Ethernet Network Interface Adapter for the Apple Macintosh II NuBus

National Semiconductor
Application Note 686
Andrew C. Pagnon
June 1990

INTRODUCTION

The gradual move in recent years towards distributed processing units with a need for these to communicate and the increasing demand for peripheral usage optimization has resulted in the fast growth of Local Area Networks. In an attempt to standardize communications between networks the International Standards Organization (ISO) has proposed a seven layer reference model called the Open System Interconnect (OSI) which provides an independent framework for all the emerging network standards. The IEEE has defined a number of these standards (802.3 to 802.6) covering the two lower layer functions, physical and data link layers.

National Semiconductor provides a three chip set which supports the Ethernet/Thin-wire Ethernet standard (a subset of IEEE802.3) the DP8390 Network Interface Controller (NIC), DP83910 Serial Network Interface (SNI), and DP8392 Coaxial Transceiver Interface (CTI).

The aim of this application note is to describe the implementation of a Macintosh II, IIx and IIcx to Ethernet/Thin-wire Ethernet interface solution using the NSC chip set. This solution takes the form of a network interface adapter card which on one side plugs into any of the six Macintosh II NuBus expansion slots and on the other supports two physical layer options, Ethernet and thin-wire Ethernet.

The board easily interfaces to the Macintosh II NuBus interface with few external components. This application note assumes the reader is familiar with NSC's Ethernet chip set and the Macintosh II NuBus.

The note begins with a hardware overview of the adapter card, and a background description of the NuBus interface. This is followed by a detailed description of hardware supported by the main sequencer/arbitrator state diagram. This covers arbitration and a detailed description of all the cycle types implemented on the card. The PAL equations and part list are included at the end of the note along with a detailed schematic and timing diagrams.

HARDWARE OVERVIEW

The main function of this adapter card is to transfer Ethernet packet data to/from the Macintosh CPU via NuBus during LAN transmissions and receptions. The card supports a NuBus interface to the CPU and an Ethernet interface to the network. Data transfers between the interfaces are routed on the card's local bus through 8k words of shared buffer memory which temporarily stores ethernet packet data, thus decoupling data transfers across the two interfaces. The 8k buffer memory can be expanded to 32k by simply replacing the memory ICs.

Figure 1 shows a simplified block diagram of the adapter. Besides the basic DP8390 chip set this diagram illustrates the connection of the slot and cycle decode logic used to select the card, and generate read/write cycles. The arbiter controls whether the NIC or NuBus can access the buffer RAM. The RAM contains the transmit/received packet data, and the ROMs (actually one chip) contain the Ethernet Address and the Macintosh configuration information. The address bus interface latches store the NuBus address from the multiplexed address/data bus, and the data bus interface consists of buffers and latches to assemble the 16-bit RAM buffer data into a 32-bit word for the NuBus.

Transmission/Reception

For Ethernet transmissions the host CPU writes data into the transmit area of the adapter card buffer memory over the NuBus interface. The host CPU then sets up the NIC to transmit the data by writing to its internal registers. The NIC responds by fetching the data into its internal FIFO using its local DMA channel, from where it is sent to the SNI-CTI and onto the Ethernet cable. Once the data has been transmitted the NIC issues an interrupt back to the host CPU and sets a status bit in its internal register.

For Ethernet receptions data is loaded from the Ethernet cable into the internal FIFO of the NIC from the SNI and CTI. When a programmable threshold is reached in the FIFO, the NIC transfers the data into the receive area of the adapter card buffer memory using its local DMA channel. Once a complete packet has been loaded into memory, the NIC sets up a pointer in its internal register, issues an interrupt to the host CPU and sets a status bit in its internal register. The host processor responds by reading the packet from the adapter card memory over the NuBus interface and updating the packet pointers stored in the NICs internal register.

General Adapter Architecture Considerations

A shared memory architecture has been chosen for this design to maximize data throughput while not adding any extra cost or intelligence on the card. The buffer memory is mapped into the NuBus address space and a NuBus slave interface plus local bus arbitration logic is implemented on the adapter card. The reasoning for this decision is given below.

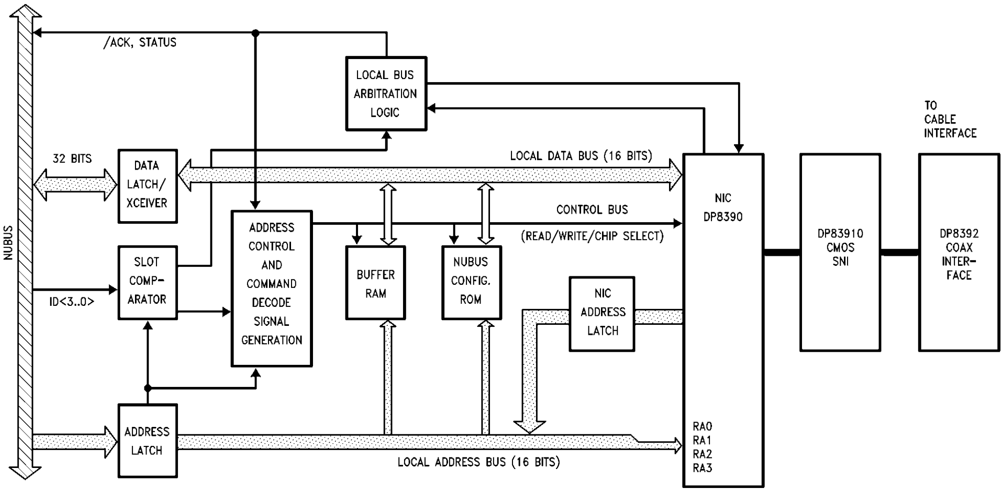
The DP8390 efficiently supports an input/output port architecture, in which the adapter card makes use of the NICs Remote DMA facility to transfer network data between the buffer RAM and an input/output port interfacing to the NuBus and to the host CPU. This implementation is a slightly less expensive option than others however the throughput of the port interface is somewhat limiting, and there are no memory addressing limitations on the NuBus that would require and I/O port technique.

A bus master architecture, in which the adapter card can gain ownership of the host CPU bus and transfer data directly into system memory is significantly more costly and with the current generation of controllers will not yield significantly better performance across NuBus without going to the expense of adding an on-card processor.

Thus, using a buffer RAM that is addressed directly by the NuBus and the Ethernet Controller, provides the flexibility of reading/writing data via the NuBus at fast speeds, and since the DP8390's local DMA only utilizes a small percentage of the RAM's total access time (12%) the RAM is mostly free for NuBus activity.

PAL® is a registered trademark of and used under license from Advanced Micro Devices, Inc.

Ethernet Network Interface Adapter for the Apple Macintosh II NuBus AN-686



TL/F/10805–1

FIGURE 1. General Block Diagram for the NuBus Shared RAM Adapter

Once the Shared Adapter RAM approach is chosen another architectