of rotation, and check for motor stall are also necessary, although they are not shown in this application note.

Figure 4 shows the response of the system to various changes in load. The data was captured in an emulator trace buffer (Motorola CDS8 Jewelbox) and plotted using a data base program. Beginning from a no-load condition at 4 s, loading (an uncalibrated friction brake) was ramped to cause approximately a 50-percent duty cycle. Starting at 10 s, the load was then increased again until the system was at the limit of compliance — i.e., at full power and still maintaining the desired speed. Next, at 14 s, approximately half the load was rapidly (0.1 s) removed. The gain of the proportional term was 2, and the derivative constant was 1. In systems where a low-pass filter would be beneficial or the steady state error is potentially large, an integral term could be added for full PID control.

REFERENCES

1. Kuo, Benjamin C., *Automatic Control Systems*, New Jersey: Prentice-Hall, 1987.
2. M6805UM/AD2, *M6805 HMOS/M146805 CMOS Family User's Manual*, New Jersey: Prentice-Hall, 1983.
3. MC68HC05B6/D, *MC68HC05B6 Data Sheet*, Motorola, 1988.
4. M68HC05AG/AD, *M68HC05 Applications Guide*, Motorola, 1989.

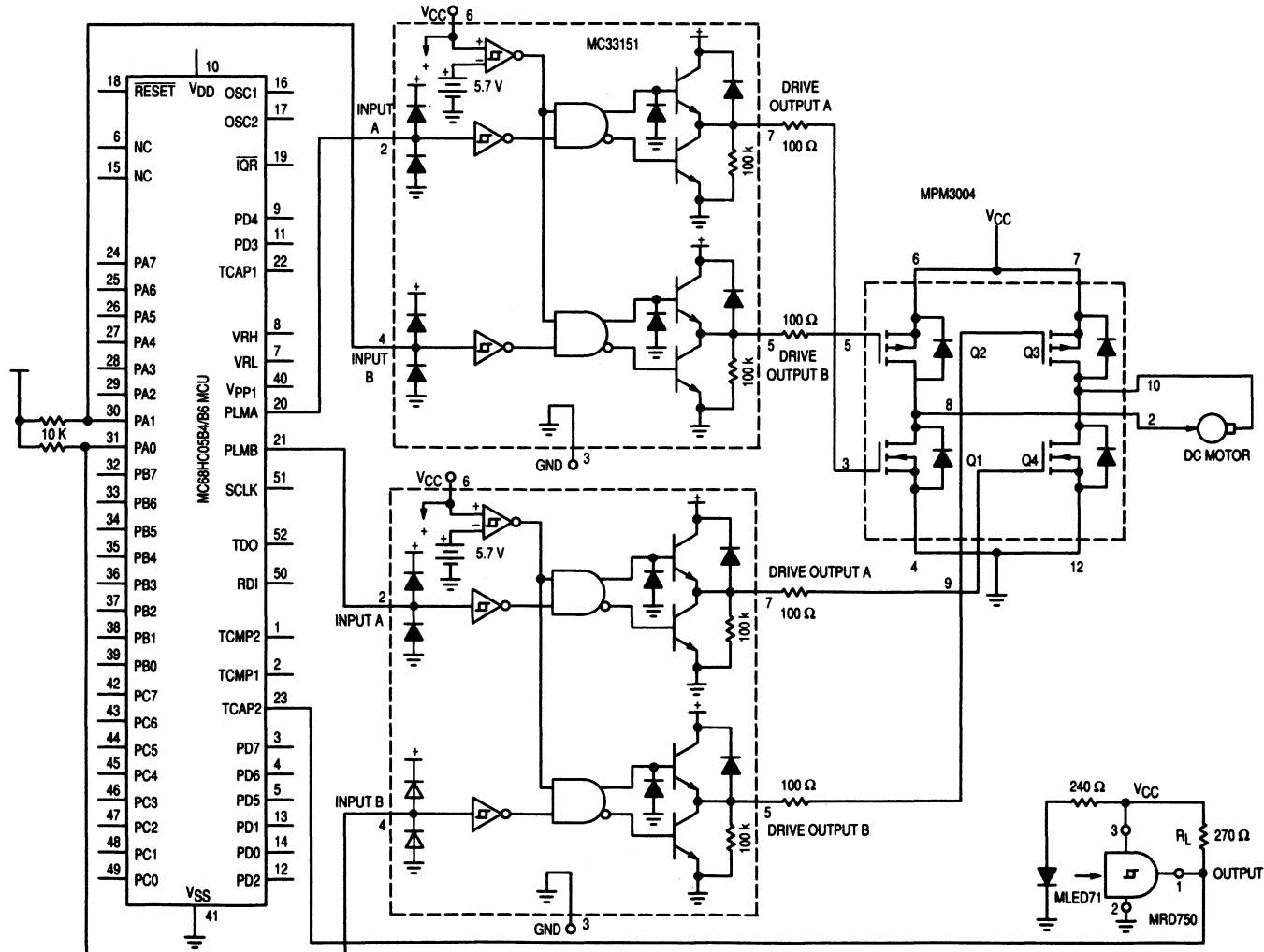


Figure 2. Block Diagram of Servo Loop Motor Control

Figure 2. Block Diagram of Servo Loop Motor Control

```

1          *****
2          *          MC68HC05B6 SERVO LOOP MOTOR CONTROL EXAMPLE          *
3          * This program performs a closed loop servo speed control using PLMA for *
4          * output. Speed is measured optically with a slotted disk. The optically *
5          * detected index mark controls TCAP2 which allows calculation of the *
6          * period of revolution for the loop input.                          *
7          *****
8
9 0000          org      $0
10          cycles off
11 0000
12 0000          PADR    RMB    1
13 0001          PBDR    RMB    1
14 0002          PCDR    RMB    1
15 0003          PDIDR   RMB    1
16 0004          PADDR   RMB    1
17 0005          PBDDR   RMB    1
18 0006          PCDDR   RMB    1
19
20 000A          ORG     $0A
21
22 000A          PLMA    RMB    1
23 000B          PLMB    RMB    1
24 000C          MISC    RMB    1
25
26 0012          ORG     $12
27
28 0012          TCR     RMB    1
29 0013          TSR     RMB    1
30 0014          CAHR1   RMB    1
31 0015          CALR1   RMB    1
32 0016          COHR1   RMB    1
33 0017          COLR1   RMB    1
34 0018          CNTHR   RMB    1
35 0019          CNTLR   RMB    1
36 001A          ACNTHR  RMB    1
37 001B          ACNTRLR RMB    1
38 001C          CAHR2   RMB    1
39 001D          CALR2   RMB    1
40
41 0050          ORG     $50
42
43 0050          BCNTH   RMB    1
44 0051          BCNTL   RMB    1
45 0052          ECNTH   RMB    1
46 0053          ECNTL   RMB    1
47 0054          PERIOD  RMB    1
48 0055          PLMTMP  RMB    1          MUST BE INITIALIZED WITH STARTING VALUE
49 0056          DESPRD  RMB    1          MUST BE INITIALIZED WITH DESIRED PERIOD COUNT
50 0057          DELTAN  RMB    1
51 0058          DELTAO  RMB    1
52 0059          DELTADC RMB    1
53 005A
54 0F00          ORG     $F00
55
56 0F00 A604          BEGIN  LDA    #S4          SELECT SLOW PLM REPETION RATE
57 0F02 B70C          STA    MISC          SPEED
58 0F04 B655          LDA    PLMTMP        LOAD PLM VALUE
59 0F06 B70A          STA    PLMA
60 0F08 B613          KEYS  LDA    TSR          CLEAR FLAG AND ANY PENDING INT.
61 0F0A B61C          LDA    CAHR2
62 0F0C B61D          LDA    CALR2
63 0F0E 1E12          BSET   7,TCR          SET INPUT CAPTURE INTERRUPT ENABLE
64 0F10 9A          CLI
65 0F11 20FE          WAIT  BRA    WAIT          CLEAR I BIT ALLOWING TIMER INTERRUPTS
66 0F13 B613          RPM   LDA    TSR          WAIT FOR OPTO INDEX TCIC INTERRUPT
67 0F15 B61C          LDA    CAHR2          CLR TSR BIT 4 TO ENSURE
68 0F17 B61D          LDA    CALR2          SYNCHRONIZATION TO INDEX

```

Figure 3. MC68HC05B6 Servo Loop Motor Control Example

69	0F19	081302	TFLAG1	BRSET	4,TSR,INDEX1	TEST FLAG FOR INDEX1
70	0F1C	20FB		BRA	TFLAG1	
71	0F1E	B61C	INDEX1	LDA	CAHR2	STORE COUNT
72	0F20	B750		STA	BCNTH	
73	0F22	B61D		LDA	CALR2	
74	0F24	B751		STA	BCNTL	
75	0F26	4F		CLRA		DELAY TO AVOID RETRIGGER ON SAME INDEX
76	0F27	4A	DEC1	DECA		
77	0F28	26FD		BNE	DEC1	
78	0F2A	B613		LDA	TSR	CLEAR FLAG AND WAIT
79	0F2C	B61C		LDA	CAHR2	FOR INDEX2
80	0F2E	B61D		LDA	CALR2	
81	0F30	081302	TFLAG2	BRSET	4,TSR,INDEX2	
82	0F33	20FB		BRA	TFLAG2	
83	0F35	B61C	INDEX2	LDA	CAHR2	STORE SECOND COUNT
84	0F37	B752		STA	ECNTH	
85	0F39	B61D		LDA	CALR2	
86	0F3B	B753		STA	ECNTL	
87	0F3D	B652		LDA	ECNTH	CALCULATE PERIOD
88	0F3F	B050		SUB	BCNTH	THEN
89	0F41	B754		STA	PERIOD	STORE.
90	0F43	B657		LDA	DELTAO	GET PREVIOUS ERROR AND
91	0F45	B758		STA	DELTAO	STORE IT.
92	0F47	B656		LDA	DESPRD	LOAD DESIRED PERIOD, SUBTRACT ACTUAL
93	0F49	B054		SUB	PERIOD	TO FORM DELTAN.
94	0F4B	2529		BLO	INCSPD	GO TO INCREMENTING PLM
95	0F4D	48		LSLA		MULTIPLY ERROR BY 2.
96	0F4E	B757		STA	DELTAO	OR FALL THRU TO DECREMENTING HERE.
97	0F50	B658		LDA	DELTAO	FORM RATE OF CHANGE
98	0F52	B057		SUB	DELTAO	OF ERROR
99	0F54	B759		STA	DELTADC	AND STORE.
100	0F56	B657		LDA	DELTAO	GET CURRENT ERROR
101	0F58	B059		SUB	DELTADC	AND APPLY DE/DT CORRECTION
102	0F5A	B759		STA	DELTADC	THEN STORE.
103	0F5C	B655		LDA	PLMTMP	GET CURRENT PLM
104	0F5E	B057		SUB	DELTAO	AND APPLY CORRECTION.
105	0F60	2208		BHI	ADJDN	BRANCH TO DECREMENT IF RESULT POSITIVE
106	0F62	A610	PLMMIN	LDA	#\$10	OTHERWISE IN LOW SATURATION SO
107	0F64	B70A		STA	PLMA	KEEP PLM AT MINIMUM.
108	0F66	B755		STA	PLMTMP	
109	0F68	2023		BRA	DONE	
110	0F6A	A110	ADJDN	CMP	#\$10	SEE IF PLM AT MINIMUM
111	0F6C	2202		BHI	DECSPD	
112	0F6E	20F2		BRA	PLMMIN	
113	0F70	B70A	DECSPD	STA	PLMA	DECREMENT PLMA
114	0F72	B755		STA	PLMTMP	UPDATE PLMA TEMPORARY LOCATION
115	0F74	2017		BRA	DONE	
116	0F76	48	INCSPD	LSLA		MULTIPLY ERROR BY 2
117	0F77	B757		STA	DELTAO	INCREMENT WITH SATURATION
118	0F79	B658		LDA	DELTAO	FORM RATE OF CHANGE
119	0F7B	B057		SUB	DELTAO	OF ERROR.
120	0F7D	BB57		ADD	DELTAO	NOW ADD IT TO CURRENT DELTA
121	0F7F	B759		STA	DELTADC	TO FORM RATE OF CHANGE COMPENSATED ERROR.
122	0F81	B655		LDA	PLMTMP	GET CURRENT PLM
123	0F83	B059		SUB	DELTADC	AND APPLY CORRECTION.
124	0F85	2502		BLO	ADJUP	
125	0F87	2004		BRA	DONE	IN SATURATION OR CORRECTION EQUALS 0
126	0F89	B70A	ADJUP	STA	PLMA	
127	0F8B	B755		STA	PLMTMP	
128	0F8D	80	DONE	RTI		RETURN TO WAIT
129						
130						
131	1FF0		ORG	1FF0		set vectors
132	1FF0	0F00	FDB	BEGIN		R
133	1FF2	0F00	FDB	BEGIN		SCI
134	1FF4	0F00	FDB	BEGIN		TOV
135	1FF6	0F00	FDB	BEGIN		TOC
136	1FF8	0F13	FDB	RPM		TIC
137	1FFA	0F00	FDB	BEGIN		IRQ
138	1FFC	0F00	FDB	BEGIN		SWI

139 1FFE 0F00
140 2000

FDB BEGIN RES
END

Symbol Table:

Symbol Name	Value	Def.#	Line Number	Cross Reference
ACNTHR	001A	*00036		
ACNTLR	001B	*00037		
ADJDN	0F6A	*00110	00105	
ADJUP	0F89	*00126	00124	
BCNTH	0050	*00043	00072	00088
BCNTL	0051	*00044	00074	
BEGIN	0F00	*00056	00132	00133 00134 00135 00137 00138 00139
CAHR1	0014	*00030		
CAHR2	001C	*00038	00061	00067 00071 00079 00083
CALR1	0015	*00031		
CALR2	001D	*00039	00062	00068 00073 00080 00085
CNTHR	0018	*00034		
CNTLR	0019	*00035		
COHR1	0016	*00032		
COLR1	0017	*00033		
DEC1	0F27	*00076	00077	
DECSPP	0F70	*00113	00111	
DELTADC	0059	*00052	00099	00101 00102 00121 00123
DELTAN	0057	*00050	00090	00096 00098 00100 00104 00117 00119 00120
DELTAO	0058	*00051	00091	00097 00118
DESPRD	0056	*00049	00092	
DONE	0F8D	*00128	00109	00115 00125
ECNTH	0052	*00045	00084	00087
ECNTL	0053	*00046	00086	
INCSPP	0F76	*00116	00094	
INDEX1	0F1E	*00071	00069	
INDEX2	0F35	*00083	00081	
KEYS	0F08	*00060		
MISC	000C	*00024	00057	
PADDR	0004	*00016		
PADR	0000	*00012		
PBDDR	0005	*00017		
PBDR	0001	*00013		
PCDDR	0006	*00018		
PCDR	0002	*00014		
PDIDR	0003	*00015		
PERIOD	0054	*00047	00089	00093
PLMA	000A	*00022	00059	00107 00113 00126
PLMB	000B	*00023		
PLMMIN	0F62	*00106	00112	
PLMTMP	0055	*00048	00058	00103 00108 00114 00122 00127
RPM	0F13	*00066	00136	
TCR	0012	*00028	00063	
TFLAG1	0F19	*00069	00070	
TFLAG2	0F30	*00081	00082	
TSR	0013	*00029	00060	00066 00069 00078 00081
WAIT	0F11	*00065	00065	

Errors: None
Labels: 47

Last Program Address: \$1FFF
Last Storage Address: \$FFFF
Program Bytes: \$009E 158
Storage Bytes: \$0020 32

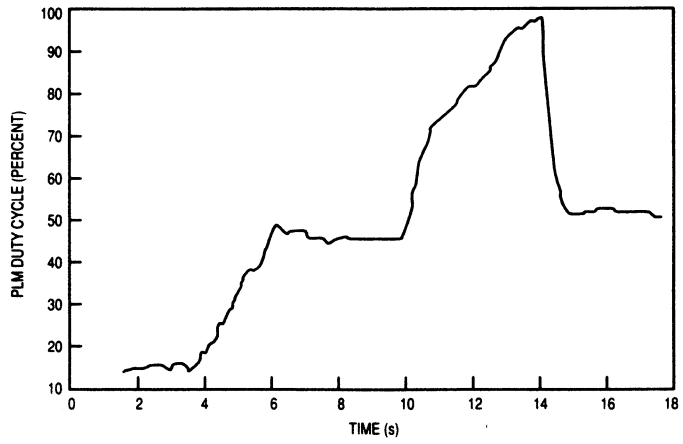


Figure 4. Step Response of PLM Motor Control

A Software Method for Decoding the Output from the MC14497/MC3373 Combination

Prepared by: Steve Reinhardt

The electronics industry has used infrared media as a simple, easy, and effective method of wireless communications over short distances. It is not without its problems since simple on/off modulation is affected by the many infrared sources in our environment today. To provide immunity from the noise created by lamps, lighters, electronics, and even humans, the IR carrier is modulated at a rate that would not occur in nature. The industry has settled on around 40 kHz as the modulation frequency.

The data that is transmitted usually takes the form of AM (or CW – continuous wave); that is the carrier is turned on and off for variable periods of time. Some have used a FM scheme, where the modulation frequency is changed to represent 1 or 0. The output of detectors is generally the same: that is a logic 0 represents a presence of carrier in AM, or one of the frequencies in FM. A logic 1 then represents no carrier in AM, or the second frequency in FM.

The encoding of the data varies widely, from schemes that encode the data as variable pulse widths, constant length coding schemes, or simple ASCII, to the biphase scheme used in the MC14497. Any of these schemes can be decoded by the use of a microcomputer that has a timer, such as the MC68HC05 family or the MC68HC11 family of parts.

THE MC14497

The MC14497 is a complete building block for IR data transmission, lacking only a high current driver to power the IR LED (or LEDs, depending on the range required) such as the MLED81. The chip limits the duty cycle of the LED to about 10%. The use of an inexpensive ceramic resonator generates the 31.25 kHz carrier. A simple SPST matrix keyboard completes the transmitter.

THE MC3373

The MC3373 is a companion chip to the MC14497. It provides all of the front-end signal processing to interface an IR photo detector, such as the MRD821, to a TTL level. It includes the gain stages, with automatic background level control (AGC), a simple frequency discriminator to eliminate interference from other sources, and a wave shaper that generates a TTL or CMOS output level. The MC3373 does not decode the data, it merely reconstructs it in logic level form (the way it was trans-

mitted) to facilitate decoding. It too requires few outside components, such as a tuned circuit and a few capacitors for wave shaping. Care must be taken in circuit layout, as this device operates at very high gain to accommodate the low level input signal from the photodetector. Since there are frequency sensitive components in the circuit, it is best to minimize lead lengths, work on a ground plane, and perhaps even put a shield around the components that make up the receiver.¹

THE ENCODED BIT STREAM

To understand how to decode the data from the MC14497, it is important to understand what is transmitted. Each word transmitted consists of an AGC burst, a start bit, and 6 data bits. The 6 data bits represent 64 individual channels (0–63). However, channel 63 (111111) is never sent.

The carrier frequency is determined by dividing the oscillator frequency by 16. For a 500 kHz resonator, the carrier is 31.25 kHz. The baud rate (signalling rate) is equal to the carrier divided by 32, or the oscillator divided by 512. Again, for a 500 kHz resonator, the baud rate is approximately 976 bps. Each command word takes approximately 8 ms to send.

Refer to Figure 1. Data is the representation of the channel code (001010, channel 10). Carrier represents the output of the MC14497. Recovered is the signal that is output from the MC3373. Notice that the data is inverted, or normally high. When a key is pressed, the chip sends a signal to setup the AGC in the receiver. In the AM mode, which is most common, each transmitted word is preceded by a 512 μ s burst (oscillator divided by 256). One bit time later, the start bit is sent. The biphase modulation scheme then uses the position of a carrier within the bit time to represent the data value. Refer to Figure 2. The presence of carrier immediately after the bit time boundary represents a 1. The lack of the carrier represents a 0. The phase then changes so that there is a constant modulation, with clock edges on each bit time boundary. This feature is important in some communications schemes, but is not important here.

If a key is held down, the code is repeated at 90 ms intervals. See Figure 3. This results in a duty cycle of about 10% so that the IR LED can be pulsed at high peak power. At this duty cycle, the MLED81 can tolerate peak currents well in excess of 100 mA. Once a key is released, the MC14497 automatically sends the code for channel 62 (111110), which indicates end of transmission (EOT).

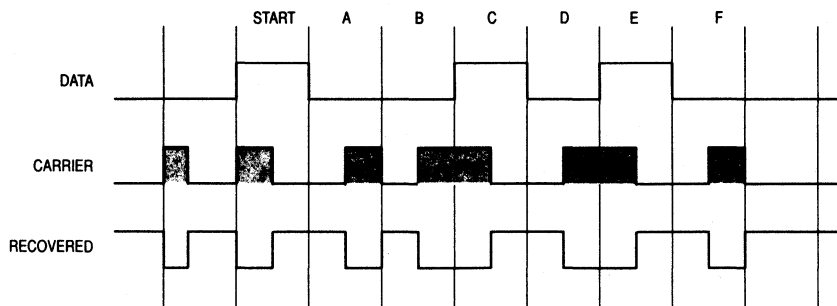


Figure 1

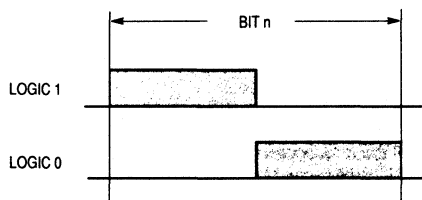


Figure 2

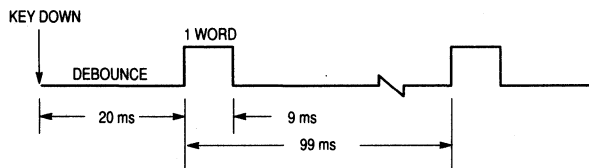


Figure 3

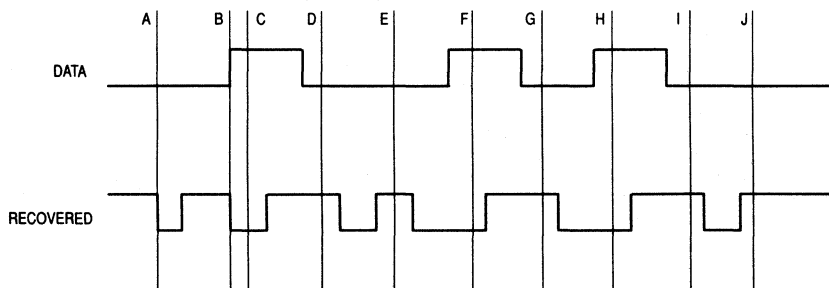


Figure 4


```

*occurs, sample the input line and shift the equivalent value
*into the data register. Do so for all six data bits, then word
*is complete.
*
bitint      ldaa    PORTA      *get the present value of the data
           anda    #4         *mask all but PA3, the input
           rora    *shift towards carry
           rora    *once more
           rora    *now the input bit is in carry
           cmc     *data from 3373 is inverted
           ldaa    data      *get the saved data byte
           rola    *rotate the carry in
           staa    data      *save the byte
           dec     count     *the number of bits is complete?
           bne    next2     *not done yet
           bset   flags,1   *we're done
           bra    endin     *no more in this word
next2      ldd     OC1       *get the last timer value
           addd   baud      *add the baud interval
           std    OC1       *store it for the next interrupt
           ldaa   #1        *
           staa   TMSK2     *and clear the flag for the next interrupt
endin      rti

```

MC68HC05 PROGRAM LISTING

```

start      lda    #$C0      *set up timer control register
           sta    TCR      *interrupts enabled, negative edge
           lda    #$0      * make sure port a is an input
           sta    PADDR    * write to data direction
           lda    #6
           sta    count    *number of bits to assemble
           .
           .
           .
main       equ    *        *main program
           .
           .
           .
*input capture service routine. The first time through, the value of
*the timer is saved. The second time through, the difference is
*calculated, determining the bit interval. The time slice is shifted
*by one quarter time for sampling the data.
*
*timeint   brset   flags,0,next
           lda    ICH      *grab the high byte
           sta    saveit   *store it
           lda    ICL      *then the low byte
           sta    saveit+1 * and save that.
           bset   flags,0 *got first edge flag
           bra    endint
next      lda    ICL      *get the new low byte
           sub    saveit+1 *subtract the old low byte
           sta    byte+1   *and save the result
           lda    ICH      *get the new high byte
           sbc    saveit   *subtract the old, with carry
           sta    byte     *save the byte interval
           lsr    *divide by two
           sta    OCH      *to output compare
           lda    byte+1   *get low byte
           ror    *use ror to get carry in
           sta    OCL      to output compare
           lda    OCH
           lsr
           sta    OCH
           lda    OCL
           ror
           sta    OCL      *finish divide by four
           lda    #$7F     *mask off input capture
           sta    TSR      *by writing to timer status
           bclr   flags,0  *clear first edge flag bit

```

```

endint      rti
*output compare service routine. For each interrupt that occurs,
*sample the input line and shift the appropriate data bit into the
*data register. Do so for all six bits, then word is complete.
*
bitint      lda    PADR          *get the data bit value
            lsl    data          *get the data to carry
            lda    data          *get the data byte
            rol    data          *and assemble the byte
            sta    data          *to data
            dec    count        *check to see if all done
            beq    endbit       *exit if done
            lda    OCL
            add    byte+1       *set up the next interrupt
            sta    OCL          *write to OCL resets flag
            lda    OCH
            adc    byte          *use add with carry to do 16 bit
            sta    OCH
            rti
endbit      lda    data          *get the word
            coma          *remember the data was inverted
            and    #$5F         *and there were only 6 bits
            sta    data
            bset   flags,1     *word ready flag bit
            rti

```


Bi-directional Data Transfer between MC68HC11 and MC6805L3 using SPI

by
Richard Soja, Motorola, EKB

INTRODUCTION

One of the most powerful features shared by a wide range of Motorola MCUs is the Serial Peripheral Interface (SPI). It is primarily designed to operate as a synchronous, 8-bit communication system and is implemented entirely with on-chip hardware. This frees the CPU for other tasks and ensures a minimum of software overhead associated with the SPI system.

The SPI is available in two basic forms:

1. Level 1 SPI — implemented on the MC68HC11, HC05C4 MCUS,
2. Level 2 SPI — implemented on the MC6805S2/S3/L3/L8 MCUs.

Note that the HCMOS family of MCUs only support level 1, while level 2 is implemented only on HMOS MCUs.

Though both levels of SPI can communicate easily with each other, level 2 has a number of additional capabilities, including asynchronous communication. This application note is aimed at describing a method of achieving synchronous communication between a level 1 and level 2 SPI, and details the subtle relevant differences in the on-chip implementation of each.

DESCRIPTION

The two MCUs used in this application are the high performance MC68HC11 and the low cost MC6805L3.

Data is transferred between the MCUs on a single

bi-directional line, with the clock supplied on an additional line. Also, to ensure initial synchronisation between each MCU, a software handshake sequence is implemented on the same lines which provide the clock and data. This has the considerable advantage of minimising the number of lines between each MCU as additional control lines are not needed.

The handshake sequence is also necessary for two other reasons; the 6805L3 receive data register is unbuffered, and it is not possible for the 6805L3 to stop transmission of data from the 68HC11 by inhibiting the clock signal. The fact that the 6805L3 data register is unbuffered means that, if a handshake sequence was not implemented, new data could begin to get clocked in before the previous data were read. Also, the on-chip configuration of the 68HC11's SPI means that, if an attempt were made to slow down or stop its clock during transfer of a byte, there would be a resultant loss of synchronism between the transmitting and receiving MCUs.

The 68HC11 software is implemented as the clock master (i.e. it provides the clock output), while the 6805L3 is the clock slave. As there are no other clock masters or slaves in the system, software is kept to an absolute minimum. In fact the main transfer routine (XFER) for the 68HC11 is only 27 bytes long, while the 6805L3 uses only 30 bytes.

The other significant advantage of this implementation is that none of the 6805L3 timers are required for SPI operation, thus ensuring a minimal impact on any other application dependent tasks the MCU may be executing.

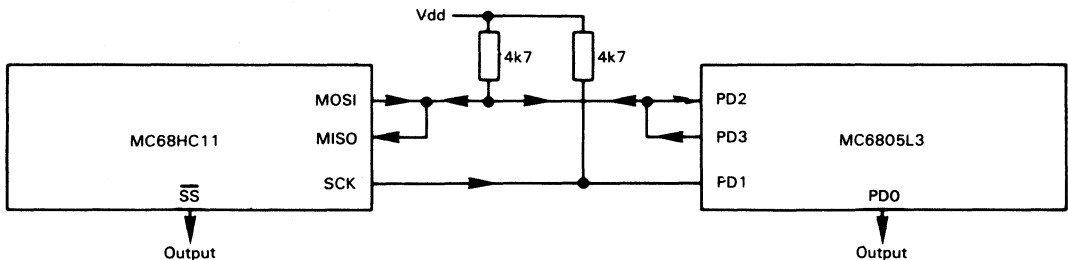
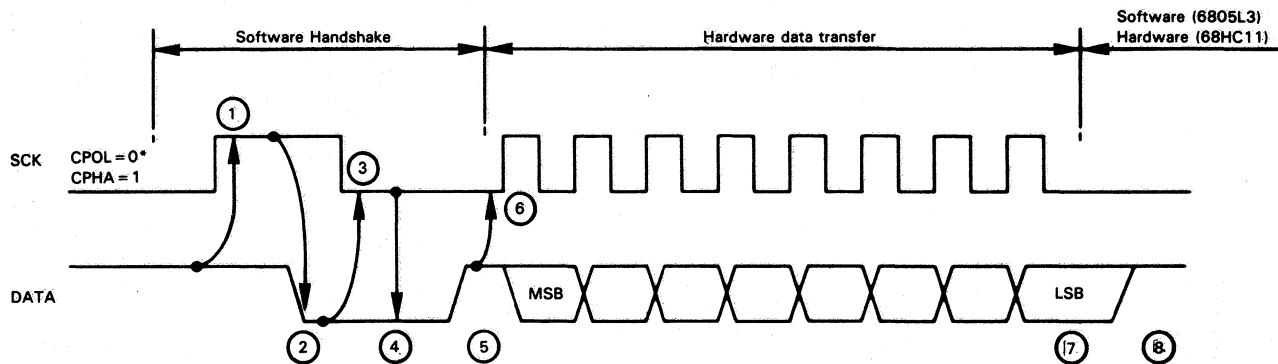


Figure 1. Hardware Implementation

Figure 2. 68HC11-6805L3 SPI Timing



*Clock control bits for 68HC11

- ① Master releases clock line by disabling SPI.
- ② Slave clears data line by forcing o/p clamp on PD3.
- ③ Master clears clock line by enabling SPI.
- ④ Once clock line goes low, slave stores data in SPI data register, sets its data DDR to correct state, and enables SPI.
- ⑤ Slave releases data clamp.
- ⑥ Master starts SPI clock by storing data in SPI register.
- ⑦ Both master and slave detect end of transmission and read data from SPI register.
- ⑧ Slave disables SPI to ensure data line is released.

Configuring the MCUs

The method of configuring each MCU for bi-directional data transfer is slightly different, due to the differences in their SPI silicon implementation. Figure 1 shows the hardware implementation. On the 68HC11, the input and output pins must be connected together externally. On the 6805L3, this can be done internally by software, thus requiring only 1 external data I/O pin. The other 6805L3 SPI data pin is now free to be used in any other way.

As the slave select (SS) pins are unused by either MCU, they must be configured as outputs to prevent SPI fault conditions occurring.

The spare 6805L3 data pin (PD3) is used to control the bi-directional data line, thus providing a handshake signal to the 68HC11. The 68HC11's handshake is on the clock line, and is controlled by disabling and enabling its SPI.

Data transfer and timing

Figure 2 shows details of the handshake sequence. The significant point to note from Figure 2 is that the handshake sequence is implemented purely in software, while the 8-bit data and clocks are generated by the SPI hardware.

To prevent data contention, both during data transfer and during the handshake sequence, both SPIs must operate in wired-or (open drain) mode. On the 6805L3, this is done by setting bit 3 in the miscellaneous register, while on the 68HC11, bit 5 of the SPI control register must be set. The SPI utilised on the 68HC05 family of MCUs does not support this open-drain option and cannot, therefore, be used in this application.

The clock format is: idle low, data output on positive edge and sampled on negative edge. Both SPIs must be configured to operate with the same clock format (see Figure 2). Eight data bits are transferred, at a maximum clock rate of 125 kHz,

which is limited by the 6805L3 SPI. Data transfer is preceded by the handshake sequence, which ensures that both MCUs are in the correct state, and ready to transfer a new byte of data.

The most significant bit of data appears first, on the rising edge of the first clock. Data is latched into both SPI shift registers on the falling edge of the clock. Once the last data bit is latched, the data line is released high. This is necessary to ensure correct operation of the handshake sequence. When the 68HC11 is acting as a transmitter, this state occurs automatically — as a by product of its SPI hardware implementation. However, on completion of data transmission from the 6805L3, the LSB is permanently maintained on the data line, so its driver routine has been designed to ensure that the data line is always restored to the high state.

Software routines

A glance at the software listing (see Appendix 1) reveals that the transmit and receive routines for each MCU are essentially the same! The entry and exit conditions of each are as shown in table 1.

On the 6805L3, the X register dictates the operating mode of the transfer routine. This is necessary because the same I/O pin is used for transmitting and receiving data, so its data direction register must be changed appropriately (by the contents of X).

On the 68HC11, separate pins are used for transmitting and receiving data, so their DDR pins are set up once only in the initialisation routine.

There are other important subtle differences. Prior to reception of data, the 6805L3 SPI data register content is irrelevant, while the 68HC11 SPI data register must be loaded with \$FF, to prevent data bus contention. This could occur with the 68HC11, in this application, because data is simultaneously output to and read back on the same external line

Table 1

68HC11:

	Transmit ACCA	Receive ACCA	
Entry	Data to send	\$FF	X Reg = base address of I/O Register block (normally \$1000).
Exit	Data sent	Data received	X Reg = unchanged

All other registers are unused by the 68HC11 transfer routine.

6805L3:

	Transmit		Receive	
	ACC	X Reg	ACC	X Reg
Entry	Data to send	\$5	Don't care	\$1
Exit	Data sent	\$D	Data received	\$D

during data transfer. Loading \$FF into the 68HC11 data register ensures that this data is replaced by that transmitted from the 6805L3.

Note that, as the 68HC11 is the clock master, it must provide the clock signal, not only when transmitting data, but also when receiving data from the 6805L3. It does this by writing to its SPI data register.

Completion of data transfer is indicated by a single flag bit (SPIF). On the 6805L3, this is bit 7 of the SPI control register, while on the 68HC11, it is bit 7 of the SPI status register. This flag bit is used to indicate completion of either transmission or reception of data.

Flag clearing techniques are quite different for the 68HC11 and 6805L3. On the latter, the SPIF flag is cleared simply by writing '0' to the flag bit. On the 68HC11, a two stage operation is required to clear the SPIF flag; the SPI status register must first be read, with the flag set, followed by an access of the SPI data register.

An examination of the SPI software drivers shows that on completion of a data transmission, the SPI data register is read again. This data should be the same as that transmitted, and provides information on whether data contention or corruption occurred during transfer. This facility could be incorporated in a data validation routine to improve reliability of data transfer.

The key features in this implementation are:

1. An orderly start-up sequence to ensure the correct initial synchronisation. As the 6805L3 is the slave, its initialisation routine is not exited until it detects a low on the clock line — this will occur only when the 68HC11 gains control of the SPI. Before this happens, all I/O pins are set to inputs, so the clock line will be pulled high by the external resistor.
2. A well defined transfer protocol is used. The master device (i.e. 68HC11) must always dictate the data transfer direction and the data stream size must be specified by the currently selected transmitter, so that the receiver knows when the last byte has been sent.

The transfer protocol operates such that after initialisation, the master MCU (68HC11) transmits a control byte to the slave (6805L3). This control byte selects the subsequent data transfer direction and is either a slave listen address or a slave talk address. If it is a slave listen address, then the 6805L3 stays in receive mode. The next byte indicates the total number of bytes to be received, followed by the data stream (see Figure 3).

Once the last byte is transferred, the 6805L3 can await a new control byte, or alternatively it can return to some other task, such as processing the previously received data. Similarly, at this point, the

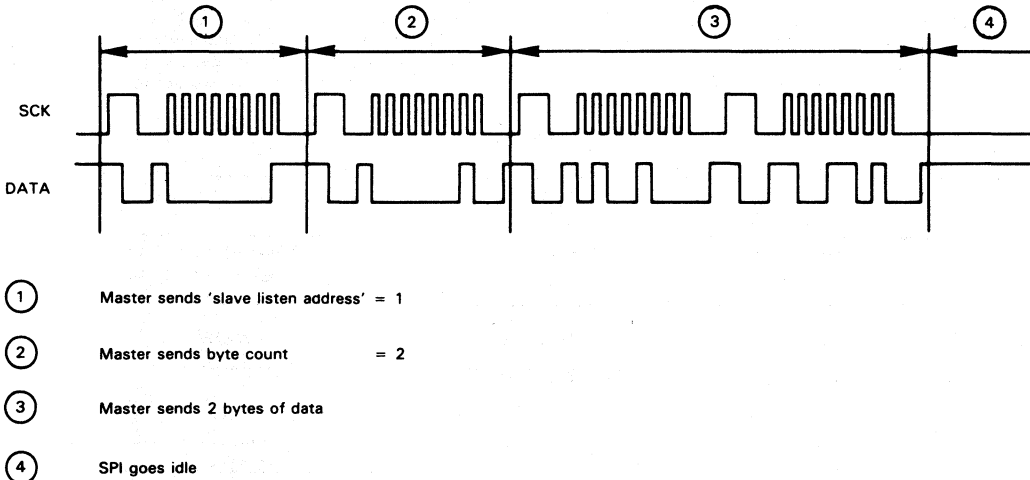


Figure 3. Master Transmitter — Slave Receiver

master transmitter (68HC11) can return to another task, or send a new control byte.

If the control byte now sent is a slave talk address, then the 6805L3 will switch to transmit mode, and the master will switch to receive mode. The 6805L3 will send a byte count, followed by the data stream to the master (see Figure 4).

Note that, once the last byte is transferred in either direction, both processors are free to continue other tasks or attempt a new data transfer. The handshake sequence always ensures synchronisation of data transfer, independent of the response time of either MCU.

CONCLUSION

Potential uses for this type of data transfer are in applications which require remote interrogation of an

MCU based system via a minimal number of lines, such as:

- Development and diagnosis of engine management systems
- Smart card and key card security applications
- Instrumentation and data logging equipment.

APPENDIX

The demonstration programs listed in the following pages simply transfer a string of characters from the 68HC11 to the 6805L3, which converts them to upper case and sends them back again.

The HC11SPI program is listed on pages 6 to 7; the L3SPI program on pages 8 to 10.

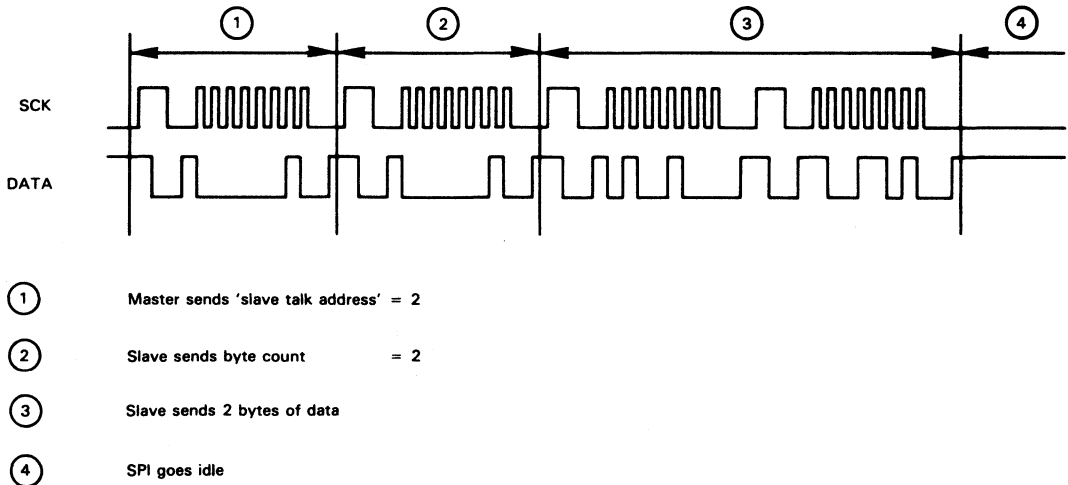


Figure 4. Master Receiver – Slave Transmitter

```

1 P *****
2 P *
3 P *           HC11SPI 4/9/86           *
4 P *
5 P * SPI TRANSFER WITH HANDSHAKE ON SPI DATA AND *
6 P * CLOCK LINES. THIS IS NECESSARY AS HC11'S CLOCK *
7 P * CANNOT BE SLOWED BY SLAVE DEVICE. *
8 A 0008 PORTD EQU 8
9 A 0009 DDRD EQU 9
10 A 0028 SPCR EQU $28
11 A 0029 SPSR EQU $29
12 A 002A SPDR EQU $2A
13 A 0024 TMSK2 EQU $24
14 A 0025 TFLG2 EQU $25
15 P *
16 A 0004 MISO EQU 4
17 A 0008 MOSI EQU 8
18 A 0010 SCK EQU $10
19 A 0080 SPIF EQU $80
20 A 0040 SPE EQU $40
21 P *
22 A 0000 ORG $0
23 A 0000 0001 TIMCOUNT RMB 1
24 A 0001 1C MSG1 FCB 28
25 A 0002 62692D6469 FCC /BI-DIRECTIONAL DATA TRANSFER/
26 A *
27 A C000 ORG $C000
28 A C000 8E0035 START LDS #35
29 A C003 8D3C BSR INIT
30 A C005 18CE0001 START1 LDY #MSG1
31 A C009 8D08 BSR SEND
32 A C00B 18CE0001 LDY #MSG1
33 A C00F 8D17 BSR READ
34 A C011 20F2 BRA START1
35 A *
36 A C013 SEND EQU *
37 A C013 8601 LDAA #1
38 A C015 8D3A BSR XFER
39 A C017 18A600 LDAA .Y
40 A C01A 16 TAB
41 A C01B 8D34 BSR XFER
42 A C01D 1808 INY
43 A C01F 18A600 LDAA .Y
44 A C022 8D2D BSR XFER
45 A C024 5A DECBB
46 A C025 26F6 BNE SEND1
47 A C027 39 RTS
48 A *
49 A C028 READ EQU *
50 A C028 8602 LDAA #2
51 A C02A 8D25 BSR XFER
52 A C02C 86FF LDAA #$FF
53 A C02E 8D21 BSR XFER
54 A C030 16 TAB
55 A C031 18A700 STAA .Y
56 A C034 1808 INY
57 A C036 86FF LDAA #$FF
58 A C038 8D17 BSR XFER

```

NUMBER OF BYTES IN DATA BLOCK
/BI-DIRECTIONAL DATA TRANSFER/

LOAD INTO EVB RAM
BUFFALO USES RAM ABOVE THIS

Y POINTS TO BEGINNING OF DATA BLOCK TO TRANSFER
TRANSMIT BLOCK TO SLAVE, STARTING WITH BYTE COUNT
RE-INITIALISE POINTER TO DATA BLOCK
AND READ THE BYTES BACK.

ENTER WITH Y POINTING AT BYTE COUNT OF DATA BLOCK.
COMMAND SLAVE TO RECEIVE

NOW SEND SLAVE THE BYTE COUNT.
BUT 1ST STORE IT IN BYTE COUNTER.

POINT AT NEXT BYTE

AND SEND IT
UNTIL ALL DONE

ENTER WITH Y POINTING AT BYTE COUNT OF DATA BLOCK.
COMMAND SLAVE TO TRANSMIT

NOW READ BYTES BACK FROM SLAVE.
1ST BYTE IS BYTE COUNT SO
STORE IT IN BYTE COUNTER
THEN STORE IT AT BEGINNING OF DATA BLOCK.
BUMP POINTER

THEN CONTINUE TO READ

```

59 A C03A 18A700      STAA    .Y          AND STORE BYTES
60 A C03D 5A         DECB
61 A C03E 26F4      BNE     READ1      UNTIL ALL DONE
62 A C040 39        RTS
63 A
64 A          CO41    *
        *          INIT  EQUJ    *
65 A C041 CE1000    LDX     #$1000
66 A C044 8638      LDAA   #$38        ENABLE OUTPUT MODE ON SS,SCK,MOSI, INPUT ON MISO
67 A C046 A709      STAA   DDRD,X     (WITH SS AS OUTPUT, MODF IS DISABLED.)
68 A C048 8676      LDAA   #$76        ENABLE SPI AS 125KHZ CLOCK MASTER, WIRED-OR MODE
69 A C04A A728      STAA   SPCR,X
70 A C04C 8618      LDAA   #$18
71 A C04E A708      STAA   PORTD,X    SET DATA & CLOCK OUTPUT BUFFERS TO LOGIC '1'
72 A C050 39        RTS              TO GENERATE ACKNOWLEDGE CLOCK.
73 A
74 A
75 A
76 A
77 A
78 A
79 A
80 A          C051    *
        *          XFER  EQUJ    *
81 A C051 1F0804FC  BRCLR  PORTD,X,#MISO,* 1ST WAIT FOR SLAVE TO RELEASE DATA LINE.
82 A C055 1D2840   BCLR   SPCR,X,#SPE     SEND ACK ON CLOCK LINE, BY DISABLING SPI
83 A C058 1E0804FC  BRSET  PORTD,X,#MISO,* WAIT FOR SLAVE TO ACKNOWLEDGE.
84 A C05C 1C2840   BSET   SPCR,X,#SPE     ENABLE SPI, CLEARING CLOCK LINE.
85 A C05F 1F0804FC  BRCLR  PORTD,X,#MISO,* WAIT FOR SLAVE TO RELEASE DATA LINE, THEN
86 A C063 A72A     STAA   SPDR,X         START SPI TRANSACTION BY WRITING DATA.
87 A C065 1F2980FC  BRCLR  SPSR,X,#SPIF,* NOW WAIT FOR TRANSMISSION COMPLETE FLAG.
88 A C069 A62A     LDAA   SPDR,X         CLEAR SPIF, AND READ DATA
89 A C06B 39        RTS              BEFORE RETURNING.
90 A
91 A
92 A          *          ORG $FFFE
93 A          *          FDB START
94 A          *
        *          END

```

```

***** TOTAL ERRORS 0-- 0
***** TOTAL WARNINGS 0-- 0

```

SYMBOL TABLE LISTING

SYMBOL NAME	SECT	VALUE	SYMBOL NAME	SECT	VALUE
DDRD	A	0009	SPCR	A	0028
INIT	A	C041	SPDR	A	002A
MISO	A	0004	SPE	A	0040
MOSI	A	0008	SPIF	A	0080
MSG1	A	0001	SPSR	A	0029
PORTD	A	0008	START	A	C000
READ	A	C028	START1	A	C005
READ1	A	C034	TFLG2	A	0025
SCK	A	0010	TIMCOUNT	A	0000
SEND	A	C013	TMSK2	A	0024
SEND1	A	C01D	XFER	A	C051


```

1 P
2 P
3 P
4 P
5 P
6 P
7 P
8 P
9 A      0000   PORTA   EQU     0
10 A     0003   PORTD   EQU     3
11 A     0004   DDRA    EQU     4
12 A     0007   DDRD    EQU     7
13 A     0008   TADAT   EQU     8
14 A     0009   TACR    EQU     9
15 A     000A   MISC    EQU     $A
16 A     000C   TBDAT   EQU     $C
17 A     000D   TBCR    EQU     $D
18 A     000E   SPIDAT  EQU     $E
19 A     000F   SPTCR   EQU     $F
20 A     0010   PRESCL  EQU     $10
21 P
22 A     0000   SS      EQU     0
23 A     0001   SCK     EQU     1
24 A     0002   SDA     EQU     2
25 A     0003   CLAMP   EQU     3
26 A     0004   SPE     EQU     4
27 A     0007   SPIF    EQU     7
28 P
29 A     0020                   ORG     $20
30 A 0020 0001   BYTCNT  RMB     1
31 A 0021 0001   DATLEN  RMB     1
32 A 0022 0028   DATA   RMB    40
33 A
34 A     0080                   ORG     $80
35 A     0080   START   EQU     *
36 A 0080 AD3D   BSR     CLRA
37 A 0082 4F   START1  LDX     #1
38 A 0083 AE01   BSR     XFER
39 A 0085 AD4C   CMP     #1
40 A 0087 A101   BEQ    READ
41 A 0089 2706   CMP     #2
42 A 008B A102   BEQ    SEND
43 A 008D 271A   BRA    START1
44 A 008F 20F2
45 A
46 A     0091   READ    EQU     *
47 A 0091 AE01   LDX     #1
48 A 0093 AD3E   BSR     XFER
49 A 0095 B720   STA    BYTCNT
50 A 0097 B721   STA    DATLEN
51 A 0099 AE01   READ1  LDX     #1
52 A 009B AD3E   BSR     XFER
53 A 009D BE20   LDX    BYTCNT
54 A 009F A020   SLB    #$20
55 A 00A1 E722   STA    DATA,X
56 A 00A3 3A20   DEC    BYTCNT
57 A 00A5 26F2   BNE    READ1
58 A 00A7 20DA   BRA    START1

```

L3SPI 4/9/86

HANDSHAKED BIDIRECTIONAL DATA TRANSFER
BETWEEN 6805L3 & 68HC11 USING SPI.
HC11 IS CLOCK MASTER.

DATA LINE CLAMP ON D3

RESERVE ENOUGH SPACE FOR RECEIVED DATA.

INITIALISE SPI, AND WAIT FOR HC11 TO BECOME READY

1ST RECEIVE COMMAND BYTE

IF SLAVE REQUESTED TO LISTEN, THEN
READ MORE DATA.

IF SLAVE REQUESTED TO TALK, THEN
SEND DATA TO MASTER.

GET NEXT BYTE, WHICH IS BYTE COUNTER.

ALSO STORE VALUE AS DATA LENGTH.
NOW READ ALL REMAINING BYTES

(CONVERT LOWER CASE TO UPPER CASE)
AND STORE IN BUFFER

UNTIL ALL DONE,
AND RETURN.

```

59 A
60 A      00A9      * SEND EQU *
61 A 00A9 B621 LDA DATLEN
62 A 00AB B720 STA BYTCNT
63 A 00AD AE05 LDX #5
64 A 00AF AD22 BSR XFER
65 A 00B1 BE20 SEND1 LDX BYTCNT
66 A 00B3 E622 LDA DATA,X
67 A 00B5 AE05 LDX #5
68 A 00B7 AD1A BSR XFER
69 A 00B9 3A20 DEC BYTCNT
70 A 00BB 26F4 BNE SEND1
71 A 00BD 20C4 BRA START1
72 A
73 A      00BF      * INIT EQU *
74 A 00BF A648 LDA #$48
75 A 00C1 B70A STA MISC
76 A 00C3 A60D LDA #$D
77 A 00C5 B703 STA PORTD
78 A 00C7 A609 LDA #9
79 A 00C9 B707 STA DDRD
80 A 00CB A644 LDA #$44
81 A 00CD B70F STA SPICR
82 A 00CF 0203FD BRSET SCK,PORTD,*
83 A 00D2 81 RTS
84 A
85 A
86 A
87 A
88 A
89 A
90 A
91 A
92 A
93 A
94 A      00D3      * XFER EQU *
95 A 00D3 0303FD BRCLR SCK,PORTD,*
96 A 00D6 1703 BCLR CLAMP,PORTD
97 A 00D8 0203FD BRSET SCK,PORTD,*
98 A 00DB B70E STA SPIDAT
99 A 00DD 180F BSET SPE,SPICR
100 A 00DF BF07 STX DDRD
101 A
102 A 00E1 A644 LDA #$44
103 A 00E3 AE0D LDX #D
104 A 00E5 0F0FFD BRCLR SPIF,SPICR,*
105 A 00E8 B70F STA SPICR
106 A 00EA B60E LDA SPIDAT
107 A 00EC BF03 STX PORTD
108 A 00EE BF07 STX DDRD
109 A 00F0 81 RTS
110 A
111 A      END

```

GET LENGTH OF DATA AND STORE IT IN BYTE COUNTER.
 SEND BYTE COUNT TO MASTER. NOW GET NEXT BYTE TO SEND IN ACC
 SEND IT TO MASTER
 UNTIL ALL DONE, AND RETURN.
 INHIBIT INT2. ENABLE PORTD OPEN DRAIN.
 SET DATA,SS & CLAMP O/P BUFFERS. CLEAR REST. NOTE: CLAMP ON D3
 SELECT SPI CLOCK SLAVE, D2 AS I/P, CLAMP & SS AS O/P. (SS O/P STOPS SPI RESETTING)
 DISABLE START BIT DETECTION, DATA I/O ON D2, DATA SAMPLED ON -IVE CLOCK, SPI DISABLED. MUST WAIT FOR MASTER TO GAIN CONTROL OF SPI.

 BIDIRECTIONAL DATA TRANSFER ON SPI
 ENTERED WITH SPI DISABLED, DATA PIN HIGH*
 D2 PIN I/P, CLAMP PIN O/P (HIGH)
 ENTRY: TX MODE: ACCA=DATA, X=5
 RX MODE: ACCA=X, X=1
 EXIT: TX MODE: ACCA=TX DATA, X=\$D
 RX MODE: ACCA=RX DATA, X=\$D

 WAIT FOR MASTER TO ACKNOWLEDGE ON CLOCK LINE. SEND ACKNOWLEDGE TO MASTER.
 WAIT FOR MASTER TO ENABLE ITS SPI
 NOW WRITE DATA TO SPI REGISTER, BEFORE ENABLING SPI, AND RELEASING CLAMP CLAMP PIN I/P: DATA EITHER I/P OR O/P.
 TRANSFER STARTS NOW!
 PREPARE TO CLEAR SPI FLAG, DISABLE SPI, AND SEND ACKNOWLEDGE.
 WAIT FOR DATA TO ARRIVE.
 DISABLE SPI TO ALLOW DATA PIN TO BE FORCED HIGH
 READ DATA.
 FORCE DATA PIN HIGH & RELEASE CLAMP BY MAKING BOTH OUTPUTS.

***** TOTAL ERRORS 0-- 0
 ***** TOTAL WARNINGS 0-- 0

SYMBOL TABLE LISTING

SYMBOL NAME	SECT	VALUE	SYMBOL NAME	SECT	VALUE
BYTCNT	A	0020	SEND	A	00A9
CLAMP	A	0003	SEND1	A	00B1
DATA	A	0022	SPE	A	0004
DATLEN	A	0021	SPICR	A	000F
DDRA	A	0004	SPIDAT	A	000E
DDRD	A	0007	SPIF	A	0007
INIT	A	00BF	SS	A	0000
MISC	A	000A	START	A	0080
PORTA	A	0000	START1	A	0083
PORTD	A	0003	TACR	A	0009
PRESCL	A	0010	TADAT	A	0008
READ	A	0091	TBCR	A	0000
READ1	A	0099	TBDAT	A	000C
SCK	A	0001	XFER	A	0003
SDA	A	0002			

MC68HC805B6 LOW-COST EEPROM MICROCOMPUTER PROGRAMMING MODULE

Prepared by: Bill Mathews, Product Engineer, Motorola Semiconductors Ltd, East Kilbride, Scotland

INTRODUCTION

The EEPROM feature of the MC68HC805B6 microcomputer unit (MCU) enables the user to emulate the MC68HC05B6 and MC68HC05B4 MCU devices. This application notes describes one programming technique which may be used to program the MC68HC805B6 internal EEPROM, and provides a description of the programming module used in conjunction with this application note. All that is required to program the MC68HC805B6 is the programming module (which can be implemented on a single PCB), and a twin +5V/+19V dc power supply.

PROGRAMMING TECHNIQUE

A multi-byte EEPROM programming technique is used to load a user program into the MC68HC805B6 EEPROM in order to emulate the MC68HC05B6 and MC68HC05B4 devices. This type of operation is accomplished via a bootstrap mode of operation. The user program contained in an external EPROM is copied into the internal EEPROM of the MC68HC805B6 device, then verified against the contents of the EPROM.

The bootstrap program follows the following sequence following reset:

1. EEPROM6 Erase.

The 6K EEPROM (which emulates the ROM on the MC68HC05B4 and MC68HC05B6 mask devices) is erased for a nominal 100 ms (with a 4MHz crystal). This memory area is then tested for complete erasure by verifying that all bytes in the EEPROM6 map read \$FF. If any bytes have failed to erase, then the red LED will be illuminated and the bootstrap routine will cycle back and continue to erase EEPROM6 until all bytes are erased.

2. EEPROM1 Erase.

The 256 bytes EEPROM (which emulates the 256 bytes of byte-eraseable EEPROM on the MC68HC05B6 mask device) is erased in a similar fashion to the EEPROM6, with illumination of the red LED indicating a failure to erase. Completion of this program step ensures that the security bit (which is implemented in EEPROM1) is also erased.

3. EEPROM Program.

EEPROM1 and EEPROM6 are then programmed in turn from the data contained in the EPROM, in increasing address order. Areas in the MC68HC805B6 memory map which do not include EEPROM will be skipped by the programming routine, as will \$FF bytes, thus speeding the programming operation. During programming the green LED should flash at approximately 3Hz, though this frequency will increase for data equal to \$FF.

4. EEPROM Program Verify

EEPROM1 and EEPROM6 are verified against the contents of the EPROM, with any error causing illumination of the red LED. A successful programming operation will result in green LED on, red LED off. Note that the red LED may glow very dimly, this is normal.

PROGRAMMING OPERATION

To program the MC68HC805B6 MCU EEPROM, perform the following steps.:

1. With power to the module removed install MCU and EEPROM devices into the programming module.
2. Place switch S1 to RESET position.
3. Apply +5V power supply to the programming module.
4. Apply +19V power supply to the programming module.
5. Place switch S1 in the RUN position.
6. Once the green LED is illuminated, place switch S1 to RESET position.
7. Remove +19V power supply from the programming module.
8. Remove +5V power supply from the programming module.

Note: To avoid possible damage to the MC68HC805B6 it is essential that power to the programming module is applied and removed in the sequence specified above.

PROGRAMMING MODULE CONSTRUCTION

Table 1 provides the parts list for construction of the module, with component tolerances generally not critical. A schematic of the circuit is included as Figure 1.

HARDWARE CONSIDERATIONS/DEBUG

Functionality of the module may be adversely affected by faults in the following areas:

1. IRQ and RESET

The circuitry on these pins ensures that the IRQ pin sees a voltage level equal to approximately +10V dc during the RESET period. This is necessary for bootstrap mode capture. Correct operation of D1 and D2 should be verified.

2. VPP6

The input to this pin is controlled via a switch driven by port pin PC7. This method ensures that a minimum +5V dc level is always present at the VPP6 pin, and that +19V dc is available during programming and erase operations on EEPROM6. Correct operation of D5, 6 and 7 should be verified. Zener diode ZD1 protects the EEPROM from

possible damage caused by excursions greater than +20V dc.

3. VPP1

It is important that this pin (at which the output of the internal EEPROM1 charge pump is visible) is allowed to float. Clamping this pin to +5V dc will prevent successful programming and erase operations from taking place on EEPROM1, whilst clamping this pin to 0V could damage the device.

4. Port D

For correct bootstrap mode capture it is necessary to connect these inputs to 0V.

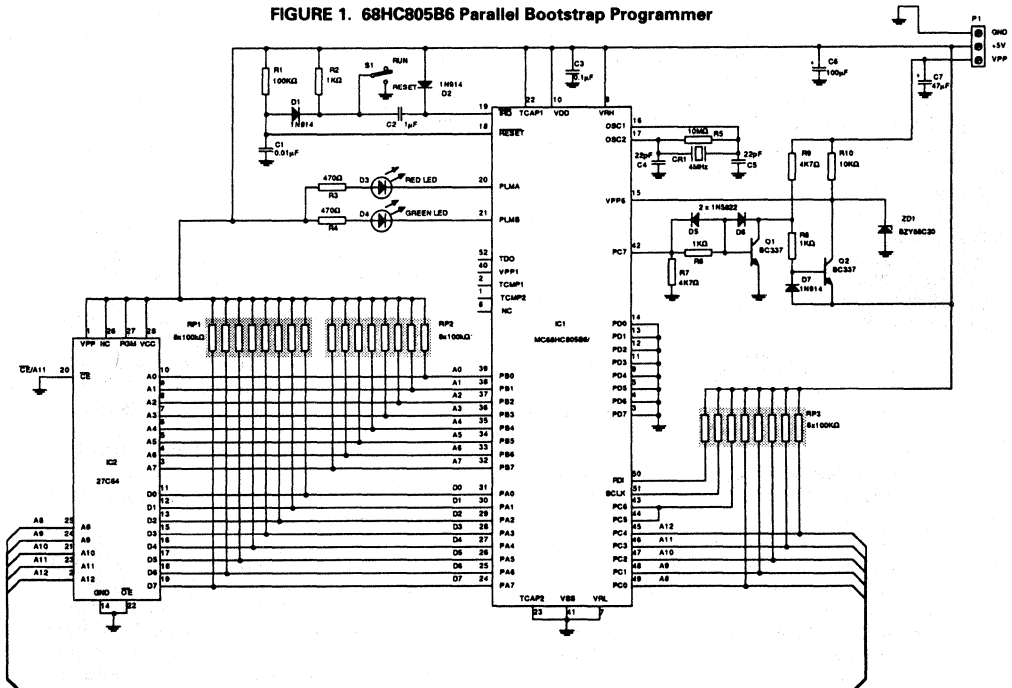
5. PC5/PC6

These pins provide a handshake which is not used in this application, therefore for correct operation of the module PC5 should be connected to PC6.

TABLE 1. 68HC805B6 EEPROM Programming Module Parts List

Resistors		Diodes	
R1	100K	D1,D2	1N914
R2	1K	D3	LR3160 Red LED
R3,R4	470	D4	LG3160 Green LED
R5	10M	D5,D6	1N5822
R6	1K	D7	1N914
R7	4K7	ZD1	BZY88C20
R8	1K		
R9	4K7	Sockets	
R10	10K	IC1	52 pin plcc zif
RP1-RP3	100K	IC2	28 pin lif
			3M textool
Capacitors		Connectors	
C1	0.01µF	P1	3 way terminal connector
C2	1.0µF		
C3	0.1µF	Switches	
C4,C5	22pF	S1	2 way, toggle
C6	100µF		
C7	47µF	Miscellaneous	
Transistors		CR1	4MHz Crystal
Q1,Q2	BC337-25		

FIGURE 1. 68HC805B6 Parallel Bootstrap Programmer



Monitor Program for the MC68HC05B6 Microcomputer Unit

Prepared by: Bill Mathews, Product Engineer, Motorola Semiconductors Ltd, East Kilbride, Scotland

INTRODUCTION

The MC68HC05B6 HCMOS microcomputer is a member of Motorola's MC68HC05 family of low-cost single-chip microprocessors. This 8-bit microcomputer (MCU) contains an on-chip oscillator, CPU, RAM, ROM, EEPROM, A/D, PULSE LENGTH Modulated outputs, I/O, a serial Communications Interface, Timer system and Watchdog timer.

A monitor program is available in the mask ROM of a 68HC05B6, (XC68HC05B6FN MONITOR), which when used with a monitor circuit module, a power supply, and a video terminal will allow the user to write and debug small portions of 68HC05B6 code. This application note contains a description of the facilities available via the monitor software, a diagram of the monitor circuit, and a listing of the monitor code.

HARDWARE

The monitor module requires a single(+5V) power supply, and all communications between the device and terminal take place via an RS-232 link. All 68HC05B6 I/O pins are available to the user to be configured as required.

Terminal setup is as follows:

9600 Baud, Half Duplex,
and either 7 bit data and parity '0'
or 8 bit data and no parity.

A circuit diagram of the monitor is given in Figure 1.

MONITOR OPERATION

The following sequence of operations should be followed:

1. Remove power supply from module
2. Insert XC68HC05B6FN MONITOR device
3. Place S1 in 'RESET' position
4. Apply +5V dc power supply
5. Connect video terminal
6. Place S1 in 'RUN' position

The 68HC05B6 will then begin to execute the monitor program, and the following message should appear on the monitor:

"Hi! I'm the MC 68HC05 B6 from MOTOROLA

European Design Centre, Geneva."

A prompt "." will be displayed to indicate that the device is ready to receive commands from the terminal. If no message appears, then the setup of the terminal should be verified.

MONITOR COMMANDS

The following commands are available:

Command Description

R Display the content of the registers in format.

HINZC AA XX PPPP ZZ = DD where

HINZC	=	Condition code register
AA	=	Contents of Accumulator
XX	=	Contents of Index Register
PPPP	=	Program Counter
ZZ,DD	=	User specified byte (which addresses f00 to fFF) and contents. Set up using the 'V' command. Reset initialises ZZ to f08 (A/D data register).

Note that this command assumes that the stack pointer is at address \$FA.

A Display/Change the Accumulator

The contents of the Accumulator are displayed, then the monitor waits for input of two hex digits (new accumulator contents). Typing a carriage return will return the program to the command mode with the contents of the Accumulator unchanged.

Note that this command assumes that the Stack Pointer is at address \$FA.

X Display/Change the Index Register

The contents of the Index Register are displayed, then the monitor waits for the input of two hex digits (new Accumulator contents). Typing a carriage return will return

the program to the command mode with the contents of the Index Register unchanged.

Note that this command assumes that the Stack Pointer is at address \$FA.

M nnnn Examine/Change Memory

The contents of any address in the range \$000 to \$1FF (Register, RAM and EEPROM1) are displayed, and the program will await further input, namely:

- .
 - ^
 - <CR>
 - +
 -
 - /D
 - nn
- Redisplay the contents of the current address
 Display the contents of the previous address (nnnn-1)
 Open the next address (nnnn+1)
 Increase the contents of the open location by 1
 Decrease the contents of the open location by 1
 Replace the contents of the open location with the Ascii code for alphanumeric character D, and go to the next address.
 Replace the contents of the currently open address with two hex digits nn and go to the next address

Typing any other character will return the monitor to the command mode.

L nnnn List a block of memory starting at address nnnn. The default address (if nnnn is not specified) is \$100. The data is displayed on screen as four blocks of eight by eight bytes, with the address printed at every sixteenth byte.

V nn Change the address of the page zero byte displayed with the R command with the hex byte specified by nn.

K nn Set Program and Erase times for operations on EEPROM1 where nn is milliseconds entered in decimal. The default values are 10ms. Typing 'enter' will skip the command.

P nn Program the entire EEPROM1 array (except

address \$100) with nn. This command may take some time to execute as firstly the entire array is erased, then each byte is programmed in turn.

If non-hex data is entered, then the following commands are available:

- Z** Program the entire array (except address \$100) with Data = Address, i.e.
 Address \$100 = not programmed
 Address \$101 = \$01
 Address \$102 = \$02
 Address \$103 = \$03 ...etc.
- P** Program a chequer-board pattern
- Q** Program an inverse chequer-board pattern

- E nnnn** Start execution at address nnnn
- C** Continue program execution according to the current program counter, accumulator, index register and condition code register stored on the stack.

BREAKPOINTS AND INTERRUPTS

The SWI instruction may be used as a breakpoint. To continue following a breakpoint first replace the SWI with another command (such as NOP) then type 'C' to continue.

The interrupt vectors point to the RAM as shown below, and are spaced three bytes apart allowing the use of a JMP or BRA instruction to a service routine located in either RAM or EEPROM1.

Vector	Address	
SCI	\$00DF	
Timer Overflow	\$00E2	
Timer O/P CMP	\$00E5	
Timer I/P CAP	\$00E8	
IRQ	\$00EB	
SWI	\$08A6	Pointing to monitor for breakpoint
RESET	\$0C22	Start of monitor code

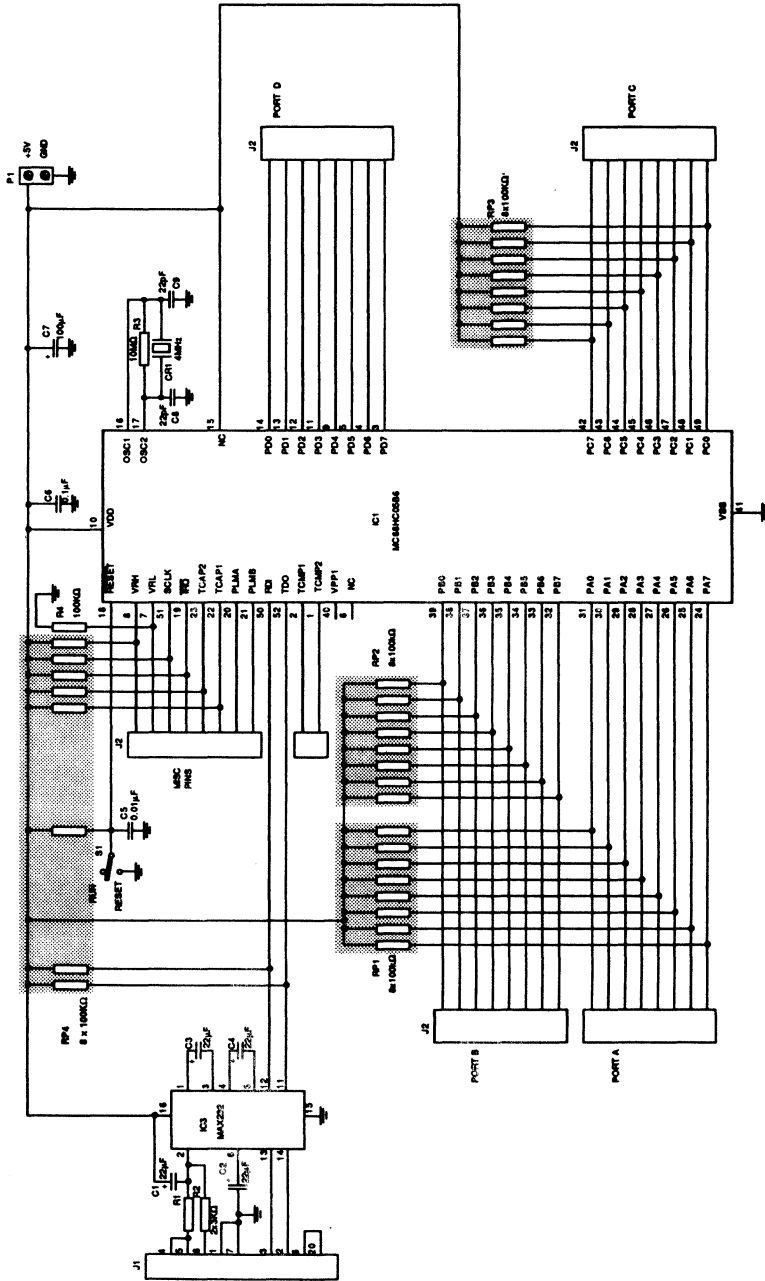


Figure 1 - 68HC05B6 Monitor Circuit Diagram

TTL MC68HC05B6 SELF-CHECK AND MONITOR
 OPT P=58, LLE=118, CRE

F I R M W A R E F O R T H E M C 6 8 H C 0 5 B 6
 =====

MONITOR LISTING ONLY
 =====

I/O and INTERNAL registers definition

I/O registers

PORTA	EQU	\$00	port A.
PORTB	EQU	\$01	port B.
PORTC	EQU	\$02	port C.
PORTD	EQU	\$03	port D.
DDRA	EQU	\$04	port A DDR.
DDRB	EQU	\$05	port B DDR.
DDRC	EQU	\$06	port C DDR.

EEPROM register

EECONT	EQU	\$07	EEPROM control register.
.E1PGM	EQU	0	
.E1LAT	EQU	1	
.E1ERA	EQU	2	

A/D registers

ADDATA	EQU	\$08	A/D data register.
ADSTCT	EQU	\$09	A/D status and control register.
.COCO	EQU	7	Conversion complete flag.

PLM registers

PLMA	EQU	\$0A	pulse length mod reg A.
PLMB	EQU	\$0B	pulse length mod reg B.

Miscellaneous register

MISC	EQU	\$0C	Miscellaneous register.
.WDOG	EQU	0	Watchdog control bit.
.SM	EQU	1	Slow Mode.

SCI registers

BAUD	EQU	\$0D	SCI baud register.
SCCR1	EQU	\$0E	SCI control register 1.
SCCR2	EQU	\$0F	SCI control register 2.
.SBK	EQU	0	Send break bit.
SCSR	EQU	\$10	SCI status register.

```

.RDRF EQU 5 Receive data register full flag.
SCDAT EQU $11 SCI data register.
*
* TIMER registers
*
TIMCTL EQU $12 Timer control register.
.TOIE EQU 5 Timer overflow interrupt enable.
.OCIE EQU 6 Timer output compares interrupt enable.
.ICIE EQU 7 Timer input captures interrupt enable.
TIMST EQU $13 Timer status register.
.OCF2 EQU 3 Timer output compare 2 flag.
.ICF2 EQU 4 Timer input capture 2 flag.
.TOF EQU 5 Timer overflow flag.
.OCF1 EQU 6 Timer output compare 1 flag.
.ICF1 EQU 7 Timer input capture 1 flag.
TIMIC1 EQU $14 Timer input capture register 1 (16-bit).
TIMOC1 EQU $16 Timer output compare register 1 (16-bit).
TIMCTR EQU $18 Timer free running counter (16-bit).
TIMALT EQU $1A Timer alternate counter register (16-bit).
TIMIC2 EQU $1C Timer input capture register 2 (16-bit).
TIMOC2 EQU $1E Timer output compare register 2 (16-bit).
*
* MEMORY MAP DEFINITION
*
*
TEST EQU $20 TEST register
ROM0 EQU $0020 Start address of ROM0.
RAM EQU $0050 Start address of RAM.
EEPROM EQU $0100 Start address of EEPROM 256 bytes. NOT USE IN B4
.SEC EQU 0 Security bit.
ROM1 EQU $0200 Start address of ROM1.
ROM1ND EQU $02BF End address of ROM1.
ROM2 EQU $0800 Start address of ROM2. $0F00 IN B4
ROMBOT EQU $1F00 Start address of bootstrap ROM2.
*
* Miscellaneous definitions and equates
*
*
RAM1 EQU RAM+1 Working memory.
RAMINT EQU RAM+10 RAM location used for interrupt test.
VECT EQU $1FE2 Start of bootstrap vectors.
*
* Synthetized instructions
*
EOR EQU $D8 Exclusive or, 2 bytes indexed.
STA EQU $C7 Store A, extended mode.
PAGE
*****
*
* MONITOR PROGRAM
*
*
* Rev 1987/07/22

```


* Note : TEST is a write only register and its access is
 * authorized only when E1LAT is cleared. When E1LAT
 * is set, address \$20 (POEEP6) is accessed for writing.

* DEFINITION OF MONITOR VALUES

STACK	EQU	\$FA	STACK FOR MONIT
* Miscellaneous definitions and equates			
HI	EQU	0	hi byte offset
LO	EQU	1	lo byte offset
LONG	EQU	\$C4	long timing factor (100 ms nominal)
SHORT	EQU	\$14	short timing factor (10 ms nominal)
RED	EQU	PLMA	red LED on PLMA
GREEN	EQU	PLMB	green LED on PLMB
E1BW	EQU	7	bulk bit of TEST register
E6PGM	EQU	4	EECONT
E6LAT	EQU	5	EECONT
E6ERA	EQU	6	EECONT
TDRE	EQU	7	
MBIT	EQU	4	8 data bits flag in SCCR1
ADON	EQU	5	A/D converter control bit

* MONITOR DEFINITIONS

MPRT	EQU	'.
EPRT	EQU	'*
FWD	EQU	\$0D
BACK	EQU	'~
SAME	EQU	'.
PLUS	EQU	'+
MINUS	EQU	'-
ASCII	EQU	'/'
SPACE	EQU	\$20
CR	EQU	\$0D
LF	EQU	\$0A
BEEP	EQU	\$07
EOT	EQU	\$04

* MONITOR AND COMMON RAM DEFINITIONS

	ORG	RAM	
COUNT	RMB	1	BINBCD
CHAR	RMB	1	CURRENT INPUT/OUTPUT CHARACTER
XTEMP	RMB	1	TEMP FOR GETC,PUTC
ATEMP	RMB	1	TEMP FOR GETC,PUTC
MEMADD	RMB	1	MEMORY ADDRESS FOR REGS
FLAG	RMB	1	ITEM COUNTER & TEMP FOR EEPROM R/W
GET	RMB	4	FOR PICK AND DROP SUBROUTINES
WRITEK	RMB	1	CONSTANT FOR EEPROM WRITE TIME
ERASEK	RMB	1	CONSTANT FOR EEPROM ERASE TIME
ASC	RMB	1	/ FLAG

PAGE

MONITOR PROGRAM

ORG ROM0

FCC /Rev /
FCB REV
FCC ./.
FCB REL
FCC / /

* ONE ROUTINE IN PAGE 0 - JUST FOR THE (C)HECK OF IT

* PCC --- PRINT CONDITION CODE REGISTER

PCC LDA STACK+1 GET CCR
ASLA MOVE H BIT TO BIT 7
ASLA
STA GET SAVE IT
CLR
PCC2 LDA #'.
ASL GET PUT BIT IN CARRY BIT
BCC PCC3 BIT OFF MEANS PRINT '.'
LDA CCSTR,X PICKUP APPROPRIATE CHAR
PCC3 JSR PUTC PRINT '.' OR CHAR
INX POINT TO NEXT IN STRING
CPX #5 5 BITS ARE GOOD ENOUGH
BLO PCC2
RTS

* Fill page 0 ROM

FCC /0123456789AB/

PAGE

ORG ROM2

* NOW, OTHER ROUTINES IN MAIN EEPROM

* TABLES

BBTBL FCB \$A,\$14,\$28,\$50

```

CCSTR FCC    /HINZC/
*
MSG      FCC    /Hi ! I'm the MC 68HC05 B6 /
          FCC    /From MOTOROLA European /
          FCC    /Design Center, Geneva/
          FCB    EOT
*
CMA      EQU    *          COMMON MESSAGE AREA
NGM      FCB    '? ,BEEP,CR,LF,EOT
ERA      FCC    /Erase : /
          FCB    EOT
WRI      FCC    /Write : /
          FCB    EOT
MS       FCC    / ms /
          FCB    EOT
*
*
* SETA --- EXAMINE / CHANGE ACCUMULATOR
*
SETA     LDH    #STACK+2 POINT TO A
          BRA    SETANY
*
* SETX --- EXAMINE / CHANGE INDEX
*
SETX     LDH    #STACK+3 POINT TO X
*
* SETANY - PRINT £X| AND CHANGE IF REQUESTED
*
SETANY   LDA    ,X          PICK UP THE DATA
          JSR    PUTBYT     AND PRINT IT
          JSR    PUTS       AND A SPACE
          JSR    GETBYT     CHANGE ?
          BCS    BLEEP      ERROR, NO CHANGE
          STA    ,X          ELSE REPLACE WITH NEW VALUE
          BRA    MAIN       NOW RETURN
*
* REGS --- PRINT CPU REGISTERS
*
REGS     JSR    PCC         PRINT CCR
          JSR    PUTS       SEPARATE FROM NEXT STUFF
          CLR    GET+1+HI   POINT TO PAGE ZERO
          LDA    #STACK+2
          STA    GET+1+LO   POINT TO A ON STACK
          JSR    OUT2HS     NOW PRINT A
          JSR    OUT2HS     AND X ,
          JSR    OUT4HS     THE PC,
          JSR    DISADD     THE CURRENT ADDRESS
          LDA    #'=
          JSR    PUTC       PRINT '='
          JSR    PUTS       SPACE
          CLR    GET+1+HI   CLR ADDRESS CARRY
          JSR    PRDAT      FINALLY THE DATA AND A SPACE
*
*
* MAIN --- PROMPT , GET AND DECODE COMMAND

```

```

*
MAIN   JSR    CRLF      START A NEW LINE
        LDA    #MPRT
        JSR    PUTC     PRINT THE PROMPT
        JSR    GETC     GET COMMAND
        JSR    PUTS     PRINT SPACE
*
        CMP    #'A      CHANGE A
        BEQ    SETA
        CMP    #'X      CHANGE X
        BEQ    SETX
        CMP    #'R      REGISTERS
        BEQ    REGS
        CMP    #'E      EXECUTE
        BEQ    EXEC
        CMP    #'C      CONTINUE
        BEQ    CONT
        CMP    #'M      MEMORY
        BEQ    MEMORY
        CMP    #'L      LIST BLOCK OF MEM
        BEQ    LISTR
        CMP    #'V      ADDRESS CHANGE
        BEQ    MEMDIS
        CMP    #'P      PRESET EEPROM1
        BEQ    PRESTR
        CMP    #'K      SET CONSTANTS
        BEQ    CONSTR
*
        FALL INTO BLEEP
*
* BLEEP -- PRINT '?' AND PROTEST
*
BLEEP  LDX    #NGM-CMA ?+BEEP MESSAGE
        JSR    PUTMSG
        BRA    MAIN
*
*
LISTR  JMP    LIST
PRESTR JMP    PRESET
CONSTR JMP    CONSTS
*
* EXEC --- EXECUTE FROM GIVEN ADDRESS
*
EXEC   JSR    GETBYT   GET HIGH BYTE
        BCS    BLEEP   NOT A HEX DIGIT
        AND    #$1F    MAX ADDR $1FFF
        TAX
        JSR    GETBYT   NOW THE LOW BYTE
        BCS    BLEEP   BAD ADDRESS
        STA    STACK+5 PC LOW
        STX    STACK+4 PC HIGH
*
* CONT --- CONTINUE USER PROGRAM
*
CONT   RTI
*
        COULDN'T BE SIMPLER
*

```

* MEMDIS - MEMORY ADDRESS CHANGE (FOR REGISTER DISPLAY)

*

MEMDIS	JSR	DISADD	PRINT PRESENT ADDRESS
	JSR	GETBYT	GET NEW VALUE
	BCS	BLEEP	RETURN FOR NON VALID INPUT
	STA	MEMADD	
	BRA	MAIN	RETURN

*

* MEMORY - MEMORY EXAMINE / CHANGE

*

MEMORY	JSR	MEM8	GET ADDRESS
	BCS	BLEEP	NOT HEX CHAR
MEM2	JSR	CRLF	BEGIN NEW LINE
	JSR	PRADD	PRINT CURRENT ADDRESS
	JSR	PRDAT	AND ASSOCIATED DATA
	JSR	GETBYT	TRY TO GET A BYTE
	BCS	MEM3	MIGHT BE A SPECIAL CHAR
MEMA	JSR	DROP	OTHERWISE, PUT IT AND CONTINUE
MEM4	JSR	BUMP	GOTO NEXT ADDRESS
	BRA	MEM2	AND REPEAT

*

MEM3	CMP	#SAME	RE-EXAMINE SAME ?
	BEQ	MEM2	YES, RETURN WITHOUT BUMPING
	CMP	#FWD	GO TO NEXT ?
	BEQ	MEM4	YES, BUMP THEN LOOP
	CMP	#BACK	GO BACK ONE BYTE ?
	BEQ	MEM5	YES, GO DECREMENT ADDRESS
	CMP	#ASCII	NEXT VALUE ASCII ?
	BEQ	MEM9	GO READ VALUE
	CMP	#PLUS	INCREMENT DATA ?
	BEQ	MEM6	YES, GO READ DATA
	CMP	#MINUS	DECREMENT DATA ?
	BNE	BLEEP	NO, EXIT MEMORY COMMAND

*

	JSR	PICK	GET THE DATA BYTE
	DECA		
	BRA	MEM7	AND GO PUT IT BACK
MEM6	JSR	PICK	GET THE DATA BYTE
	INCA		
MEM7	JSR	DROP	PUT IT BACK
	BRA	MEM2	READY

*

MEM5	DEC	GET+1+LO	DECREMENT LOW BYTE
	LDA	GET+1+LO	CHECK FOR UNDERFLOW
	CMP	#\$FF	
	BNE	MEM2	NO UNDERFLOW
	DEC	GET+1+HI	
	LDA	GET+1+HI	SAME FOR HIGH BYTE
	CMP	#\$FF	
	BNE	MEM2	OK
	LDA	#\$1F	HIGHEST ADDRESS IS \$1FFF
	STA	GET+1+HI	
	BRA	MEM2	

*

MEM9	JSR	GETC	READ 1 BYTE
------	-----	------	-------------


```

      BRA      MEMA
*
MEM8  JSR      GETBYT   BUILD ADDRESS
      BCS      MEND     NOT HEX CHAR
      STA      GET+1+HI
      JSR      GETBYT
      BCS      MEND
      STA      GET+1+LO ADDRESS IS NOW COMPLETE
MEND  RTS      AND IN GET+1 HI & LO
*
* BULKW -- BULK WRITE EEPROM1 - DATA IS IN A - A IS UNCHANGED
*          BULK OPERATION BEING DISABLED IN USER MODE, THE
*          BULK OPERATIONS ARE PERFORMED BY SUCCESSIVE
*          BYTE OPERATIONS.
*
BULKW LDX      #01
      STX      GET+1+LO
      STX      GET+1+HI SET UP ADDRESS $101
NXTBTR BSR     BYTEW    BYTE WRITE
      JSR      BUMP
      BRCLR   1,GET+1+HI,NXTBTR LOOP
      RTS
*
* BULKE -- BULK ERASE OF EEPROM1 - A IS UNCHANGED
*          SEE ABOVE ABOUT BULK OPS
*
BULKE CLR      GET+1+LO
      LDX      #01     SET UP ADDRESS
      STX      GET+1+HI
NXTBTE BSR     BYTER    BYTE ERASE
      JSR      BUMP
      BRCLR   1,GET+1+HI,NXTBTE LOOP
      RTS
*
* PRESET - EEPROM SET OR ERASE
*
PRESET JSR     GETBYT   GET DATA BYTE
      BCS     BLEEPZ   MAY BE SPECIAL COMMAND
      BSR     BULKE    ERASE E2PROM
      BSR     BULKW    WRITE E2PROM
MAINT  JMP     MAIN     RETURN
*
BLEEPZ CMP     #'Z      DATA = ADDRESS ?
      BEQ     DADD
      CMP     #'P      CHCKBOARD 5'S ?
      BEQ     PS5
      CMP     #'Q      CHCKBOARD A'S ?
      BEQ     PSA      ELSE FALL INTO BLEEP
*
*
BLEEPK LDA     SCDAT    CLEAR RX STATUS
BLEEPR JMP     BLEEP
*
*
* DADD --- PRESET EEPROM WITH DATA = ADDRESS

```

```

*           (ADDRESS $100 UNCHANGED)
*
DADD   LDA    #$01    BOTTOM OF EEPROM
        STA    GET+1+HI
DADLP  STA    GET+1+LO ADDRESS LSB
        BSR    BYTEW   WRITE DATA IN A
        INCA   NEXT ADDRESS, NEXT DATA
        BNE    DADLP   NO OVERFLOW YET
DADND  BRA    MAINT   JOB DONE
*
*
* CHECKBOARD PATTERNS
*
*
PS5    LDA    #$55    START VALUE
        BRA    PSC
PSA    LDA    #$AA
*
PSC    LDX    #01
        STX    GET+1+LO
        STX    GET+1+HI
*
        STA    ATEMP
AX25   LDA    #$0F
        AND    GET+1+LO
        BNE    TNC
        COM    ATEMP
*
TNC    LDA    ATEMP
        JSR    BYTEW
        JSR    BUMP
        BRCLR 1,GET+1+HI,AX25
        BRA    DADND
*
*
* BYTEW -- BYTE WRITE TO EEPROM1 - DATA IS IN A
*           ADDRESS IN GET+1&2 - A IS UNCHANGED
*
BYTEW  STA    FLAG    SAVE A
        BSET   .E1LAT,EECONT PROG LATCH ENABLE
        JSR    DROP1  PUT ADDRESS + DATA
        BSET   .E1PGM,EECONT
        LDA    WRITEK  GET WRITE TIMING CONSTANT
        JSR    ADJDEL
        BCLR  .E1LAT,EECONT END OF
        LDA    FLAG    RESTORE A
        RTS
*
*
* BYTER -- BYTE ERASE - A IS UNCHANGED AND ADDRESS
*           IS IN GET+1 HI & LO
*
BYTER  STA    FLAG    SAVE DATA
        BSET   .E1LAT,EECONT PROG LATCH ENABLE
        BSET   .E1ERA,EECONT
        JSR    DROP1  PUT ADDRESS

```

```

BSET .E1PGM,EECONT
LDA ERASEK GET ERASE TIMING CONSTANT
JSR ADJDEL
BCLR .E1LAT,EECONT
LDA FLAG RESTORE A
RTS

```

*

* LIST --- LIST 4 BLOCKS OF 64 BYTES ON ONE PAGE

*

```

LIST JSR MEM8 GET START ADDRESS
      BCC GOL GOOD ADDRESS
      LDA #01
      STA GET+1+HI SET EEPROM1 ADDRESS
      CLR GET+1+LO

*
GOL JSR CRLF
      CLRX COUNTER 256 BYTES
LL1 JSR PRADD PRINT ADDRESS
      JSR PUTS AND A SPACE
LNEXT JSR PRDAT PRINT DATA
      JSR BUMP NEXT BYTE
      STX XTEMP A GOOD THOUGHT FOR
      JSR ADJ10 UNBUFFERED TERMINALS
      LDX XTEMP
      DEX DATA COUNT DECREMENT
      TXA
      AND #%00001111 TEST FOR 16TH DATA BYTE
      BEQ LINE
      AND #%00000111 TEST FOR 8TH DATA BYTE
      BNE LNEXT
      JSR PUTS INSERT EXTRA 2 SPACES
      JSR PUTS
      BRA LNEXT
MEC9BH JMP MAIN

*
LINE BSR ASCIIR PUT ASCII LINE
      CPX #00
      BEQ MEC9BH
      JSR CRLF
      CMPX #128 TEST FOR 128 BYTE PAGE
      BNE NOPAGE
      JSR CRLF ADD EXTRA LINE IF TRUE
NOPAGE BRA LL1

*
ASCIIR LDA GET+1+LO DECREMENT MEMORY POINTER
      SUB #16
      STA GET+1+LO
      BCC NOBORO
      DEC GET+1+HI
      LDA GET+1+HI
      AND #$1F JUST IN CASE
      STA GET+1+HI
NOBORO TXA BACK UP BYTE COUNTER
      ADD #16
      TAX

```

```

        JSR     PUTS     SEPARATE WITH 2 SPACES
        JSR     PUTS
LPASC   BSR     PUTASC   PUT CHAR
        JSR     BUMP
        DEX
        TXA
        AND     #$0F
        BEQ     LINER
        BRA     LPASC   FINISH THIS LINE
LINER   RTS
*
PUTASC  JSR     PICK
        AND     #$7F
        CMP     #$20
        BHS     DISP   '.' BELOW $20
        LDA     #'
DISP    CMP     #$7F
        BNE     DISP1
        LDA     #$20   SPACE FOR DEL ($7F)
DISP1   JSR     PUTC
        RTS
*
*
* DISPLAY AND CHANGE WRITE AND ERASE CONSTANTS
*
CONSTS  CLR     FLAG     0=ERA; 1=WRI
        LDX     #ERA-CMA
        BRA     CM1
CML     LDX     #WRI-CMA
CM1     JSR     CRLF
        JSR     PUTMSG   PUT EITHER WRI OR ERA
        LDA     ERASEK
        BRCLR  7,FLAG,CM2
        LDA     WRITEK
CM2     BSR     BINBCD
        JSR     PUTBYT   PRINT BCD VALUE
        JSR     PUTS
        LDX     #MS-CMA
        JSR     PUTMSG   PRINT ms
*
        JSR     GETNYB   GET MS NYBBLE
        BCS     NWER     IF ERROR
        CMP     #$9
        BHI     NWER     BCD INPUT !
        STA     GET      SAVE MSD
*
        JSR     GETNYB   GET LS NYBBLE
        BCS     NWER     IF ERROR
        CMP     #$9
        BHI     NWER     BCD INPUT !
        BSR     BCDBIN   SAVE IN BINARY
        TSTA    CMP     #$00
        BEQ     NWER     MIN IS 1 ms
        BRSET  7,FLAG,CM3 WRI K
        STA     ERASEK

```

```

CMX      DEC      FLAG
        BRA      CML
CM3      STA      WRITEK
CM4      JMP      MAIN
*
NWER     CMP      #CR      CR ?
        BNE      CMD
        BRCLR   7, FLAG, CMX
        BRA      CM4
CMD      JMP      BLEEP
*
*
*
* U T I L I T I E S
*
*
*
BCDBIN  CLRX      POINT TO CONV TABLE
BCBIL   LSR      GET
        BCC      BCBIL1
        ADD      BBTBL, X
BCBIL1  INX
        CPX      #$4
        BNE      BCBIL
        RTS      A HAS BCD NB
*
*
BINBCD  CLR      COUNT      FUTURE BCD
        STA      CHAR      SAVE ENTRY PARAMETER
BBCLOP  LDA      CHAR
        BEQ      DONEB
        DEC      CHAR
        INC      COUNT
*
        LDA      COUNT      OVF 9 => A
        AND      #$0F
        CMP      #$0A
        BNE      BBC1      IF NOT
        LDA      COUNT
        ADD      #$06      ADJUST
        STA      COUNT
*
BBC1    BRA      BBCLOP
DONEB   LDA      COUNT      GET BCD VALUE
        RTS
*
*
* PICK --- GET BYTE FROM ANYWHERE IN MEMORY
*        THIS IS A HORRIBLE ROUTINE (NOT MERELY
*        SELF MODIFYING, BUT ALSO SELF CREATING)
*
*        GET+1 HI & LO POINT TO ADDRESS TO BE READ.
*        BYTE IS RETURNED IN A
*        X IS UNCHANGED AT EXIT
*

```

```

PICK  STX  XTEMP  SAVE X
      LDX  #$D6  $D6 = LDA 2-BYTE INDEXED
      BRA  COMMON
*
* DROP  --- PUT BYTE TO ANY MEMORY LOCATION
*          HAS THE SAME UNDESIRABLE PROPERTIES
*          AS PICK
*
*          A HAS BYTE TO STORE AND GET+1 HI & LO POINTS
*          TO LOCATION TO STORE, A AND X ARE
*          UNCHANGED AT EXIT
*
*          THE FLOW IS DIFFERENT WHETHER FOR RAM OR EEPROM
*
DROP  BSR  TSTEE  WITHIN EEPROM ?
      BCC  DROP1  NOT EEPROM
      JSR  BYTER  ERASE BYTE
      JSR  BYTEW  WRITE BYTE
      RTS
DROP1 STX  XTEMP  SAVE X
      LDX  #$D7  $D7 = STA 2-BYTE INDEXED
*
*
COMMON STX  GET  PUT OPCODE IN PLACE
      LDX  #$81  $81 = RTS
      STX  GET+3  NOW THE RETURN
      CLRX  WE WANT ZERO OFFSET
      JSR  GET  EXECUTE THIS MESS
      LDX  XTEMP  RESTORE X
      RTS  AND EXIT
*
* TSTEE -- TEST THE ADDRESS IN GET+1 HI & LO AND RETURN
*          WITH CARRY SET IF IT IS A VALID EEPROM1
*          ADDRESS, AND CLEARED OTHERWISE.
*
TSTEE LDX  GET+1+HI
      CPX  #01  TEST HI BYTE
      BNE  NOEE  NOT EEPROM1
      SEC
      RTS
NOEE  CLC
      RTS
*
*
* BUMP  --- ADD ONE TO CURRENT MEMORY POINTER
*          A AND X UNCHANGED
*
BUMP  INC  GET+1+LO INCREMENT LOW BYTE
      BNE  BUMP2  NON-ZERO MEANS NO CARRY
      INC  GET+1+HI INCREMENT HIGH NYBBLE
BUMP2 RTS
*
* OUT4HS - PRINT BYTE POINTED TO AS AN ADDRESS AND
*          BUMP POINTER - X IS UNCHANGED AT EXIT
*

```

```

OUT4HS BSR    PICK    GET HIGH BYTE
        BSR    PUTBYT  AND PRINT IT
        BSR    BUMP    GO TO NEXT ADDRESS
*      FALL INTO OUT2HS
*
* OUT2HS - PRINT BYTE POINTED TO, THEN A SPACE,
*      BUMP POINTER. X IS UNCHANGED AT EXIT
*
OUT2HS BSR    PICK    GET THE BYTE
        BSR    PUTBYT
        BSR    BUMP    GO TO NEXT
        BSR    PUTS    FINISH UP WITH A SPACE
        RTS

*
* DISADD - PRINT ONE BYTE ADDRESS IN MEMADD
*
DISADD LDA    MEMADD  GET ADDRESS
        STA    GET+1+LO SET UP TO PRINT
        BSR    PRADD1 PRINT ADDRESS (PAGE 0)
        RTS

*
* PRADD -- PRINT CURRENT ADDRESS FROM GET+1 HI & LO
*
PRADD  LDA    GET+1+HI PRINT CURRENT LOCATION
        AND   #$1F    max $1FFF
        STA    GET+1+HI CONVENIENTLY RESTORE 1X
        BSR    PUTBYT
PRADD1 LDA    GET+1+LO
        BSR    PUTBYT
        BSR    PUTS    THEN A SPACE
        RTS

*
* PRDAT -- PRINT DATA POINTED TO BY GET+1 HI & LO
*
PRDAT  BSR    PICK    GET THAT BYTE
        BSR    PUTBYT PRINT IT
        BSR    PUTS    ANOTHER SPACE
        RTS

*
* PUTBYT - PRINT EA| IN HEX - A AND X UNCHANGED
*
PUTBYT STA    GET    SAVE A
        LSRA
        LSRA
        LSRA
        LSRA          SHIFT HIGH NYBBLE DOWN
        BSR    PUTNYB PRINT IT
PTS    LDA    GET
        BSR    PUTNYB PRINT LOW NYBBLE
        RTS

*
* PUTNYB - PRINT LOWER NYBBLE OF A IN HEX
*      A AND X UNCHANGED
*      HIGH NYBBLE OF A IGNORED
*

```

```

PUTNYB STA   GET+3   SAVE A IN YET ANOTHER TEMP
      AND   #$0F   MASK OFF HIGH NYBBLE
      ADD   #'0     ADD ASCII ZERO
      CMP   #'9     CHECK FOR A-F
      BLS  PUTNY2
      ADD   #'A-'9-1 ADJUSTMENT FOR HEX A-F
PUTNY2 JSR   PUTC
      LDA   GET+3   RESTORE A
      RTS

```

```

*
* CRLF --- PRINT CARRIAGE RETURN - LINE FEED
*       A AND X UNCHANGED
*

```

```

CRLF  STA   GET       SAVE A
      LDA   #CR
      JSR  PUTC
      LDA   #LF
      JSR  PUTC
      LDA   GET       RESTORE A
      RTS

```

```

*
* PUTS --- PRINT A SPACE - A AND X UNCHANGED
*

```

```

PUTS  STA   GET       SAVE A
      LDA   #SPACE
      JSR  PUTC
      LDA   GET       RESTORE A
      RTS

```

```

*
* GETBYT - GET A HEX BYTE FROM TERMINAL
*
*       A GETS THE BYTE TYPED IF IT WAS A VALID HEX
*       NUMBER, OTHERWISE A GETS THE LAST CHAR TYPED.
*       THE C-BIT IS SET ON NON HEX CHARS, CLEARED
*       OTHERWISE. X IS UNCHANGED IN ANY CASE.
*

```

```

GETBYT BSR   GETNYB   BUILD BYTE FROM 2 NYBBLES
      BCS  NOBYT     NON HEX CHAR
      ASLA
      ASLA
      ASLA
      ASLA           SHIFT NYBBLE TO HIGH NYBBLE
      STA   GET       SAVE IT
      BSR  GETNYB   GET LOW NYBBLE NOW
      BCS  NOBYT     NON HEX CHAR
      ADD  GET       C-BIT CLEARED
NOBYT  RTS

```

```

*
* GETNYB - GET HEX NYBBLE FROM TERMINAL
*
*       A GETS THE NYBBLE TYPED IF IT WAS IN THE RANGE 0-F,
*       OTHERWISE A GETS THE CHARACTER TYPED. THE C-BIT IS
*       SET ON NON HEX CHARACTERS, CLEARED OTHERWISE.
*       X IS UNCHANGED
*

```



```

GETNYB BSR      GETC      GET THE CHARACTER
        STA      GET+3    SAVE IT JUST IN CASE
        SUB      #'0      SUBTRACT ASCII ZERO
        BMI      NOTHEX   WAS LESS THAN '0'
        CMP      #9
        BLS      GOTIT
        SUB      #'A-'9-1 FUNNY ADJUSTMENT
        CMP      #9F      TOO BIG ?
        BHI      NOTHEX   WAS GREATER THAN 'F'
        CMP      #9      CHECK BETWEEN ASCII 9 AND A
        BLS      NOTHEX
GOTIT  CLC      C=0 MEANS GOOD HEX CHAR
        RTS
NOTHEX LDA      GET+3    GET SAVED CHAR
        SEC
        RTS      RETURN WITH 'ERROR'

```

```

*
* ADJDEL - DELAY FOR EEPROM ROUTINES = TO £A| ms
*

```

```

ADJ10  LDA      #10
ADJDEL LDX      #83      CONSTANT
AL1    BRCLR   4,ADSTCT,**+3 DUMMY
        BRCLR   4,ADSTCT,**+3 DUMMY
        BRCLR   4,ADSTCT,**+3 DUMMY
        BRN     *          DITTO
        DECX
        BNE     AL1
        DECA
        BNE     ADJDEL    LOOP A TIMES
        RTS

```

```

*
*
* PUTMSG - PRINT THE MESSAGE POINTED TO BY X
*

```

```

PUTMSG LDA      CMA,X    GET NEXT CHARACTER
        CMP      #EOT
        BEQ      NDMMSG
        BSR      PUTC    SEND CHAR
        INX
        BRA      PUTMSG
NDMMSG RTS

```

```

*
*
*
*
*
*

```

S E R I A L I / O R O U T I N E S

```

* Initialise the SCI
*

```

```

SCINIT  BCLR    MBIT,SCCR1      8 data bits
        LDA     #%11000000      baud rate 9600
        STA     BAUD
        LDA     #%00001100      TE / RE
        STA     SCCR2          end of init
        STA     SCSR          clear TDRE & TC bits

```

```

RTS
*
*
* GETC : Routine GETC services the SCI, it does that by polling
* the RDRF (received data ready flag). It returns with
* the byte of data in ACCA.
*
*
GETC   BRCLR  .RDRF,SCSR,*           Possibly wait for char
GDATA  LDA    SCDAT                  get data & clear RDRF
        CMP   #'/'                   NEXT CHAR ASCII ?
        BNE   CHARS
        DEC   ASC                     FLAG IT
        RTS
CHARS  BRSET  7,ASC,SCHAR
        CMP   #$40
        BLS   NOCHAR
        AND   #$1011111             UPPER CASE
        CLR   ASC
        RTS
SCHAR  NOCHAR
*
*
* PUTC : Routine PUTC services the SCI. It polls the TDRE
* (Transmit Data Register Empty), and puts the char
* when true.
*
*
PUTC   BRCLR  TDRE,SCSR,*           WAIT
        STA   SCDAT
        RTS
*
*
* ===== ENTRY =====
*
* MONIT - ENTRY POINT FROM RESET
*
MONIT  LDA    #10                    10 ms BY DEFAULT
        STA   ERASEK                 SET ERASE DEFAULT TIME
        STA   WRITEK                 SET WRITE DEFAULT TIME
        JSR   SCINIT                 INIT SCI
        LDA   #ADDATA
        STA   MEMADD                 DISPLAY ADR BY DEFAULT
        JSR   CRLF                    START A BRAND NEW LINE
        CLRX  CLRX                    POINT TO START OF MESSAGE
BABBLE LDA   MSG,X                   GET NEXT CHARACTER
        BRCLR 4,PORTD,BAB1           ROM MESSAGE
        LDA   EEPROM+1,X             GET NEXT CHAR (EEPROM1 MESSAGE)
BAB1   CMP   #EOT
        BEQ   BABND                   IF END OF MESSAGE
        BSR   PUTC                     PRINT IT
        INCX CLRX                     POINT TO NEXT CHAR
        BNE   BABBLE                   MORE !
BABND  JSR   CRLF                     SEPARATE MESSAGE FROM COMMANDS
        SWI   GO TO MONITOR ROUTINES
        BRA   MONIT                   LOOP AROUND
*

```

```

*
* EEPROM1 BURN IN TEST ROUTINE.
* SET UP REQUIRED NB OF ITERATION IN $70:$71
* AND DATA TO BE PROGRAMMED IN $72.
* NOTE : MAXIMUM NB OF ITERATION IS $7FFF.
*
*

```

```

ABCNT EQU $70
ABDAT EQU $72
*

```

```

ABCD JSR CRLF
ABL LDA ABDAT
      JSR BULKE
      JSR BULKW
      LDA ABCNT+LO
      DECA
      STA ABCNT+LO
      CMP #$FF
      BNE NOBURO
      DEC ABCNT+HI
      BMI NDAB
NOBURO LDA ABCNT+HI
        JSR PUTBYT
        LDA ABCNT+LO
        JSR PUTBYT
        JSR CRLF
        BRA ABL
NDAB SWI
*

```

```

=====
*
* VECTORS
*

```

```

* The unused vectors point to RAM, so as to be available
* for test purposes (RAM Bootloader, SCI loader). Their
* positionning allows 10 bytes for the stack, that is 2
* interrupt levels, or 1 interrupt and 2 subroutine levels.
*

```

```

FDB STACK-9-18 SCI
FDB STACK-9-15 TIM OVF
FDB STACK-9-12 TIM OUT COMP
FDB STACK-9-9 TIM IN CAP
FDB STACK-9-6 IRQ
FDB MAIN SWI
FDB MONIT RESET
*

```

```

=====
*
* E N D
*

```

```

*****
END

```

An introduction to SECURE SINGLE CHIP MICROCOMPUTER MANUFACTURE

By Mike Paterson
Motorola Ltd.
East Kilbride

INTRODUCTION

Motorola is one of the world leaders in the design and manufacture of advanced semiconductor devices. We have a major manufacturing capability at East Kilbride, in Scotland's "Silicon Glen", where we manufacture many of our latest microprocessor products—primarily for the European market, though we ship product world-wide, including to Japan and the U.S.A. Our business is in selling silicon; within this overall goal however we have, for more than a decade, worked on developing the necessary technology, design and manufacturing techniques required to produce a range of secure microcomputer products specifically for the SmartCard marketplace.

One of the most important and fundamental issues for SmartCards is security. There are many financial, commercial, industrial and even military applications for SmartCards which are viable only if they can provide the appropriate levels of security demanded by such applications. This brings us to the title of this Engineering Bulletin, does it mean "*manufacture of secure microcomputers*" or "*secure manufacture of microcomputers*"? Motorola is perhaps unique among today's silicon suppliers in that it can provide microcomputers both designed from the outset to be secure and from a secure manufacturing line. We recognise that the overall security of any application is dependent not only on the intrinsic security of the device itself and its ability to prevent unauthorised access, but also on the measures taken by the semiconductor supplier to prevent fraudulent tampering with the device during and after manufacture. The customer's application software is also a critical link in

the security chain. If the semiconductor manufacturer wants to play an active part in the SmartCard marketplace, he must recognise security as a key parameter in everything from the conceptual design of a new microcomputer, through his manufacturing process, even to his delivery of the product to his customer.

Security in relation to microcomputers used for SmartCards can be grouped into three main categories:

1. Designed-in ("intrinsic") security
2. Manufacturing security
3. Application security

Clearly Design and Manufacturing Security are the responsibility of the semiconductor manufacturer, and are the main subjects of this Engineering Bulletin. Application Security on the other hand is ultimately the responsibility of the SmartCard systems designer. His task is to design the (secure) application software required to meet the system specification of, for example, a financial transaction or an industrial security application. Application security takes into consideration the design of the chip and the on-chip security features which can be utilised by the user's application software. But even here the semiconductor manufacturer has a role to play; the manufacturer needs to be able to provide Applications Engineers well versed in the hardware and software features of his microcomputers and of their development support tools, to act as consultants in helping customers develop their own hardware and software.

This Engineering Bulletin will start by defining what we mean by a single chip microcomputer and what is fundamentally different about a secure one; then it will look briefly at the history of Motorola's involvement in this business, and how it has developed its SmartCard product line capability over the last decade to where it is today. Some of the techniques available to the design engineer which allow him to include security features in the device itself will be explored, along with the constraints placed on him by the demands of the SmartCard market and the need to test these devices. It will look very briefly at the wafer fabrication process and what security measures can and must be taken. Finally, it will consider the device testing process, and the particular problems associated with the testing of secure microcomputers.

SINGLE CHIP MICROCOMPUTERS

What do we mean by a single chip microcomputer, and what is special about a secure one? A single chip microcomputer (commonly referred to as an MCU) is a full microprocessor system integrated on to a single piece of silicon. It is a complete computer system in miniature, containing almost all the resources required to implement a particular application, or range of applications. In addition to the Central Processing Unit (CPU) and its control circuitry, it typically contains blocks of different types of memory, and a selection of hardware functions, optimised for general or sometimes very specific application areas. The only computer-like resources it lacks are the external human or machine interface devices such as keyboards, displays, disk drives, transducers and sensors. Most MCUs contain, at the very least, areas of random access memory (RAM) and read only memory (ROM). The RAM is used by the CPU for temporary data storage during calculations and transactions. Data can be read from or written to the RAM by the CPU in the

space of a few microseconds, however any data in the RAM is lost when the electrical power supply is removed. The ROM area is used to store information which will not change throughout the working life of the MCU. Primarily it is used to store the customer's application software (known as the user software or "ROM code"). This controls the sequence of logical operation and decision making of the CPU. It can also contain any fixed data, in the form of look-up tables for example, which may be required by the application. All this information is built into the ROM during the silicon manufacturing process and can never be altered after the manufacturing process is complete. MCUs incorporating arrays of non-volatile memory, such as EPROM (erased by UV light) and EEPROM (electrically erasable) are available for applications where variable data has to be kept for long periods of time, even when power is removed from the MCU. This is particularly important in SmartCard applications where transaction data and other records must be updated every time the card is used (possibly several times a day depending on the type of application) and which must be retained for weeks, months, or even years in some cases. Typical on-chip hardware functions for general applications are simple counter/timers, byte-wide I/O ports and serial communications interfaces.

An important thing to realise is that most typical single chip microcomputers are designed to function in a number of different operating modes which can be selected by the user. In the single chip **user-mode**, which is the normal mode of operation in most applications, the CPU runs under the control of the user software built into the on-board ROM. Some MCUs support **expanded-mode** operation where internal data and address buses are connected to the I/O pins to allow the CPU to access additional memory and I/O outside the MCU. Other operating modes are provided for **testability**.

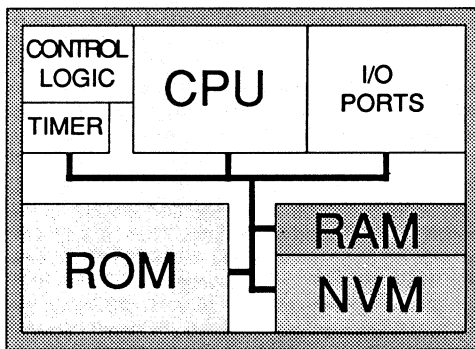


Fig. 1: General Purpose Microcomputer

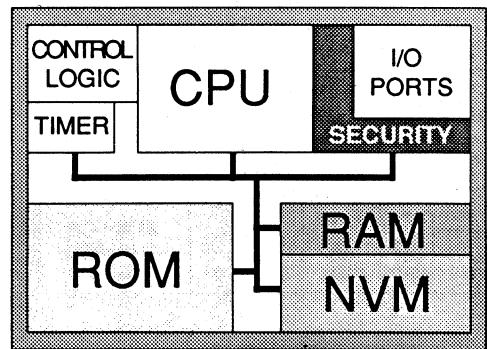


Fig. 2: Secure Single-Chip Microcomputer

A *secure* single chip microcomputer can include any or all of the features just described, but it also has the built-in capability to prevent, by various means, unauthorised access to the CPU, the memory arrays, the user/application software, and any data being processed or stored within the device, at any time. After it has been tested and passed as fully functional by the semiconductor manufacturer, the only possible mode of operation for a secure microcomputer must be the **user-mode**, i.e. under the complete control of the user software in the on-board ROM.

HISTORY

In 1977 Motorola (working with one of its European customers) began a feasibility study concerned with putting a microcomputer and a non-volatile memory within an ISO7810 credit card. This solution was fully functional by 1979. As a result of evaluation and feasibility studies it soon became apparent that "multi-chip computer" solutions have a number of inherent disadvantages which can detract significantly from their suitability for secure, reliable, high-volume SmartCard applications. In multi-chip solutions, the interconnections forming the control and data buses between the chips are easily accessible from the outside world and the data being transmitted across them may be intercepted and monitored. This can seriously limit the inherent security of such a multi-chip solution. In single chip solutions however the interconnections are buried deep within the structure of the silicon die and, in devices intended for SmartCard applications, the silicon designer can use a variety of techniques to ensure that these interconnections cannot be accessed from the outside world. Multi-chip solutions are also inherently less reliable and usually more expensive than single chip solutions because physical stresses can damage the external interconnections and fabrication, test and assembly

costs have to be incurred for each chip. We believe therefore that the concept of a single chip microcomputer solution is important to the successful realisation of secure, reliable, high-volume SmartCard applications.

Work began in Motorola's European Design Office in Geneva in 1980 on the design of a SmartCard chip to a general specification agreed with a potential customer. The design was based upon the CPU of the highly successful M6805 single chip MCU, with 8 K bits of UV erasable memory (EPROM), as well as RAM and ROM. This was all integrated on a single NMOS technology silicon chip of less than 20 mm². The merging of EPROM and NMOS MCU technologies on one piece of silicon was a new area for Motorola and our Advanced Products Research and Development Laboratory (APRDL) was given the task of designing and developing a completely new wafer fabrication process to support this combined technology. First silicon was successfully produced in APRDL in 1981, and in 1982 the new production process was transferred from APRDL to the volume wafer fabrication facility in East Kilbride, Scotland where the world's first single chip SmartCard MCU then went into full production.

Since then the advent of HCMOS technology and the ability to integrate electrically erasable memory arrays on the same chip have resulted in the extension of the family of SmartCard microcomputer chips based on the M6805 CPU. This family offers various combinations and sizes of RAM, ROM, EPROM and EEPROM areas. Other families of high-performance, high-functionality MCUs which have been developed in parallel with the SmartCard family, notably the 68HC05B-, 68HC05C- and 68HC11- families, are ideal for controlling SmartCard readers, keyboards, communications channels and interface devices.

Now, in 1990, East Kilbride is Motorola's world-wide centre of excellence for SmartCard product and manufacturing technology and, as a result of its ongoing commitment to this emerging market, has so far shipped more than 20 000 000 SmartCard devices.

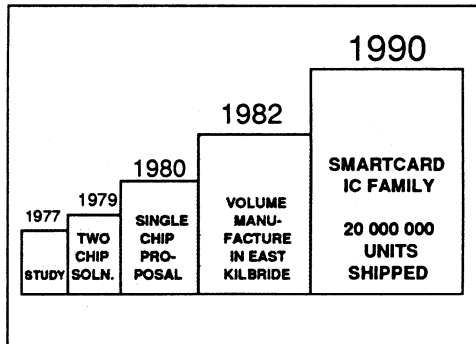


Fig. 3: Development History

DESIGN SECURITY

The silicon designer's task is never an easy one, but it is particularly difficult when the end product is intended for SmartCard applications. There is a continual trade off between what the market demands and what it is willing to pay for in terms of functionality. One of the main reasons why general purpose single chip MCUs are not a cost effective solution for SmartCard applications, forgetting for the moment the lack of in-built security, is that a considerable area of silicon is wasted supporting functions which are not really

necessary for this type of application. All they serve to do is to increase the die size which increases the cost of production, and hence the price of the end product. The relative quantities of different types of on-chip memory is another key issue. The SmartCard market has a voracious appetite for non-volatile memory. However an EPROM cell uses roughly twice the silicon area of a ROM cell, and an EEPROM cell is about twice the size of an EPROM cell. In other words, if the target die size can accommodate 8 K bytes of ROM, the customer can have approximately 4 K bytes of EPROM or 2-3 K bytes of EEPROM instead. This is very much an oversimplification as for example each type of memory has a different requirement for decoding and programming circuitry, which also add to the silicon area. Also, the combination of non-volatile memory technologies and NMOS and HCMOS semiconductor technologies makes the manufacturing and testing processes much more complex and expensive. Although technological advances have helped reduce the dimensions of non-volatile memory cells, EPROM will always be more expensive than ROM, and EEPROM will be more expensive still. RAM is the most expensive type of memory in terms of silicon area. Fortunately, very large arrays of RAM are not usually required for SmartCard applications, though this does put pressure on cryptologist-programmers to use the available RAM very efficiently.

The result of the silicon designers' efforts may well be a masterpiece of design ingenuity with a functional specification second to none, but it also has to be manufacturable, in very high volumes, at the right price for the end customer – hence it has to be testable, quickly and thoroughly. Testing of a microprocessor device is usually achieved by accessing the data, address and control buses via the pins which electrically connect the internal circuitry of the device to the outside world. This is fairly easy to achieve with standard microprocessor devices which are intended

for multi-chip applications, as their data, address and control buses appear on the pins of the device, and are designed to allow easy interface to external memory and interface chips. It is considerably more difficult to achieve in most single chip microcomputers. However the design engineer usually incorporates a test mode which makes the internal bus connections externally accessible, by switching them onto the various I/O pins leading to the outside world.

Single chip MCUs intended for SmartCard applications present the silicon designer with a number of special problems. The device circuitry must be designed in such a way that, after testing, the test mode can be permanently disabled to prevent this mode from being reactivated. The device must then operate only under the control of the user application software.

Single chip MCUs for SmartCards do not need a large number of connections to interface to the outside world. (The current ISO standard, which specifies a serial half duplex system, defines six connections, plus two reserved for future use.) Although this presents the designer with the opportunity to save on valuable silicon area, it introduces the additional problem of how to provide access to the internal circuitry, when there are insufficient I/O lines available to connect the internal buses to the outside world. This problem can be overcome by adding special test connections to the device, which are not connected when the device is packaged in a SmartCard, or by utilising serial communication techniques during testing. However, extra test pads add to the total silicon area of the die, and serial communication adds to the testing time, both of which ultimately add to the cost. Motorola's designers have developed a number of techniques and features which ease the testing burden and reduce the test time of the device. These features are disabled by blowing fusible links upon completion of the test.

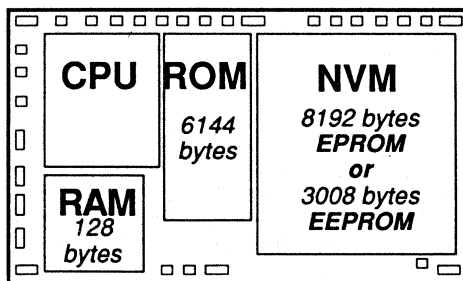


Fig. 4: Outline of MCC68HC05SC11/SC21 die showing the effect of cell complexity on the respective memory areas

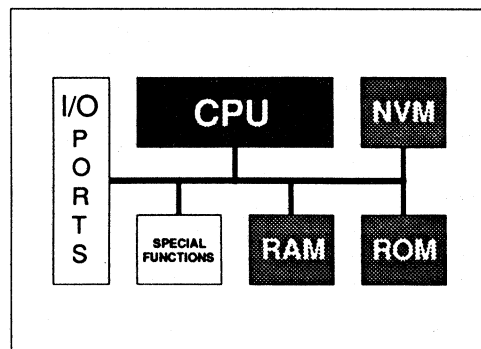


Fig. 5: General Purpose Microcomputer

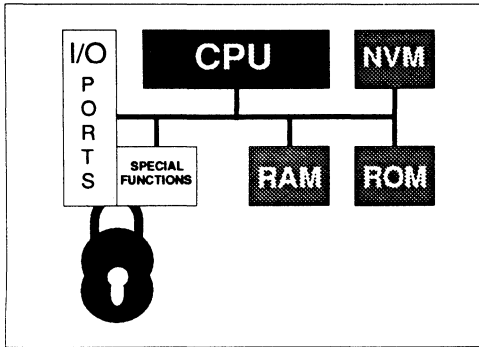


Fig. 6: Secure Microcomputer

Fusible links can be incorporated in the circuit design to connect the I/O lines to the internal data bus and other parts of the internal circuitry. These fusible links form part of the normal interconnecting layers embedded in the device during the wafer fabrication process. After testing is complete, these fuses are blown. In non-volatile memory arrays (EPROM and EEPROM) special bits or bytes can be provided within the array which allow the CPU to detect attempts to illegally erase the memory. These bits can be programmed only during testing, before the fuses are blown. Any attempt to erase the memory array will irrevocably alter the state of these bits. In some designs, it is left to the application software to decide on an appropriate course of action if such a condition is detected. In other designs, the CPU stops processing - permanently.

The layout of the circuitry of a semiconductor device on the silicon die is usually done in a fairly logical manner. However, the SmartCard chip designer can use a number of techniques to confuse anyone who may try to analyse the design. Communication buses and control lines can be routed through different masking layers rather than being routed by the most direct path. Memory topology can be made very complex with logically adjacent bits and bytes being physically distributed over the memory space. Also, dummy structures resembling transistors can be distributed within the integrated circuitry on the die.

WAFER FABRICATION

Over the last 10 years or so Motorola has progressed from 5 μm NMOS technology on 75 mm wafers to 1.5 μm low-power HCMOS technology on 150 mm wafers. The significant reduction in line widths has allowed larger memory arrays, new memory technologies, and more functional circuitry to be incorporated

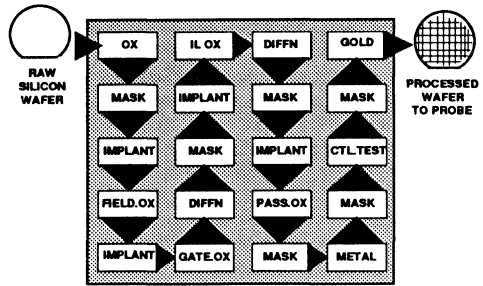


Fig. 7: Wafer Fabrication

in today's state of the art devices without increasing the die area. Unfortunately, this wonderful new technology does not come cheap. The capital investment in state-of-the-art wafer processing and test equipment is extremely high. A modern wafer fabrication facility such as the one Motorola has built in Scotland costs about £200 million to build and equip. This in itself may be viewed as an additional security feature against fraudulent manufacture!

The production process which turns raw silicon into finished wafers involves a very large number of process steps. In simple terms it is a multi-layer technology using techniques similar to photographic printing. The electronic circuit is converted into a number of layers which contain different elements of the circuit components: the interconnections; the bus lines; the bonding pads; the memory cells; and so on. Each layer is then converted into a photographic mask. (One of these masks contains the application software and customised data supplied by the end customer.) Each mask, and there can be as many as 15 in some complex device technologies, is then used in a series of exposure, print and development cycles as the electrical circuit is built up on the silicon wafer. In addition, a number of diffusion, oxidation and implant processes are used at different stages of the process to create the correct electrical and functional characteristics of the semiconductor device. The wafer fabrication process for producing secure microcomputers is essentially no different from any other type of micro-processor or microcomputer. However the inclusion of EPROM or EEPROM arrays makes the process much more complicated due to the extremely critical nature of such parameters as gate and interlevel oxide thicknesses and silicon defectivity.

MANUFACTURING SECURITY

I have just reviewed the standard wafer fabrication process. What does the semiconductor manufacturer have to do differently when the product can end up in a SmartCard application, possibly representing a considerable sum of money or controlling access to bank accounts, personal information or to high security areas?

Security must be the watchword at every stage of the process. In Motorola, for any single chip MCU, the customer's application software (the ROM code) is converted into a set of geometric coordinates and dimensions, which are stored on a pattern generation tape. This tape is then used to produce one of the masks used in the wafer fabrication process.

For SmartCard MCUs the customer's software is processed, and the pattern generation tape produced, via a special restricted-access account on Motorola's internal computer system. Transfer of the tape to mask shop and of the ROM code mask and tape back to the factory is done by special courier. The mask is then safely stored within the wafer fabrication area which is itself a highly restricted area.

During the wafer fabrication process the wafers are strictly controlled on a batch basis up to the start of the metallisation process. From then on, each wafer is tracked and traceable individually, as the devices are essentially functionally complete after metallisation. As well as having various access controls to the manufacturing areas every wafer entering the metallisation stage is fully accounted for (even to the number of good dice on each wafer, once that is determined) from then on until it is either shipped to the customer or destroyed in a secure manner.

TESTING OF NON-SMARTCARD DEVICES

The finished wafers move from the wafer fabrication line to the probe area for testing. Testing is done using a "stored response" technique. A comprehensive sequence of test signals (or vectors) is applied by the testing computer to the probe and bonding pads of each die and the resulting response signals are compared with a stored sequence of the expected responses. Any discrepancy between the actual and the expected response sequences indicates a defective device. (The device must pass every single vector in the test pattern, possibly tens of thousands of lines, to pass the test.) Parametric type tests, such as supply and leakage current measurements, are also performed.

Every die is tested for total functionality and performance to the device specification, including verification of the customer's application software in ROM. During the probe test any defective die can be physically marked or located on a wafer map to eliminate any further work on such non-functional devices. The wafers are then fixed to an adhesive plastic film before being sawn to separate the individual dice. If the devices are being supplied to the customer in packaged form, the wafers are then sent for assembly where the dice are assembled into the appropriate packages. They are then final tested and shipped to the customer via standard commercial carriers.

TESTING OF SMARTCARD DEVICES

In the factory in East Kilbride the SmartCard probe test area and all its test equipment is completely dedicated to SmartCard products and access is restricted to authorised personnel only. Each device is functionally and electrically tested as with non SmartCard product. The non-volatile memory is then erased and retested to ensure complete erasure of every cell. At this point traceability information (such as batch number, manufacturing location, test date etc.) can be written into each device together with any customer specific data, in accordance with the customer's instructions or algorithms; some of this data may be unique to each individual die. This part of the procedure is carried out via a dedicated computer system which has no communication links with the world outside the SmartCard probe area. Finally the security fuses on each die are blown.

After being sawn, the dice are ready to be shipped to the customer as complete wafers on their adhesive plastic film backing. The wafers are transported from the factory in East Kilbride to the airport bonded warehouse – and from the destination airport to the customer's premises – by armoured car. These wafers include not only good dice which have passed the test programme, but also any dice which have been identified and marked as defective. This allows the customer to account for each individual die, good or bad, on every wafer.

IN CONCLUSION

I hope that this Engineering Bulletin has given you some insight into the design and manufacture of single chip microcomputers in general. I have tried to illustrate, through the experience gained by Motorola over the past twelve years, how the additional problems set by the need for total security of design and manufacture of single chip microcomputers for the

SmartCard market have been tackled and solved. The level of expertise and overall capability required by the semiconductor manufacturer to enable him to supply the right products to the SmartCard market cannot be achieved overnight. It is not sufficient to be a world leader in microprocessor technology, with a world-beating product portfolio. The semiconductor supplier has to provide a product portfolio which includes devices designed with intrinsic security features specifically for embedding in SmartCards (or indeed in any other "security package"), and other devices with the on-chip features required for SmartCard reader and peripheral applications. Almost any general purpose MCU can be used in a SmartCard application. Some of the penalties of doing so, such as increased

die size and increased cost due to unnecessary on chip hardware and inefficient bonding pad layout, are immediately apparent; others will become obvious only when the security of such an application is put to the test, and fails to meet the necessary standards. The constraints imposed by the need for high levels of security at every stage of design and manufacture are considerable. Motorola has more than a decade of experience in volume SmartCard MCU production, and has the product portfolio required to meet all the needs of the SmartCard market in the world today. We are not standing still; we already have devices at advanced stages of planning and design to meet the emerging and future needs of this diverse market; we intend to remain ahead of the field.

GLOSSARY

BYTE (KBYTE)	8-bits of binary data (1024 bytes)	NVM	Non-Volatile Memory (for "permanent" storage)
CPU	Central Processor Unit – the "number cruncher" in an MCU	RAM	Random Access Memory – for temporary storage
DIE, pl. DICE	individual microcomputer (or other) device on silicon	ROM	Read Only Memory – contents fixed during manufacture of the silicon and unalterable "ROM Code" is customer supplied application programme
EEPROM	NVM – may be erased by applying special voltage	"SILICON GLEN"	An area stretching across the central belt of Scotland containing the factories of many international electronics companies
EPROM	NVM – may be erased by exposure to UV light	SMARTCARD	ISO credit card sized package containing a microcomputer
HCMOS	High-density low-power MOS technology	TEST MODE	Special operating mode for an MCU to allow comprehensive testing by the manufacturer prior to shipping to customer
I/O	Input/Output communication lines	USER MODE	Normal operating mode for MCUs (cf TEST MODE)
ISO (7816)	International Standards Organisation (standard concerned with specifications for IC cards)	WAFER	Slice of silicon which, after processing, contains typically hundreds of individual dice; 100-150 mm in diameter
MASK	Medium used to convert customers' application software (ROM Code) to a pattern on silicon		
MCU	(Single-chip) microcomputer unit		
MICROCOMPUTER	A silicon chip containing a microprocessor – plus memory and other peripheral devices		

SCAM modules for Smart Cards

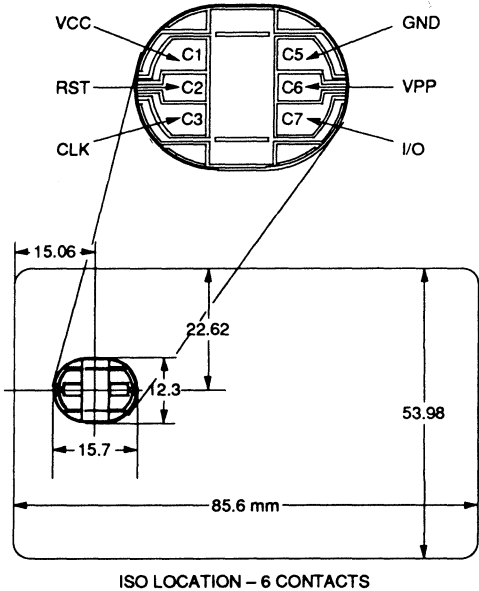
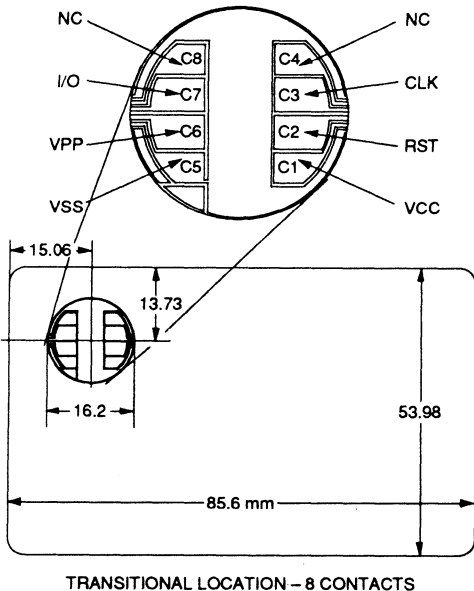
Motorola is launching its SCAM range of assembly modules by introducing its Smart Card product family packaged in modules suitable for insertion into ISO standard plastic cards).

8-contact and 6-contact modules will be available. These modules can be inserted in IS7810 plastic cards, in either the ISO standard location or the transitional location as defined in IS7816/2.

Both modules conform completely to the contact dimensions, locations and electrical connections defined in IS7816/2 (Dimensions and locations of the contacts).

	MEMORY (BYTES)			
	RAM	ROM	EPROM	EEPROM
6805SC01	36	1600	1024	-
6805SC03	52	2048	2048	-
68HC05SC11	128	6144	8192	-
68HC05SC21	128	6144	-	3008

Packaged in these modules, Motorola's secure micro-computer devices conform completely to the electrical characteristics defined in IS7816/3 (Electronic signals and transmission protocols).



During the assembly process, all pins are shorted together to minimise the risk of damage due to static electrical discharge.

Motorola is the leading supplier of secure microcomputers for smart card applications, with over 20 million units supplied to date from our manufacturing facilities in Scotland and the U.S.

“MEMORIES ARE MADE OF THIS...”

... a look at memory considerations for Smart Card applications

By Mike Paterson
 Motorola Ltd.
 East Kilbride

OVERVIEW

This paper discusses some of the issues concerning memory size and type and how they affect the specification of Secure Microcomputers for today's Smart Card marketplace. The principal premise is that: in the kind of mass market, multi-million unit per annum, application which is the main target of the Smart Card systems in use and under development at present, the cost is crucial to the success of a particular application.

From the silicon manufacturer's point of view cost is directly proportional to the area of silicon required (as a good first approximation at least). This paper therefore discusses the features and advantages of the different types of semiconductor memory, with particular attention to their cost in terms of silicon area, as they affect the typical uses required for Smart Card

applications. After discussing some of the possible trade-offs in the specification for a secure microcomputer there is a brief look at likely future developments in technology available for these devices. Europe leads the world in the design and implementation of Smart Card based systems, from bank payment cards to access control to medical records. Motorola echoes this trend through its unparalleled expertise gained from 13 years involvement in the specification, design and manufacture of secure microcomputer devices in Europe.

This paper presents an overview of the issues concerned with memory provision and utilisation for secure microcomputers used in Smart Card applications. It does not attempt an in-depth treatment of the technical nor the applications specific issues involved.

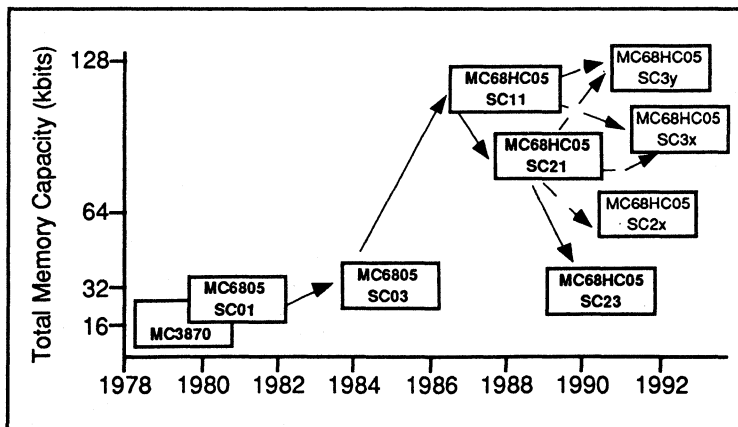


Figure 1: The evolution of Motorola's Secure Microcomputer family

INTRODUCTION

The drive towards more and more data storage, and hence bigger and bigger memory sizes, seems to be an essential part of all modern-day electronic systems. Only a few years ago desktop "personal computers" were confidently marketed, and thankfully purchased, with only 8 kilobytes (kb) of memory. Even today the ubiquitous IBM PC™ has a direct memory addressing capability of only 640 kb, whilst other desktop machines are now designed to have anything from 1 Megabyte (1 Mb = 1000 kb) to several Gigabytes (1 Gb = 1000 Mb) of storage. This increase in the amount of readily accessible memory has been made possible by enormous advances in system architectures and in the way data is stored. The semiconductor memory has come a long way from the invention of the transistor fewer than 40 years ago. Today single pieces of silicon no larger than a fingernail can store more than four million bits of data, and at a cost per bit so small as to be undreamt of when the transistor was born.

The earliest "computers" used mechanical storage means (gears and cogs); this swiftly advanced to the use of the vacuum tube in the first electronic computers. The next step was the ferrite core memory – until recently still a firm favourite with military systems designers the world over. However the development of the transistor saw the realisation of physically small memory cells each capable of storing a single bit of information, a '1' or a '0'. As the integration level (the number of transistors on a single piece of silicon) increased so then did the physical size of a given memory array fall. The decreasing physical size of memory has led inevitably in the market driven economy to a real fall in its price. Electronic goods are among the very few categories of consumer goods that have fallen in price in real terms over the last two decades and semiconductor memories have fallen (in price-per-bit terms) faster than any other commodity product.

The very success of memory design improvements and the ability to make the individual constituents of a memory cell ever smaller has in part fuelled the rush towards greater memory capacity. As more and more memory becomes available on a personal computer for example, so more and more sophisticated applications software becomes possible, which generates the need for large amounts of storage for the output of these very complex programs. The colour VDU is perhaps a good example, requiring as it does so much more memory than a simple black and white one: how often is 'colour' specified when black-and-white would be perfectly adequate, simply because it is readily available and has a perceived added value.

So memory is readily available, and cheap to buy, but does this necessarily mean that "more is better" is the right strategy to adopt? The danger is that the ready availability and low cost of memory may tend to obscure the benefits of making the optimum use of it. After all, no matter how small and cheap a memory cell becomes one cell will always be cheaper than two! The problem is that "more memory" is always the easy option when specifying a system and, as with colour VDU's, you then pay for it – whether you really need it or not.

These arguments are just as valid in the context of Smart Cards as they are in the example of the desktop computers quoted above. In addition, there is a further restriction when the silicon is destined for a credit card application: size. Size can become an issue not only of cost but of reliability, as plastic bends and silicon does not! Before we look in some detail at the particular needs of the Smart Card marketplace (and by Smart Card I mean by extension all application areas where a *Secure Microcomputer* is desired) we first need to understand a little about the different types of memory that are available.

TYPES OF MEMORY

There are basically two categories of semiconductor memory: volatile and non-volatile. In this context the terms volatile and non-volatile simply differentiate between a memory cell's ability to retain data in the absence of electrical power (non-volatile), or not (volatile). However both of these categories break down into sub-categories, each with different features and disadvantages.

We will look at the volatile memory first, as it is the simpler of the two, there being only two sub-categories. This type of memory is usually known as RAM (**R**andom **A**ccess **M**emory) and can be divided into two main sorts, Static (SRAM) and Dynamic (DRAM). Both types of RAM are very versatile: they can be written to and read from with no special preparations; the writing/reading is very rapid; there is no practical limit to the number of read-write cycles that may be performed. These features make this type of memory useful for storage of any kind of data, from fixed program data to rapidly changing data (such as time of day for a clock function). However there are disadvantages. RAM loses its information when the power is turned off, thus it is not useful for program and fixed data. RAM cells are also comparatively large (requiring as they do up to six transistors to store just one bit of information) and consume significant amounts of power. The former means that a given amount of RAM takes up much more silicon area than the same amount of any other type of memory. The

latter varies from being a nuisance in some circumstances to a very real drawback when considering, for example, battery powered equipment.

The two types of RAM vary in the way that they store data. Dynamic RAM is smaller in area than Static RAM but needs to be continually reminded of whether it is storing a '1' or a '0' by means of a regular refresh (or clock) signal. Since Static RAM does not need this signal to be provided there is less system overhead, but a larger area of silicon is necessary because of the increased cell complexity. In an application therefore the system designer needs to understand whether he will always have a clock signal available when he needs to maintain RAM data. If there is any doubt about this (for example battery powered equipment frequently shuts off system clocks, whenever possible, to reduce power consumption) then he needs to specify Static RAM, and pay the penalty of increased size, and therefore price. [Static RAM has itself two variants (using either four or six transistors per cell) which allow a trade-off between power consumption and cell size.]

Non-volatile memory (NVM) is a more complicated subject. There are three main types of NVM currently available in volume production quantities with other types just coming on-stream, or currently existing in low volumes. For now we will look only at these three main types, and towards the end of this paper discuss briefly the opportunities and challenges offered by the new technologies.

The basic feature of NVM is the fact that it retains its stored data even in the absence of electrical power. This makes it ideal for permanent and semi-permanent data storage, though as we shall see its versatility is extending its use towards the RAM's domain of frequently changing data. Again there are two basic categories of NVM: alterable and non-alterable.

ROM (for **Read Only Memory**), or mask-ROM as it is also known, is the simplest form of NVM. Its contents are defined during manufacture of the silicon and this data is then *unchangeable over the life of the device*. This type of memory has the smallest cell size (only one transistor) and is ideally suited to storing the fixed application software (operating system, program, fixed data tables etc.). However, there its usefulness ends. Since it cannot be changed after manufacture it follows that the only actions that it can perform and the only situations that the program stored in it can respond to are those thought of in advance and allowed for in the coding of the ROM contents. Because it is small and can be fully tested during silicon manufacture it is the best kind of memory to use if its features are sufficient. Even where an application demands the ability to change fundamentally during the life of the Smart Card there will be functions (such as input/output routines) which can be fixed, and hence coded in the ROM, leaving the more expensive alterable-NVM for data subject to change. There is one further advantage of ROM in the Smart Card context; because the contents are defined during manufacture, and cannot thereafter be altered, a level of security is

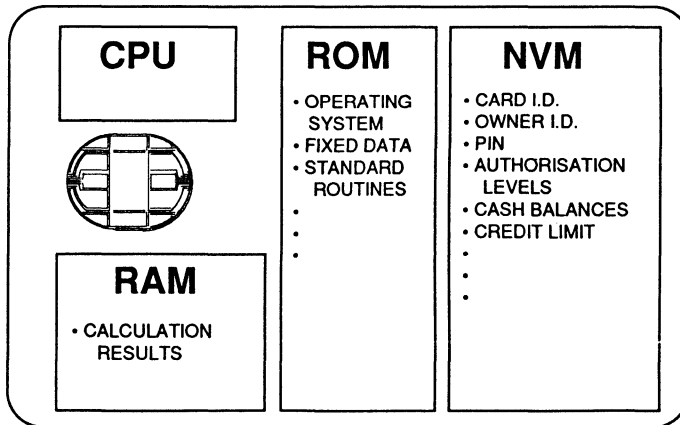


Figure 2: Some typical uses for the different types of memory in a Smart Card application

inherent in this memory. For example if an application program runs in ROM, and if this program is written such that it never allows the device to output a "secret code" stored on the card then there is no way to change this program to get the device to output the restricted data.

EPROM (Eraseable Programmable Read Only Memory) provides the next level of versatility over that of ROM. This behaves exactly like ROM when being read. However in certain circumstances the data stored in the EPROM cells may be changed. EPROM cells read as '1' or '0' dependent on the level of stored charge within each cell. Because of the construction of the EPROM cells it is possible, by using a comparatively high voltage, to change the state of the cell from, say, '0' to '1'. This is a one way process. In order to return the cell to its original, or virgin state it is necessary to erase the entire memory array by exposing it to ultra-violet (UV) light. The UV light causes a shift in the threshold voltages of the transistors in the EPROM array so as to allow the stored charge to leak away and the cells to return to their original (zero) state. Obviously by its nature an EPROM array will all be erased at once, it is not possible to selectively erase particular data. Again in the case of Smart Cards this limitation can be used to advantage. The (fixed) ROM code can be made to check that some particular part of the EPROM array is programmed (i.e. is non-virgin). On failing to detect this situation the applications program knows that the EPROM array has been erased at some point and hence the integrity of any data now in the EPROM is suspect. Further operation can then be suspended, or any other predetermined action taken. Clearly also in a Smart Card it is normally not possible to erase EPROM anyway as the die is permanently covered by opaque plastic. This means that for these applications EPROM is a 'write-once' or one-time-programmable' type of memory.

To get round the final restriction of non-selective erasure we have to move towards the most complex, largest, and most versatile type of NVM, the EEPROM array. Electrically Erasable Programmable Read Only Memory can behave exactly like EPROM but in addition offers the advantage of being able to selectively erase, and re-write specific bytes of data. Again the EEPROM cell, like that of the EPROM, uses stored charge to differentiate between a '1' and a '0'. Again like the EPROM, by use of a higher than normal voltage it is possible to change the state of the cell from, say, '0' to '1'. However this time the process is reversible and hence the cell state can be returned to '0' in this example by simply applying the correct voltages to the cell. A further advantage of EEPROM over EPROM is

the fact that because the programming/erasure currents involved are so small it is possible to generate the high voltage required for this on the silicon chip itself by using a 'charge pump', thereby removing the need for a separate high voltage power supply external to the chip. This is a significant design simplification and provides significant cost reduction in the Smart Card arena, where the number of external contacts is thus reduced from six to five. Additionally the Smart Card reader designer does not have to provide expensive, high tolerance programming circuitry like that necessary to support EPROM based cards.

However all this versatility has a price. As stated the EEPROM cell is more complex than any other type of non-volatile memory cell. It therefore takes up more space and hence costs more. Because EEPROM can be written to very easily it puts an additional burden on the system software designer in order that the integrity of the data can be guaranteed. For example: to return to our 'application running in ROM' quoted above. If this same application software is now run in EEPROM then it is possible to change the actions of the program (by simply rewriting some of the bytes containing that program) to provide an option that was previously non-existent. The software designer therefore has to ensure that his software guards against any possibility of an unauthorised user being able to gain control of the application program in order to make changes to its operation. Additionally further precautions, both in hardware and software, must be taken to minimise the chance of spurious data being written into the EEPROM, due to a power failure for example.

Because the EEPROM memory is so much more versatile it is consequently more open to abuse. This throws more dependence back on the inherent design of the hardware itself, as well as on the software. Security implemented as part of the hardware design of the device protects against a variety of ways that the memory contents could be compromised – for example by ensuring that there are checks after the device is reset to determine the contents of particular areas of EEPROM.

Alterable-NVM such as EPROM and EEPROM, by its physical nature, has a finite life in terms of data storage and therefore its integrity can be regarded as lower than that for the unchanging ROM cells. The "life" of EPROM or EEPROM is usually given in terms of data retention and write-erase endurance. Data retention is simply a measure of how long any particular information is guaranteed to be retained in memory after being written there with a given set of conditions (voltage, programming time, temperature etc.). Write-

erase endurance is quoted as the number of times that any individual cell may be written to and erased and still be guaranteed to correctly store data. Typical values are: 10 years for data retention; 10 000 write-erase cycles for EEPROM endurance. EPROM is a more mature technology than EEPROM, and needs comparatively high currents to perform the programming (electrical erase being impossible). Therefore the data integrity of EPROM is generally regarded as being better than that of EEPROM, there being a lower probability of incorrect data being stored (or good data being erased) through the action of "electrical interference" (power surge/failure, etc.). There are of course ways to improve statistically the effective reliability of data stored in EPROM or EEPROM, for example by using intelligent programming algorithms, or by using error detection and correction techniques. Each of these methods can be implemented so as to achieve the most demanding of data integrity requirements. Hence the recommendation should be that: use the flexibility of EEPROM if it is needed; but if "write-once" EPROM is sufficient for the nature of the data being stored then use it in preference; and if the data is fixed then use ROM.

We can summarise the memory types as follows:

- RAM** The fastest and most versatile type of memory; volatile
- EEPROM** Slower than RAM, but retains data in the absence of power
- EPROM** For Smart Cards this is really a write-once kind of memory
- ROM** Data defined during silicon manufacture and then unchangeable

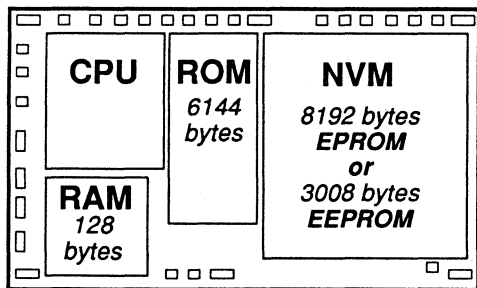


Figure 3: Outline of an actual 68HC05SC21 die showing the effect of cell complexity on the respective memory areas

MEMORY USE IN SMART CARD APPLICATIONS

Before we can look even briefly at the type of uses that memory is put to in a Smart Card application we first need to define "Smart Card". In this context I shall take a fairly narrow and literal definition. Thus the Smart Card is a device in ISO 7816 credit-card form containing a single-chip microcomputer. There are many other possible implementations which could give the same functionality, for example multiple-chip solutions, or key-fob shaped packages, etc.; the adoption of the above definition is merely to make the discussion simpler.

Whilst the possible applications for Smart Cards are limited only by the imagination of the systems designers in their ability to demonstrate added value, a number of common basic features are required. A Smart Card is by definition and intention an identification and authentication medium. No matter what actual use the card is being put to its fundamental purpose is to authenticate the card and its owner and to establish their level of entitlement; secondary functions such as payment, personal details, or data transfer are then used dependent on the nature of the application. This leads us to the basic common requirements of the secure MCU in the card: it should have the ability to store the identity of itself and its owner in such a way as to prevent the unauthorised use of this information. The method commonly, if not universally adopted at present for positive identification is through the use of a Personal Identification Number (PIN) together with a scrambling (or coding) function of some kind. Hence the individual card needs to have the ability to manipulate numbers according to some encryption system and to store in a secure manner unique data pertaining to that card and its owner.

In order to perform any data manipulation "scratchpad" memory is required. This is memory that may be written to and rewritten countless times during the progress of a single calculation or transaction. Since there is also a need for this memory to be accessed as fast as possible, to minimise calculation times, then the only choice is RAM. However, as I described earlier RAM is the most expensive kind of memory, due to its large physical size. This means that the system designer needs to pay a lot of attention to minimising the use of the "scratchpad" at any one time, to keep the percentage of the silicon area dedicated to temporary results (often of no consequence in themselves) as small as possible. The biggest use of RAM by far in a Smart Card device is for storing the intermediate results generated by the encryption or decryption algorithms and it is here that there is the biggest potential payback in terms of cost savings from efforts to optimise the use of RAM.

Alterable-NVM has the next biggest cell size after RAM, whether we are talking about EPROM or EEPROM. A fundamental requirement of a Smart Card, for whatever application, is that it can be "personalised" or customised for each individual user. This requires NVM since the individual data will have to be written immediately prior to the card being issued to a particular individual; and it will of course have to retain that data when the card is not in use and not powered up. The arguments between EPROM and EEPROM are complex ones. The significant functional difference between these two types of memory is that (at least in credit-card packaging) EPROM is a "write-once" medium, whereas EEPROM can be written and erased many times.

If write-once versus repeated write-erase was the only issue then obviously EEPROM would win every time. The flexibility of EEPROM is unchallenged. Where the exact future usage of the memory is not known and cannot be predicted then it offers the best choice. However if the use of the memory is known and it can in fact be used in a write-once mode then EPROM has a number of advantages, not the least of which is cost. Bit for bit EPROM is less than half the size of EEPROM, offers a higher level of intrinsic security (more resistant to unauthorised alteration

since erasure is not possible in a Smart Card application), and is a more mature technology with the associated benefits of reliability and predictability. By making use of individual EPROM bits the number of transactions that can be sequentially recorded before the memory fills up can be quite large, however then the card has to be thrown away whereas the EEPROM card can simply have obsolescent data erased as necessary, and the memory re-used.

There are two main uses for the NVM: frequently updated data (such as information on every transaction); and only occasionally updated data (such as a new PIN code, credit limit, personal details, etc.). For the latter category the availability of more than twice as much EPROM memory in the same area may well make it equally or even more attractive than EEPROM. In the former situation however the ability to continually "refresh" the EEPROM card's capacity is an advantage, avoiding as it does the "throw away card" concept. However it should also be remembered that the lifetime of a credit card is finite and controlled by such considerations as plastic wear, customer expectations, and even security. A mixture of EPROM and EEPROM functionality would appear to be desirable, and this development is discussed later.

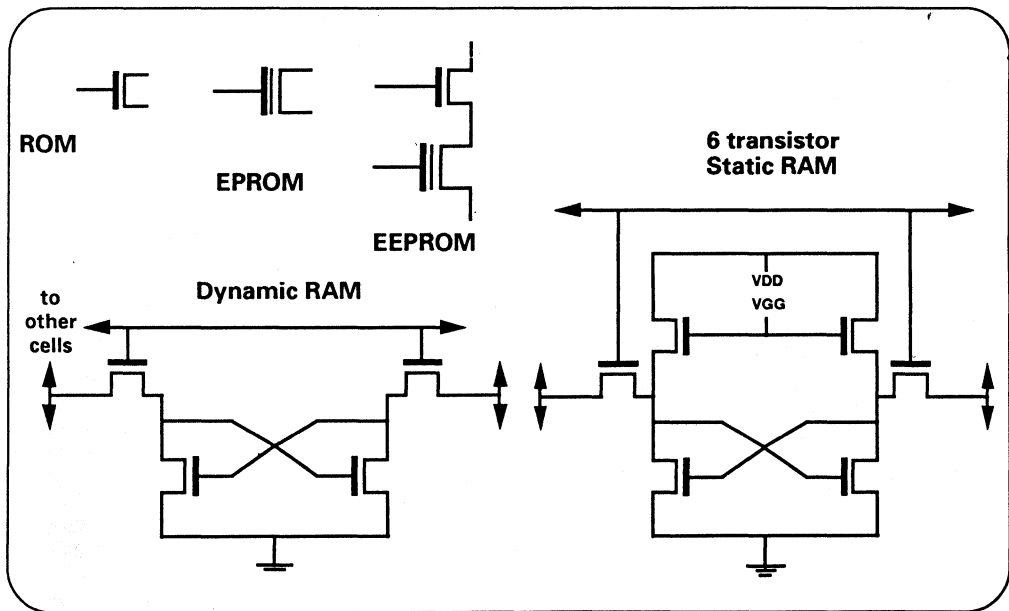


Figure 4: A comparison in schematic form of the circuitry involved in implementing ROM, EPROM, EEPROM, DRAM and SRAM cells in MOS technology suitable for microcomputers

The final memory type is the fixed ROM which can contain the applications program and general routines for input/output, PIN change etc. There is at least the temptation (and not only in Smart Card applications) for system designers to use the versatility of EPROM & EEPROM almost as a crutch, enabling last minute changes to software to be made "painlessly". Whilst this does mean that the need for fully functional final software is postponed as long as possible, this delay does carry a price as we have seen, in terms of silicon area. It makes economic sense to ensure that as much of the software as possible is fully defined before the manufacturing of the die starts. In this way the fixed data can be most efficiently stored on the silicon while enabling the available amounts of alterable-NVM, be they EPROM or EEPROM, to be reserved for functions that really require its additional features. In a typical Smart Card type of application we might reasonably expect to see the following apportionment of the software in non-volatile memory: ROM – fixed or "core" software (e.g. I/O routines, read/write sub-routines, basic encryption algorithm, planned use software (e.g. bank debit card); EPROM/EEPROM –

sections of the code which it may be desired to update on a 'regular' basis (e.g. algorithm seeds, precise flow of the algorithm, tracking credit balances & expenditure profile) or which are unique to each individual card (e.g. serial number, personalisation data). EPROM or EEPROM can also be used to allow new functions to be loaded into the card *after it has been issued to individual users* (e.g. medical data, use as a credit card, telephone prepayment card, access control). By ensuring that the basic routines and functions are stored in ROM it is then possible to change the exact way in which they operate by simply writing different linking software to be stored in the E/EEPROM. For example it would be possible to operate the data transfer to and from the card at different rates based on a single number stored in E/EEPROM used in conjunction with the basic routine in ROM.

Figures 4 and 5 show in some detail the circuitry involved in typical memory cells as they are constructed on microcomputer devices together with a comparison of the actual layout areas on silicon for some of these memory types.

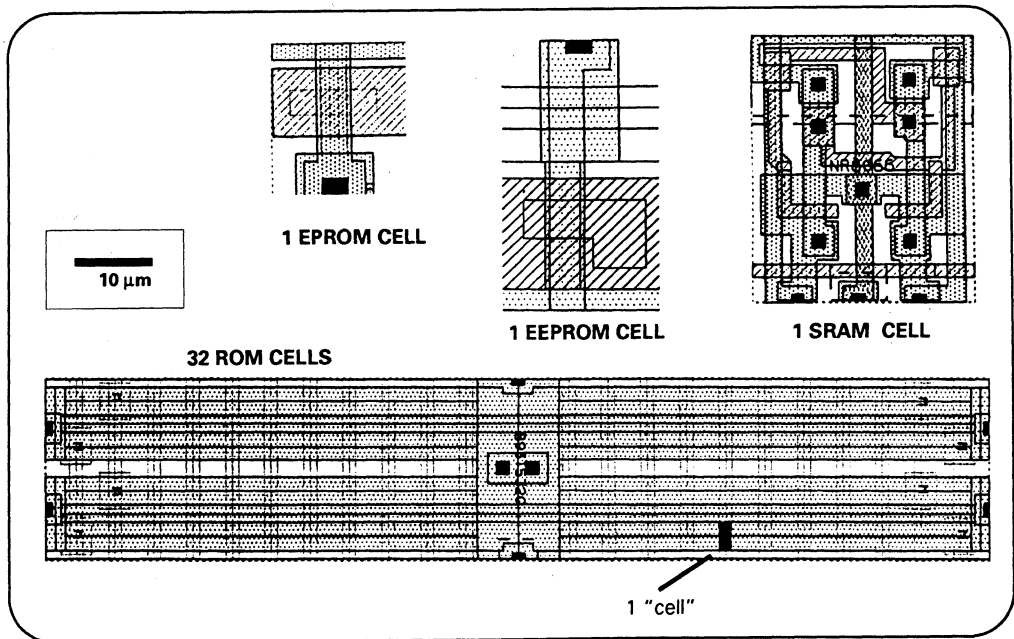


Figure 5: A comparison in terms of the physical layout on the silicon chip of the circuitry involved in implementing different types of memory array

VERSATILITY versus COST: THE MEMORY TRADE-OFF

If we take the size of a given ROM array to be 1 then the relative sizes for the other types of memory array are approximately: EPROM-3; EEPROM-7; dynamic RAM - 15; static RAM - 30. The reasons for this increasing size are simply the increasing complexity of the cells needed to provide the features of each type of memory, *plus all the associated driving and decoding logic.*

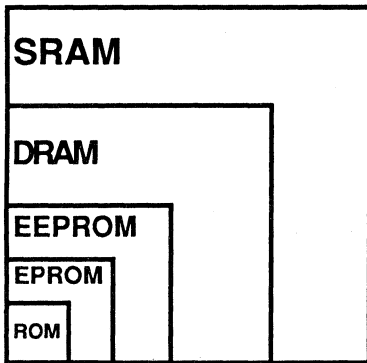


Figure 6: Relative memory cell sizes

THE FUTURE ?

Clearly the Smart Card market, and indeed the market for *secure microcomputers* as a whole is still in its infancy. There are a number of very large projects already well underway - in France, Germany, Norway, Switzerland and the U.K. for example, but there are countless more at the stage of advanced planning or preliminary trials. The future for the actual microcomputer chips themselves will be largely determined by the way this marketplace develops. Criteria such as die size, memory size, level of designed-in security, hardware features and so on will all be dependent on which of the many proposed applications actually stay the course and become volume users of this technology.

Without the ability to predict the future and say which will be the most significant uses of Smart Card technology in the 1990's and beyond all we can do is look at current trends in both technology and applications and assess their likely development in the next few years.

It is probable that we will see a reduction in the (apparent) homogeneity of the market; there will be requirements for very large memory array devices, which will be comparatively expensive (multi-function financial cards for example); there will also be a need for very cheap "stripped down" devices, aimed at providing a single, well-defined service very economically (perhaps a tracking token used for manufacturing control, or a simple access control card). Whilst it would appear at present that the latter category has the greater potential for generating very high volume business it is already clear that one of the major strengths of the "Smart Card" is its ability to be a truly multi-purpose card - where further new functions (and even distinct applications) can be added even after the cards have been issued to the individual end-users.

However, to return to the topic of this paper, the short-term silicon technology developments which we can expect to see in support of the above market forces are basically to do with putting more and more functionality in a given area of silicon. Thus memory cell sizes will continue to shrink, and a significant reduction in this size could be in the use of "flash memory technology".

Flash-EEPROM is a modification of the existing EEPROM technology which reduces the effective individual memory cell size by abandoning the ability to individually erase specific bytes of data. In order to reduce the connection overheads for each cell the flash-EEPROM can normally only be erased in bulk (i.e. the entire array) or as a relatively small number of fairly large data blocks within the array. By making this seemingly simple change it is now possible to get almost as many bytes of flash-EEPROM memory as EPROM memory in a given area of silicon. It must be borne in mind though, that flash-EEPROM is a compromise technology: it will not be as small (hence not as cheap) as EPROM; it will not have the high write-erase cycling endurance of true EEPROM (perhaps only 10 - 100 changes of data, rather than the typical guaranteed figure of 10 000 for the EEPROM); nor will it have the selectivity to be able to erase/rewrite single bytes of data, with all the speed advantages that implies; finally it will require an external high voltage source to perform the erase as the bulk nature of this erase precludes the use of an internal charge pump, since the required current is too high. As far as the Smart Card type of applications are concerned then flash-EEPROM will give the system designer some of the benefits of EEPROM erasability, with a cell size (and therefore cost) approaching that of EPROM.

Just becoming available on the market now is a new generation of microcomputer devices which combine both EPROM and true EEPROM on the same die. This mixing of manufacturing technologies has not been trivial but it offers unparalleled versatility to the system designer: he has the write-once kind of changeable memory to add, for example, a new application or a correction to an unforeseen limitation of the software; he also has the full write-erase cycling capability of the EEPROM to cope with frequently changing data, such as credit balances, entry logging etc. Because both types of memory are on the one piece of silicon it is possible to specify the minimum of the more complex and expensive EEPROM as it will only be used for data that changes many times and will not have to cope with the change-once data also, as the current EEPROM only devices have to.

Other developments for the near future which will affect the memory requirements of the secure microcomputer are likely to be in the field of algorithm handling. As we move to more and more sophisticated algorithms so the amount of processing power required increases and also the amount of working storage (RAM) necessary. It is presently a challenge to the cryptographers to cram the interim results of their encryption or decryption into the available RAM. Since RAM is expensive on silicon the pressure is unlikely to ease. Further advances in RAM technology and further shrinking of the cell size will increase the amount of available RAM; however by then the algorithms will require more working storage to cope with their ever increasing sophistication.

Finally we can expect to see the "contactless Smart Card" becoming more common as the complexity and cost problems of this technology are overcome to enable it to compete in the volume marketplace. Presently the potential disadvantage of contactless solutions is that the card has to contain at least two chips, with the associated issues of manufacturability, cost, reliability and security. One development that is firmly in the silicon manufacturer's court is to devise the means to combine the standard HCMOS process technology used for microcomputers with the RF technology needed for the transceiver function of the contactless card. Once this can be achieved then a single-chip contactless Smart Card will be a reality for the high volume low cost general marketplace, and at a stroke many of the interfacing and mechanical contact issues will be circumvented. This is not likely to happen in the immediate future, but if the demand is real then it will happen.

So in summary we can see that there is enormous scope for development of the secure microcomputer. As more flexibility becomes readily available this will of course make more and more potential Smart Card applications viable. The credit card format is a very acceptable one to the end-user, and it may well be that this is the shape of things to come. However one of the reasons that the Smart Card is so acceptable to users is because of the enormous numbers of "credit cards" for a wide range of services already being carried around in people's wallets. When the only kind of "credit cards" are those containing a secure microcomputer perhaps we will see a new, more compact and more durable form of personal identification medium.

MEMORY TYPE FEATURE	ROM	EPROM	EEPROM	RAM	FLASH EEPROM
Relative array size on silicon	1	3	7	20–30	3–6
Number of write-erase cycles	1	10–20	>10 000	∞	100
Data retention time	∞	10 yrs	10 yrs	as long as power on	10 yrs
Program voltage	N/A	external	internal	(internal)	<i>external</i>
Erase voltage	N/A	UV	internal	(internal)	<i>external</i>
Write time	N/A	5 ms	10 ms	bus speed	2–10 ms
Erase time	N/A	minutes	10 ms	bus speed	<i>seconds</i>

Figure 7: A comparison of the principal features of each memory type

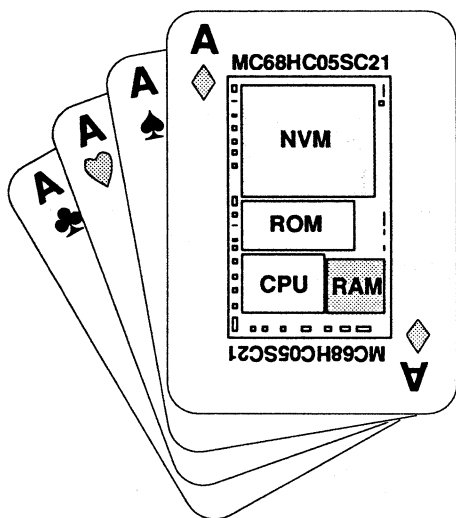
GLOSSARY

BIT	a single item of information; either "0" or "1"
BYTE	eight bits of data
DRAM	Dynamic RAM – requires a refresh, or clock, signal
EEPROM	Electrically Erasable Programmable Read Only Memory
EPROM	Erasable Programmable Read Only Memory
kb, Mb, Gb	kilo- (1 024 bytes), Mega- (1 048 576), Giga-byte (1 073 741 824)
FLASH	"Flash EEPROM" – a limited form of EEPROM
HCMOS	High density Complementary Metal Oxide Semiconductor; a manufacturing process for MCU's
ISO	International Standards Organisation
MCU	Micro Computer Unit
NVM	Non-Volatile ("permanent") Memory
PIN	Personal Identification Number
RAM	Random Access Memory: temporary storage
RF	Radio Frequency

SMART CARDS: how to deal yourself a winning hand

By Mike Paterson
Engineering Manager for Secure Microcontrollers
MOTOROLA LTD., East Kilbride, Scotland

INTRODUCTION



Motorola East Kilbride has been the company's world-wide "Centre of Excellence" for Smart Card products for more than ten years. Together with their Geneva Design Centre they have developed and supported a range of secure single chip microcontrollers for Smart Card and other security applications. To date more than 20 million secure MCUs have been sold and Motorola now has customers throughout Europe, Japan and the Americas using its Secure MCU product range. The author has worked in Product Engineering for several years on supporting this range of device types.

What is a "Smart Card"? There are many definitions for this term, probably one for every article ever to appear in the popular press! However, before I define just what I consider to be a "Smart Card", I would like briefly to explain my choice of title and summarise what I hope to cover in this Engineering Bulletin.

For a number of years it has been fashionable to consider Smart Cards as a "solution looking for a problem", or as an "invention searching for a market". These are now becoming less fashionable expressions as the market for Smart Cards grows and diversifies. However, it is this historical basis and the very diversity of potential applications that are now beginning to come to fruition, which are in some ways hampering the growth of the marketplace.

This Engineering Bulletin sets out to explain, from a silicon manufacturer's perspective, the background to the current market and the variety of products that are on offer. It defines the "Smart Card" and looks at the ways to determine what a particular application needs, in terms of features, if a Smart Card solution is going to succeed for it. By its nature this Bulletin provides only an overview of the many, and complex, arguments that accompany any decision to introduce a Smart Card application, but it is hoped that this will give some basic guidelines to help the prospective end-user to ask his supplier the right kind of questions about the product types available. By choosing the optimum product for the particular market the likelihood of commercial success in that market is enhanced. So, by asking the right questions you end up with the right product. Remembering that (like Poker) you do not see your competitor's hand until it is actually played, it is clearly advantageous to be holding all the Aces before you place your bets on success.

WHAT IS A SMART CARD?

The simplest definition might be that it is capable of "thinking" for itself – in other words, it has built-in computational ability within the credit card itself which can modify, and even create, data in response to external stimuli. This, then, distinguishes the microcontroller (MCU) based card from all other types. This definition, and its isolation of the MCU based card, is the one I am going to use throughout. All other cards are therefore "dumb" as opposed to "smart". However, it is only fair to say that there are varying degrees of "smartness" and "dumbness". The optical cards for example, with their vast memory capacity, are capable of deploying great amounts of data in order to enhance their operation. However their operational responses are rigidly defined and they cannot create any data: that requires an external "computer" to be connected on-line.

TYPES OF "CREDIT CARD"

The magnetic stripe card has been around for many years now and it is typically against this bench-mark that all new card technologies are measured. Before we look at how the Smart Card compares with the magnetic stripe card, it may be useful to list what types of card are available and look briefly at their advantages and limitations.

The types of card available are:

1. Embossed plastic (with or without hologram or other visual security feature)
2. Card with magnetic stripe containing personalised/ security data
3. Card with optical storage medium
4. Card with physical/mechanical storage medium
5. Card with silicon chip
 - a) memory only
 - i) EPROM (write once)
 - ii) EEPROM (multiple write)
 - iii) Battery-backed RAM
 - b) MCU (with various memory types)
 - c) multi-chip ("contactless cards" etc.)

Of course not all of the above categories are exclusive, indeed many cards currently are a combination of (1), (2) and (5).

As Table 1 shows, the main differentiators between the various types of card are: cost; storage capacity; versatility; and security.

Type of Card	Cost	Capacity	Versatility	Security
- embossed plastic	low	nil	nil	nil
- with hologram	low	nil	nil	low
- with magnetic stripe	low	low	low	low
- with non-volatile memory	medium	medium	low	low
- with battery-backed RAM	high	medium	medium	low
- with MCU	high	medium	high	high
- with multi-chip solution	high	medium	high	high
- with optical storage	medium	high	medium	medium
- with mechanical storage	low	low	nil	nil

Table 1: A comparison of Card types and their features

SMART CARD VERSUS MAGNETIC STRIPE

The Smart Card IS NOT a replacement for the magnetic stripe cards that are commonly to be found in all our pockets, at least not directly. The magnetic stripe card has two overwhelming advantages: it is cheap; and it is established – there are hundreds of thousands, if not millions, of standardised magnetic stripe card readers all over the world. This second advantage is, of course, held over all possible competition, and is a temporary hurdle, which will ensure that the new standard (whatever it is) really does offer some distinct advantages! However, on the issue of card cost alone, it is quite safe to say that the MCU based Smart Card will never be as cheap as a magnetic stripe card.

Given the above facts it is essential to look for value added from a proposed Smart Card implementation. This may either be direct, such as enabling a previously impossible application, or indirect, for example in reducing losses due to fraud or misuse.

So, the Smart Card is more expensive than a magnetic stripe card. On the other hand, it surpasses the magnetic stripe card when any other feature is considered. It has many times the data storage capacity, it offers a much higher degree of transaction security and data integrity, and it is flexible – it can support more than one type of transaction, and can be reconfigured after it has been issued to the end-user. If your application needs only the features of the magnetic stripe card then it is unlikely that a Smart Card system will succeed for you. If, however, the application would benefit from the increased flexibility for example, and this can be quantified, then it is much more likely that the system will be commercially viable.

FEATURES OF A SMART CARD

As I have already said, the benefits of the Smart Card are only relevant if the proposed application gains something from them; it is not necessary to play an "ACE" to beat your opponent's "TWO", a "THREE" WILL do. Nevertheless, let us look at the strengths of the Smart Card, then later we can determine what a specific application actually needs.

I would list the principal features of Smart Cards as follows (in alphabetical order, rather than by merit!):

- Anti-fraud Capability
- Continuous Application and Transaction Validation
- Cost

- Flexibility
- Multi-purpose
- Off-line Capability / "Stand-alone"
- Positive User Authentication
- Reconfigurable In Use
- Security
- Speed
- User Friendly

As you can see, I feel that even overall cost can be considered a feature, when looked at in terms of initial costs, operating costs and cost effectiveness!

Many of the features are, of course, inter-dependent and so I will try to discuss the overall impact of all of them as well as looking at them individually.

Because of the Smart Card's ability to perform its own validation checks, and to store details of transactions internally, it is at once secure and capable of operating in a "stand-alone" mode, that is, with no need to communicate with a central computer/database for every transaction. It is this ability to store data securely, and to operate on that data and on external data within the card, that distinguishes the Smart Card from all other "data cards".

Security in the application is on many levels. There may be the normal software controls, such as passwords or PINs. There are hardware monitoring functions on the silicon itself (to protect against bulk erasure of data for example). There are also the specific hardware features of a Secure Microcontroller which effectively prevent access to data stored on the chip, except in the very limited way permitted by the application software, to ensure that data cannot be selectively modified. By making sure that all the links in the security "chain" are strong ones it is possible to rely on the integrity of the data stored in the die – and hence to perform authentication and validation tests within the card. The benefit of this feature is that the card can perform its own validation of any system it is connected to. This places requirements in terms of security and traceability considerations from the initial design of the chip itself, through silicon manufacture, to system software, and ultimately features of the application.

Given that the data storage is accepted as being secure, the card can then retain information about, for example, a person's credit balance (and balance remaining at a given time) as well as storing PIN data

or passwords, to avoid the need for a central computer link-up. It can also store usage patterns, voluntary limitations of use, and any number of specific details – so that it becomes possible, for example, for the card itself to initiate a fraudulent use check if the usage changes suddenly. The internal computing capability may also be used to detect fraudulent terminals, by continuously exchanging information cryptographically and so validating the terminal to which the card is connected.

ANTI-FRAUD CAPABILITY

Because there is intelligence in the card, if it is used in a Credit Card application it can prevent overspending by its authorised users, unlike the conventional magnetic stripe card. This is simply because the credit limit and the current balance can be stored on the card, making the response time to overspending immediate and independent of human interaction.

Biometric data is perhaps the most visible of the new techniques being employed to avoid fraudulent use. Though by no means restricted to Smart Cards, the ability of the card itself to allow for changes in response (for example voice change due to a cold) make it a powerful tool. The basic advantage here for the Smart Card is that it can compare stored data with actual data in complex ways, and relate many different pieces of information. For example, there are always tolerances allowed for in biometric data. The card can monitor and update tolerance levels as it “sees” your signature change over a period of time. It can also ask for reconfirmation of identification in cases where a large amount of money, or entry into a high security area, is at stake, especially if the biometric data is on the extreme limit of acceptability.

In addition to all the security features on the silicon and in the software, it is possible for the card to store “typical usage” data. This then allows for the possibility of triggering a “fraudulent use check” if the usage suddenly changes.

CONTINUOUS APPLICATION AND TRANSACTION VALIDATION

The card can re-verify at any time that it is connected to a legitimate system and application by requesting security data from either the user (eg: biometric or PIN data) or the system (eg: en/decryption keys).

COST

Clearly the unit cost of a Smart Card is much higher than, for example, a magnetic stripe card. However, the initial cost can be offset by such factors as: the ability to reconfigure the card after issue; the capacity to store data for several distinct applications (eg: parking meter, access control, prepaid ticketing...); reduced loss of issuer income due to fraudulent use or fraudulent duplication; and the ability to function in stand-alone mode, thereby reducing system dependence on expensive, permanent and rapid access to central computing facilities for authorisation and validation etc.

FLEXIBILITY

The card can have several distinct applications stored on it at once, with safeguards to ensure that there is no unwanted “sharing” of data. It can be re-programmed (for example with a new credit balance, or even an entirely new application and entitlement) after issue to the end user – without recalling the card.

The exact way in which the card and the system interact is controlled by the software stored in the system and in the card. Obviously the system software can be changed in any card-based system. An advantage of a system based on a Smart Card is that the card software, and hence the form of its responses too, can be changed at any time, by simply reprogramming part of the non-volatile (i.e. the changeable) memory in the card. This means that any operating deficiencies can be corrected, or any newly conceived and valuable features can be added.

MULTI-PURPOSE

The number of applications installed on a card at one time is limited only by memory space. The card can respond “intelligently” to question-and-answer sessions enabling it to deal with an indefinite number of different application protocols.

OFF-LINE CAPABILITY / “STAND-ALONE”

The card can validate both the user and the system. It can store credit balances, area access authorisations and records of recent transactions. Standard transactions (perhaps defined as fitting the usage pattern and lying within predefined limits) may be authorised, completed and recorded, for later

communication to a central computer. "Unusual" transactions can still require the card to seek external authorisation in certain circumstances if required.

This capability both reduces cost and increases transaction speed.

POSITIVE USER AUTHENTICATION

Biometrics are beginning to be used, the limitations at present being principally memory size and response time. Even without this feature, however, there is the simple benefit of the user being able to define his own PIN number, including the length of it and its exact format, and being able to redefine it when and where he wants. There are, of course, varying degrees in-between these extremes of PIN and biometrics.

RECONFIGURABLE IN USE

If an application is compromised, or just upgraded (an extra subscription TV channel for example), then this facility can be uploaded on to the card the next time it is used with access to a host computer. This means that it is transparent to the end user, as well as being easy and cheap to accomplish – and fast.

SECURITY

This technology can increase security in many ways. These include designed-in chip security features (such as fusible links and illegal frequency detection circuitry); reliable tracking and safeguarding, through a controlled manufacturing process; and features written into the user software (such as a check on Reset to ensure that a particular byte of data is present). All this is possible, and moreover, because the hardware security is there it is possible to build up to very high degrees of additional software security.

SPEED

Waiting for a telephone link to a central computer is both expensive and slow. Local transaction processing is cheaper and faster. It is also possible to respond rapidly to a new business opportunity (such as being able to use your telephone card in another country, now that there is an agreement on national standards) by simply updating the cards already issued, rather than having to offer new ones, with all the related cost and time delay implications.

USER FRIENDLY

The user can: change PIN data; update personal data; customise operation; define voluntary limits; find out remaining credit/outstanding debit balances... In other words he can feel much more in control of the way the card works, which adds to its perceived convenience.

WHAT DO YOU NEED?

As I have already implied, all these features are only worth paying for if they pay for themselves.

As in every area of commerce we have now to look at Cost versus Features. In this particular area past experience has shown that one of the critical parameters to consider is the cost of failure. This means, for example, how much of your revenue will you lose if the security of your product is compromised; or, how much bigger is your potential market if you offer this additional feature (or conversely, how much bigger will your market be if you forego this feature and reduce the price)? The inherent security of the Smart Card solution is advantageous here as it allows you to plan your revenue with much greater confidence.

From the silicon manufacturer's viewpoint the cost issue is quite straightforward; cost is roughly proportional to area of silicon. To minimise the cost of the chip, therefore, you have to trade off features which take up significant amounts of silicon against their expected payback in use.

The biggest single element of any microcontroller die is the memory. The memory area can typically take up to 80% of the total die area. This is the first region where you need to look for cost effectiveness. Amount of memory, and type of memory are crucial. By considering the cost of memory from the outset it is possible to structure at least some of a system's memory requirements to make use of the most compact types of memory. For example, the more of the system software that can be fixed (and therefore stored in ROM as opposed to EPROM or EEPROM), the bigger the potential savings. Of course this does reduce flexibility from the point of view of being able to completely rewrite the system software in the card after it has been issued.

The frequency of change of the data determines the type of memory required. Small amounts of rapidly changing data will be stored in the relatively large RAM cells; EEPROM can cope with large amounts of data being updated several thousand times; the smaller EPROM cells can only cope with one change of data; and the data in the (physically smallest) ROM cells are fixed during manufacture.

Reliability and data integrity is also an issue. All changeable media, be they semiconductor, magnetic or otherwise are subject to loss of data. This again leads to the use of "permanent" (i.e. fixed, ROM) memory where possible. Once the data in ROM has been verified during manufacture, its contents are known and cannot change. For RAM, EEPROM and EPROM the functioning of the cells can, of course, be checked during manufacture but it is impossible to predict the behaviour of an individual cell under the widely varying conditions and long time periods typically to be found in a Smart Card system. It is, therefore, wise to incorporate into the system software (and perhaps the hardware too) a variety of error detection and correction techniques, exactly as is always done with magnetic recording on disk or tape. In this way the reliability and data integrity perceived by the system user is much higher.

Having considered the pros and cons of the various memory types, and also looked at the benefits of offering particular features in a specific system the next step is to optimise the card specification, and hence the silicon specification. Much of the operating code can be fixed from the outset. Where it is necessary to retain flexibility, control of how the fixed routines are linked together, and the exact parameters used, can be stored in alterable memory; this lets cards be changeable up to the point of issue, and beyond. The security of the overall application is controlled by features of the silicon and of the software. Having looked at the costs of the system being compromised it is easier to assess what degree of security is necessary. By writing the software to include a unique identity for every card it is possible to minimise the potential loss if one card is stolen; in other words, the fact that one card is compromised does not necessarily compromise the system. This effectively makes your system immune from illegal card duplication.

Communication between the card and the external system is well defined for ISO type systems. Nevertheless, it is possible to optimise the operation of your card for the desired application, so that for example, the most efficient data transfer rate and format is used for the kind and quantity of data that you are typically dealing with.

Another feature which may prove valuable in certain applications is the ability of the silicon manufacturer to identify each die uniquely during manufacture. This provides traceability information, which can have many uses: serial number; manufacturing date (plus manufacturer, test conditions,...); a unique encryption key; etc. Because this data can only be written during manufacture, and thereafter is fixed, it enables a reliable user identification and authentication arrangement, whereby individual users can be traced back to individual cards with no possibility of error. Such data can also prove useful in analysing failure patterns, as it allows the contribution of the silicon to be evaluated very quickly, by showing for example that it is all from one manufacturer.

Finally, I return to the cost of failure. What is the "unit worth" of one of your proposed cards? If it is a card which may be used as an "electronic purse" and be preloaded with up to £1000 then the unit worth is apparent. Here it is obviously of paramount importance to ensure that the current balance cannot be tampered with, or recorded erroneously. There is considerable need to ensure that the reloading of a new balance is protected by very high security procedures. The capability of the Smart Card to perform system validation, and respond to authentication requests from the system, is crucial. If however the card is used as a "subscriber authorisation" for something such as Pay TV then the "unit worth" of a card is fairly low – a few pounds a month for the validity period of the card. Even in this circumstance it is very important that the overall integrity of the system cannot be challenged. If, for example, it is possible, by "breaking in" to one card, to compromise the whole system (by, for example, being able to issue many fraudulent cards) then the unit worth of the individual card is very high. By putting in place appropriate security and traceability features on the card it is possible to ensure that defrauding one card, if it were possible, will be limited to that one card. Hence, for security and revenue reasons, it is generally desirable to limit and minimise the unit worth of an individual card.

THE RIGHT QUESTIONS – to ask about the silicon and the system

If we assume that you have asked yourself the right questions about what your application really needs to succeed, then we can turn to the questions you should ask your card or system supplier – to make sure that the features you have decided are important to you are encompassed by the system you finally get.

SECURITY – is my revenue safe? Is my reputation safe? What features of the silicon/card/system guarantee the particular security needs of my application?

MEMORY SIZE – options and costs?

VERSATILITY – Ease of use and expandability of the system?

DELIVERY – when can I have samples? ...in volume?

QUALITY – can I depend on the product specification?

DEPENDABILITY – supplier's track record and guarantees?

UPGRADE PATH – supplier's history and plans?

COST – is it competitive? What am I paying for? Do I need it all?

As with buying anything, if you know what you want and why, there is a much better chance that you will get it!

THE FUTURE

From the point of view of silicon for Smart Cards there seems no reason to doubt that progress will continue. The silicon chip will continue to get smaller, cheaper, more versatile and with a wider range of features. Likely enhancements in the not-too-distant future include:

- wider supply voltage range (< 3V to > 6V?)
- lower power consumption
- faster clock speeds
- more memory
- mixed memory (EPROM & EEPROM)
- flash EEPROM
- greater system integration (e.g. RF transceiver elements on same silicon as the MOS micro-processor and memory)

There will also be a drive towards more and more complex cards, to support many applications, or to support a very sophisticated real-time data processing application, perhaps for example involving complex biometric data. This means that, in addition to the above mentioned "smaller, cheaper" trend there is a simultaneous trend to "bigger, faster, more powerful" (and therefore unfortunately, more expensive). This may give rise to a slightly different set of advances in silicon for Smart Cards:

- real-time processing capability (> Digital Signal Processors?)
- dedicated hardware encryption engines on silicon
- parallel data I/O
- much more memory

Of course there is no reason why all future developments need conform to the "credit card" format. Indeed, we are already seeing applications for "secure microcontrollers" which are built in to systems and may therefore be more conventionally packaged. However, it is fair to say that, for consumer applications, the CARD has much in its favour, whether it is with or without contacts. The package problems encountered with the "credit card" are likely to become less and less significant as power consumption continues to fall and as chip architecture continues to shrink.

IN SUMMARY

Know what YOU need – specify it, and insist on it.

As a service provider, only you understand all the needs and limitations of your business. It is easy to ask for "some of everything", but this will not necessarily make your system succeed. If you have quantified the benefits and shown that you do indeed need a Smart Card system, then make sure that it is optimised and specified for its intended use. You can then ask the vendors what they can offer to meet your specifications – and you can ask them the right questions to determine whether what they are proposing is indeed suited to your application. Extra features mean extra costs, and so need extra revenue.

If you quantify the benefits and optimise the product specification to bring about these benefits then you have done the best with the cards dealt to you. If in addition you ask the right questions of your suppliers then you may even draw an ACE.

MC68HC705T3 Bootloader

By Peter Topping
Motorola Ltd., East Kilbride, Scotland

INTRODUCTION

Figure 1 shows the circuit required to program the EPROM of an MC68HC705T3 from an external EPROM. There is a direct correspondence between the addresses in the external EPROM and the memory map of the MC68HC705T3. The OSD characters should be in the area \$0400-\$0AFF, the program between \$2000 and \$7EFF and the vectors between \$7FF0 and \$7FFF. It should be noted that these addresses must be used even if the MC68HC705T3 is being used to emulate the MC68HC05T1 or MC68HC05T2. The addresses of the vectors (and the program start address for the MC68HC05T2) must thus be changed from their normal position when emulating these devices. It should also be remembered that MC68HC05T1 and MC68HC05T2 emulation should not employ any resources not present on the target devices. The MC68HC705T3 has more OSD buffers (2 rows) and characters (112), more RAM and more ROM.

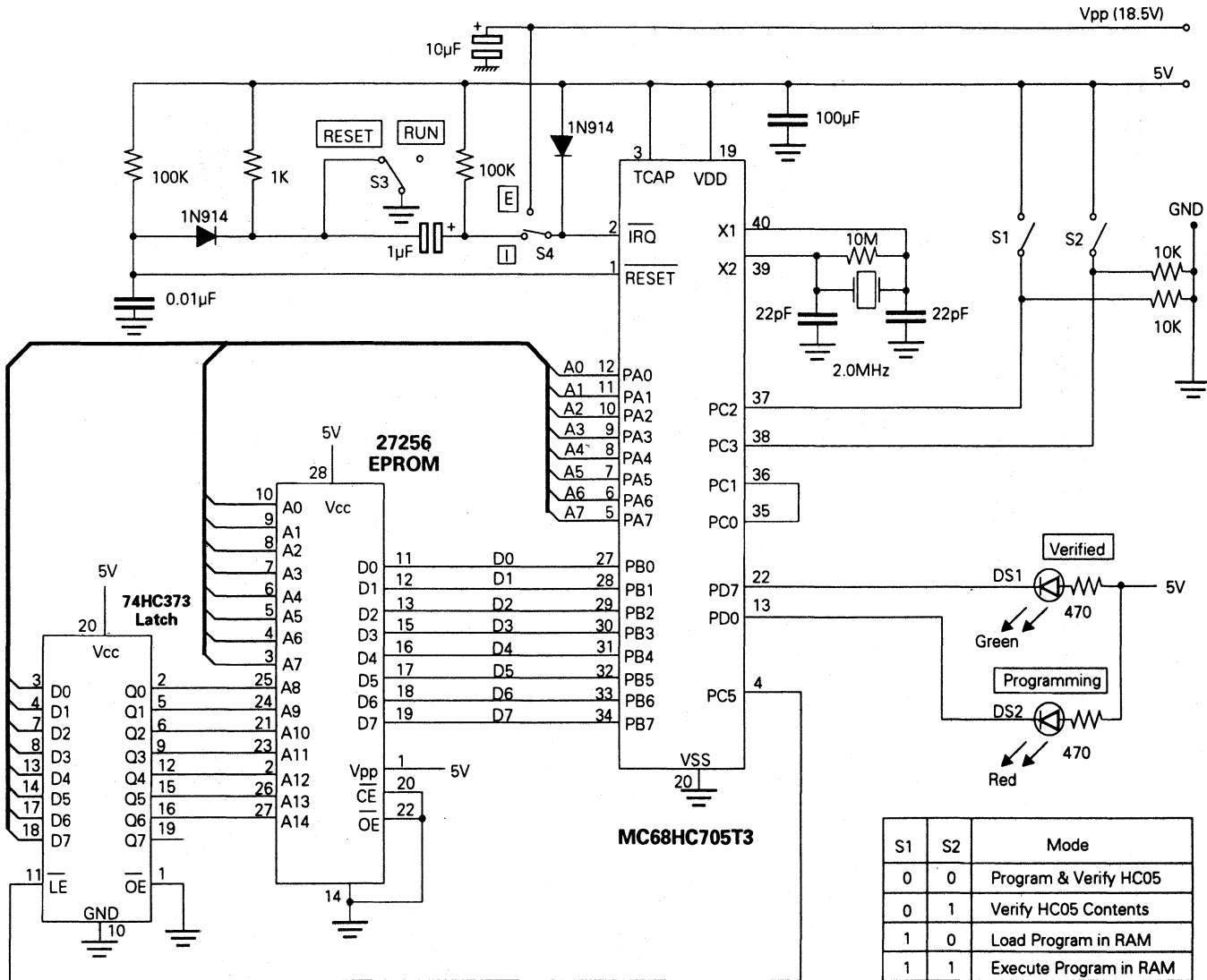
The bootloader code in the MC68HC705T3 has 4 modes of operation, selected by switches S1 and S2. In addition to EPROM programming and verifying it is also possible to load and execute a program in RAM locations \$0100-\$01FF. Like the EPROM programmer, the RAM loader transfers data from the corresponding addresses in an external EPROM; there is no serial load facility.

The progress of the loader can be observed in more detail if LEDs are connected to all the pins on port D, in a manner similar to that shown in figure 1 for bits 0 and 7. If this is done, bit 7 is still used for verification and should be a different colour. During EPROM programming the high order address byte is displayed. For RAM loading the 7 lowest addresses are shown. Address 7 is not displayed in order to avoid a false impression of a successful verify.

OPERATING PROCEDURE

1. The Vdd supply should be off, S3 closed (reset) and S4 in position I (internal).
2. Insert MC68HC705T3.
3. Switch on Vdd. If this is not done before step 4, the MC68HC705T3 may be damaged.
4. If EPROM programming is required, switch on Vpp by changing S4 to position E (external); it is assumed that an external Vpp (18.5V) is present. Vpp is not necessary for verification or for the RAM load and execute modes. These other modes do, however, require at least 9 Volts on pin 2 immediately after reset. The charge pump will supply this voltage when S3 is opened if S4 has been left in position I.
5. Select required mode using switches S1 and S2.
6. Open S3; this starts the selected mode. The red LED flashes to indicate that the bootloader is running. At the end of the programming modes a verification is carried out and the green LED switched on if this verification passes. If the green LED is not on at the end of the procedure the verification has failed. The red LED may be on or off.
7. Close S3, at this stage the bootloader can be run again by returning to step 5.
8. If S4 is in position E, return it to position I. This disconnects an externally supplied Vpp. If this is not done before step 9, the MC68HC705T3 may be damaged.
9. Switch off Vdd.
10. Remove MC68HC705T3.

Figure 1. MC68HC705T3 Bootloader Circuit



S1	S2	Mode
0	0	Program & Verify HC05
0	1	Verify HC05 Contents
1	0	Load Program in RAM
1	1	Execute Program in RAM

ADDITIONAL FEATURES

A handshake facility has been included to allow the external EPROM to be replaced by an intelligent data source and to provide a limited debug capability. An alternative source of data could, for example, be a microprocessor controlling a gang programmer. When the direct connection shown in figure 1 between bits 0 and 1 on port C is made, the handshake is performed automatically. When this connection is not made the boot-loader stops immediately before reading data from port B and outputs a high on bit 1. In order to proceed bit 0 should be pulsed high and low again. The high on bit 1 can be used to indicate that the MC68HC705T3 is ready to receive data.

If bit 0 is held low with a 10kohm resistor and a push-button switch connected between bit 0 and Vdd the boot-loader can be single-stepped. This feature is intended for use with the RAM loader.

To effectively use single-stepping an LED should be connected to every port D pin. If the boot-loader is started in the RAM loading mode with no link between bits 0 and 1 on port C, it will stop at the first address (\$0100). Pressing the button will cause the boot-loader to increment one address at a time (a debouncing circuit may be required to prevent it skipping addresses). The LEDs display addresses A0-A6. If switch S2 is closed once the boot-loader has started, the LEDs display the MC68HC05T3's RAM data instead of the address. With S2 closed the contents of the external EPROM are ignored. Although the loader does not permit the inspection of any locations other than \$0100-\$01FF, it is possible to read other locations using a program loaded into RAM. The data can be saved in unused locations between \$0100 and \$01FF and then read using the above procedure.

SOFTWARE

The boot-loader resides in the area occupied by the self-test program in the ROMed MC68HC05T3. This area extends from \$7F00 to \$7FEF with the last 16 bytes (\$7FE0-\$7FEF) intended for vectors. In this boot-loader only the reset vector is actually used.

The program starts at address \$7F00 and immediately checks the state of bits 2 and 3 on port C. If they are both high a jump to \$0100 is executed. This is to allow the running of a program previously loaded into RAM. If either I/O line is low, the software switches the OSD character EPROM into the memory map and initialises the ports. As the port B pins are always inputs, its data direction register does not need to be written to. The subroutine STXHIS is then used to initialise the address bus to the first EPROM location to be

used (\$0400). As the MC68HC705T3, in common with all M6805 microprocessors, has only an 8-bit index register, it is necessary to execute a program in RAM to allow the required flexibility in selecting the address for writing to, and reading from, the on-chip EPROM. For writing to EPROM this program consists simply of an extended STA instruction followed by a 2-byte address and an RTS instruction. This program is built by transferring the contents of the 6 bytes at TABLE into RAM. The 5th and 6th bytes are constants used for controlling EPROM write timing (actually 8 bytes are transferred but only the first six are relevant). This transfer is accomplished by the loop at MOVE. After this, a conditional branch depending on the level of bit 2 in port C is taken. If the line is high, the RAM loader is entered, but if it is low the EPROM boot-loader modes (program or verify according to the level of bit 3) are started.

The RAM loader transfer the contents of \$0100-\$01FF of the external EPROM into the same addresses in the MC68HC705T3 and then verifies them, switching on the LED (bit 7 of port D) and entering the STOP mode if the verify succeeds. If the verify fails, the program will hang at the first address to fail. If during this process bit 3 of port C becomes high, then the external data is ignored and the contents of internal RAM appear, inverted, on port D. The decision to display data rather than write it is made by two conditional branches, one in the main loop of the RAM loader and one in the subroutine HAND1. This subroutine also performs handshaking and reads the external EPROM. If bit 3 of port C is high at the end of a successful verify, a jump to address \$0100 is performed instead of a STOP.

The EPROM loader similarly loops round its required addresses (\$0400-\$0AFF, \$2000-\$7EFF and \$7FF0-\$7FFF), transferring data from external EPROM. For the EPROM, however, several milliseconds are required to write data. The boot-loader performs two loops, each incorporating a 2mS write time per byte. After this is completed (it takes 110 seconds) a verify loop is performed and the green LED on bit 7 of port D is switched on if the verify succeeds. Alternatively, if an error is detected, the verify loop will hang up at the address of the first byte which did not verify. An inspection of the address bus (port A) will determine the low-order address of this location. The high-order addresses can be seen between the output of the latch and the external EPROM. If all eight LEDs are fitted the high-order address will be displayed.

The last page of the listing consists of the subroutines STXHI which has two entry points, the subroutine NXTADR, TABLE and the boot-loader's vectors. NXTADR is responsible for incrementing the EPROM address and skipping the areas not required for EPROM programming.

MC68HC705T3 BOOTLOADER LISTING

0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016 0000
0017 0001
0018 0002
0019 0003
0020 0004
0021 0006
0022 0007
0023
0024 001c
0025
0026 003e
0027
0028
0029
0030
0031
0032
0033
0034
0035
0036
0037
0038
0039
0040
0041
0042
0043
0044
0045 0040
0046
0047 0040
0048 0041
0049 0043
0050 0044
0051 0045
0052
0053 7f00
0054

```

*****
*
*                               MC68HC705T3 Bootloader.
*
*
* This software was developed by Motorola Ltd. for demonstration purposes.
* No liability can be accepted for its use in any specific application.
* Original software copyright Motorola - all rights reserved.
*
*
*                               P. Topping                               14-Feb-91
*
*****

```

```

PORTA EQU $00      Port A address
PORTB EQU $01      Port B  "
PORTC EQU $02      Port C  "
PORTD EQU $03      Port D  "
DDRA  EQU $04      Port A data direction reg.
DDRC  EQU $06      Port C  " " "
DDRD  EQU $07      Port D  " " "
PROG  EQU $1C      EPROM program register
TR1   EQU $3E      Test register 1

```

```

*****
*                               Switch options.
*
*                               2,C 3,C
*
*                               0 0      Program and verify EPROM.
*                               0 1      Verify EPROM.
*                               1 0      Parallel RAM load/verify.
*                               1 1      Execute prog. in RAM.
*
* Note: ROM always starts at $2000, Vectors
*       are always at $7FFF. These are not
*       the normal addresses for T1 or T2.
*
*****

```

```

ORG $40

RAM RMB 1
ADDR RMB 2
RET RMB 1
LOOP RMB 1
TIME RMB 1

ORG $7F00

```

```

0055
0056
0057
0058
0059
0060
0061 7f00 05 02 06
0062 7f03 07 02 03
0063 7f06 cc 01 00
0064
0065 7f09 10 3e
0066 7f0b a6 ff
0067 7f0d b7 04
0068 7f0f b7 07
0069 7f11 b7 03
0070 7f13 a6 22
0071 7f15 b7 06
0072 7f17 4f
0073 7f18 ae 04
0074 7f1a cd 7f a7
0075 7f1d 58
0076 7f1e d6 7f d5
0077 7f21 e7 3f
0078 7f23 5a
0079 7f24 26 f8
0080 7f26 05 02 29
0081
0082
0083
0084
0085
0086
0087
0088 7f29 ad 20
0089 7f2b ad 5b
0090 7f2d 06 02 03
0091 7f30 d7 01 00
0092 7f33 3c 00
0093 7f35 5c
0094 7f36 26 f3
0095
0096 7f38 ad 11
0097 7f3a ad 4c
0098 7f3c d1 01 00
0099 7f3f 26 fe
0100 7f41 3c 00
0101 7f43 5c
0102 7f44 26 f4
0103
0104 7f46 06 02 bd
0105 7f49 20 38
0106
0107 7f4b ae 01
0108 7f4d 4f
0109 7f4e ad 57
0110 7f50 5f
0111 7f51 81
0112

.....
*
*      Execute program in RAM, port set-up, dump.
*
*
.....

START  BRCLR  2,PORTC,BOOT  CHECK FOR JMP TO RAM OR BOOT
        BRCLR  3,PORTC,BOOT
RSTRT  JMP    $0100        JUMP TO PROGRAM IN RAM

BOOT   BSET   0,TR1        SWITCH OSD CHARACTER EPROM INTO MEMORY MAP
        LDA   #$FF
        STA  DDRA        ALL OUTS, ADDRESSES
        STA  DDRD        ALL OUTS, LEDS
        STA  PORTD       LEDS OFF
        LDA  ##00100010  0,1: HANDSHAKE, 2 & 3: OPTION SWITCHES,
        STA  DDRC        4: TCAP, 5: LATCH, 6 & 7: NOT THERE
        CLRA
        LDX  #4          X <- 00000100
        JSR  STXHIS     ADHI <- $04, ADLO <- $00
        LSLX          X <- $08
MOVE   LDA   TABLE-1,X
        STA  RAM-1,X    $C7,$04,$00,$81,$02,$02
        DECX
        BNE  MOVE
        BRCLR 2,PORTC,PRGVER EPROM OR RAM

.....
*
*      Parallel RAM load/verify ($0100-$01FF).
*
*
.....

LDRAM  BSR    RMSTRT
PLOOP  BSR    HAND1    HANDSHAKE AND GET DATA
        BRSET 3,PORTC,NEXT SKIP RAM LOAD ?
        STA  $0100,X  NO
NEXT   INC    PORTA
        INCX
        BNE  PLOOP

        BSR    RMSTRT
PVERF  BSR    HAND1    HANDSHAKE AND GET DATA
        CMP  $0100,X
        BNE  *        HANG UP IF NOT OK
        INC  PORTA
        INCX
        BNE  PVERF

        BRSET 3,PORTC,RSTRT IF PUT HIGH DURING VERIFY
        BRA  FIN      THEN EXECUTE PROGRAM IN RAM.

RMSTRT LDX   #1
        CLRA        START @ $0100
        BSR  STXHIS
        CLRX
        RTS

```

```

0113
0114
0115
0116
0117
0118
0119
0120 7f52 06 02 22
0121
0122 7f55 ad 3c
0123 7f57 1a 1c
0124 7f59 bd 40
0125 7f5b 10 1c
0126 7f5d b6 45
0127 7f5f ae a6
0128 7f61 5a
0129 7f62 26 fd
0130 7f64 4a
0131 7f65 26 f8
0132 7f67 3f 1c
0133 7f69 ad 45
0134 7f6b 26 e8
0135
0136 7f6d ae 04
0137 7f6f bf 41
0138 7f71 ad 32
0139 7f73 3a 44
0140 7f75 26 de
0141
0142 7f77 3c 40
0143 7f79 ad 18
0144 7f7b bd 40
0145 7f7d 26 fe
0146 7f7f ad 2f
0147 7f81 26 f6
0148
0149 7f83 a6 7f
0150 7f85 b7 03
0151 7f87 8e
0152
0153
0154
0155
0156
0157
0158
0159 7f88 9f
0160 7f89 a4 7f
0161 7f8b 07 02 07
0162 7f8e d6 01 00
0163 7f91 20 02
0164
0165 7f93 b6 41
0166 7f95 43
0167 7f96 b7 03
0168 7f98 12 02
0169 7f9a 01 02 fd
0170 7f9d 13 02
0171 7f9f 00 02 fd
0172
0173 7fa2 b6 01
0174 7fa4 81
0175

```

```

*****
*
*   Program & verify EPROM.
*   $0400-$0AFF, $2000-$7EFF & $7FF0-$7FFF.
*
*****

PRGVER  BRSET   3,PORTC,VERF   DO A VERIFY ONLY ?

PRGLOP  BSR     HAND2          HANDSHAKE AND GET DATA
        BSET   5,PROG         LATCH ADDRESS & DATA
        JSR    RAM            WRITE ONE BYTE
        BSET   0,PROG         APPLY VPP
        LDA    TIME           GET PROGRAMMING TIME IN ms
DELNMS  LDX     #SA6           2ms- INNER LOOP (2MHz XTAL)
MS1     DECX
        BNE    MS1
        DECA          x A OUTER LOOP
        BNE    DELNMS
        CLR     PROG         REMOVE VPP
        BSR    NXTADR        NEXT ADDRESS
        BNE    PRGLOP        DONE ?

        LDX     #4           GET INITIAL MS ADDR
        STX    ADDR          ADDRHI <- $0400
        BSR    STXHI         A8-A14 <- $04
        DEC    LOOP
        BNE    PRGLOP        DO PROG LOOPS TWICE

VERF    INC     RAM           CHANGE STA TO EOR
CHECK   BSR     HAND2        HANDSHAKE AND GET DATA
        JSR    RAM           COMPARE WITH EPROM BYTE
HANG1   BNE    HANG1        HANG UP IF DIFFERENT
        BSR    NXTADR        NEXT
        BNE    CHECK

FIN     LDA     #$7F
        STA    PORTD        VERIFY LED ON, REST OFF
        STOP

*****
*
*   Handshake and Read external EPROM.
*
*****

HAND1   TXA
        AND    #$7F          ADDRLO (RAM)
        BRCLR  3,PORTC,DISP  MAKE SURE VERIFY LED IS OFF
        LDA    $0100,X       DISPLAY RAM DATA OR ADDRESS ?
        BRA    DISP          DATA

HAND2   LDA    ADDR          ADDRHI (EPROM)
DISP    COMA
        STA    PORTD        DISPLAY ADDRESS (OR DATA)
        BSET   1,PORTC       HANDSHAKE HIGH
        BRCLR  0,PORTC,*
        BCLR   1,PORTC       AND LOW AGAIN
        BRSET  0,PORTC,*

        LDA    PORTB        READ AN EXTERNAL BYTE
        RTS

```

```

0176
0177
0178
0179
0180
0181
0182 7fa5 b6 00      STXHI  LDA    PORTA      ADDRLO, LOAD ORIG. CONTENTS
0183 7fa7 1a 02      STXHIS BSET   5,PORTC    UPDATE LATCH
0184 7fa9 bf 00              STX    PORTA      ADDRHI
0185 7fab 1b 02              BCLR   5,PORTC    LATCH CONTENTS OF ADDRHI
0186 7fad b7 00              STA    PORTA      RESTORE CONTENTS OF ADDRLO
0187 7faf 81              RTS
0188
0189 7fb0 3c 42      NXTADR  INC    ADDR+1    INC. ADDRLO
0190 7fb2 3c 00              INC    PORTA
0191 7fb4 26 13              BNE   GOBACK      RETURN IF NOT PAGE BOUNDARY
0192 7fb6 be 41              LDX   ADDR        GET ADDRHI
0193 7fb8 5c              INCX  ADDR        INC. ADDRHI
0194 7fb9 a3 80              CPX   #$80
0195 7fbb 27 0c              BEQ   GOBACK      END OF VECTORS ? IF SO, EXIT WITH Z=1
0196 7fbd a3 7f              CPX   #$7F
0197 7fbf 26 09              BNE   NOTEND      END OF MAIN BLOCK ?
0198 7fc1 a6 f0              LDA   #$F0        MOVE TO USER VECTORS
0199 7fc3 b7 42              STA   ADDR+1     UPDATE ADDRLO
0200 7fc5 bf 41              STX   ADDR        UPDATE ADDRHI
0201 7fc7 ad de              BSR   STXHIS     UPDATE LATCH (ADDRHI)
0202 7fc9 81              GOBACK RTS        Z=1 IF FINISHED
0203
0204 7fca a3 0b      NOTEND  CPX   #$0B    WAS THAT END OSD EPROM ?
0205 7fcc 26 02              BNE   GO
0206 7fce ae 20              LDX   #$20        MOVE TO USER EPROM BEGINNING
0207 7fd0 bf 41              STX   ADDR        UPDATE ADDRHI
0208 7fd2 ad d1              BSR   STXHI      UPDATE EXTERNAL LATCH OF ADDRHI
0209 7fd4 43              COMA  ADDR        CLEAR Z FLAG
0210 7fd5 81              RTS
0211
0212 7fd6 c7      TABLE  FCB   $C7    STA EXTENDED INSTRUCTION
0213 7fd7 04 00      FDB   $0400        START ADDRESS
0214 7fd9 81              FCB   $81          RTS INSTRUCTION
0215 7fda 02              FCB   2            2 PROGRAMMING LOOPS
0216 7fdb 02              FCB   2            2 ms PROGRAMMING TIME (PER LOOP)
0217
0218
0219
0220
0221
0222
0223
0224 7fee              ORG    $7FEE
0225
0226 7fee 7f 00      RESET  FDB   START   RESET VECTOR
0227
0228              END

```


Additional Information

[Faint, illegible text]

Additional Information

Additional information relevant to 8-bit MCU applications may be found in the following Motorola documents, available through your Franchised Distributor by quoting the appropriate reference.

BR266/D	M68HC11EVM Evaluation Module (Rev. 3)
BR278/D	M68HC11EVB Evaluation Board (Rev. 2)
BR285/D	M68701EVM Evaluation Module
BR291/D	M68705EVM Evaluation Module
BR295/D	M68HC05EVM Evaluation Module (Rev. 2)
BR411/D	The M68HC11 Microcontroller Family
BR433/D	M68HC05 8-bit Microcontrollers. The Home of the Industry Standard Microcontroller (Rev. 2)
BR459/D	MC68HC05SC24 Secure 8-bit Microcomputer with EEPROM: Product Preview
BR468/D	Secure MCU Product Packaging
BR568/D	MCU Firmware (Rev. 1)
BR706/D	M68HC11F1EVM Evaluation Module
BR730/D	M68HC05PGMR Programmer Board
BR735/D	M68HC05P8EVS CSIC Evaluation System
BR736/D	M68HC11EVBU Universal Evaluation Board
BR748/D	M68HC711D3PGMR Programmer Board
BR764/D	M68HC05 CSIC Portfolio
BR909/D	The Military Microprocessor Fleet is Arriving
BR911/D	Military Microprocessor Fact Sheet (Rev. 4, 1992)
BR913/D	The Military 68HC11A0 and 68HC11A1 are Available Now
BR922/D	Military MCU – 68HC811E2
BR1111/D	M68HC705J2/P9PGMR Programmer Board
BR1113/D	M68HC705B5PGMR Programmer Board
BR1116/D	Advanced Microcontroller Unit (AMCU) Literature
BR1310/D	Our Low-Cost 68HC05 CSICs Can Take Your Designs to New Heights
BRE435/D	M1468705EVM Evaluation Module (replaces BRE294/D)
BRE447/D	M6805SC13 Product Preview
BRE448/D	M68HC05SC1121 Product Preview
BR452/D	Motorola Development Support Guide (Rev. 2, 1991)
DL411/D	Communications Applications Manual
DLE404/D	M6804 MCU Manual (1984)
HC711D3EVB/AD1	M68HC711D3EVB Evaluation Board User's Manual
HC711D3PGMR/AD1	M68HC11711D3PGMR Programmer Board User's Manual
M68HC05AG/AD	M68HC05 Applications Guide
M68HC05PGMR/AD1	M68HC05PGMR Programmer Board User's Manual
M68HC11RM/AD	M68HC11 Reference Manual (Rev. 3, 1991)
M68PCBUG11/D1/D	M68HC11 PCbug11 User's Manual
M6805UM/AD3	M6805 HMOS / M146805 CMOS Family User's Manual (1991)
M6809PM/AD	MC6809-MC6809E Microprocessor Programming Manual (1981)
MC68HC05CxRG/AD	MC68HC05Cx HCMOS Single-Chip Microcontrollers Programming Reference Guide (Rev. 1)
MC68HC11A8RG/AD	MC68HC11A8 Programming Reference Guide (Rev. 1)

Additional Information (continued)

MC68HC11D3RG/AD	MC68HC11D3/MC68HC711D3 Programming Reference Guide
MC68HC11E9RG/AD	MC68HC11E9 Programming Reference Guide
MC68HC11F1RG/AD	MC68HC11F1 Programming Reference Guide
MC68HC11L6RG/AD	MC68HCL6/MC68HC711L6 Programming Reference Guide
MC68HC811E2RG/D	MC68HC811E2 Programming Reference Guide
MC6801RM/AD2	MC6801 8-bit Single-Chip Microcomputer Reference Manual
MC6840UM/AD1	MC6840 Programmable Timer Fundamentals and Applications
SG96/D	Linear/Interface Integrated Circuits Selector Guide & Cross Reference (Rev. 5, 1992)
SG138/D	Military IC & Discrete Selector Guide (Rev. 2, 1992)
SG165/D	CSIC Microcontrollers Update – Quarter 2, 1992
SG166/D	Advanced Microcontroller Division Update – Quarter 4, 1991
TB301/D	Basic Microprocessors and the 6800 (Bishop, 1979)
TB302/D	What Every Engineer Should Know About Microcomputers (Bennett, Evert and Lander, Rev.1, 1991)
TB303/D	Using Microprocessors and Microcomputers: The Motorola Family (Greenfield and Wray, Rev. 1, 1988)
TB309/D	Programming the 6809 (Zaks & Labiak, 1982)
TB316/D	Single- & Multi-Chip MCU Interfacing (Lipovski, 1988)
TOOLWARE/D	Software Development Tools for MS-DOS

Literature Distribution Centers:

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.

JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.

ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong.



MOTOROLA

JIT PRINTED IN THE USA 1993 MPS

DL408/D

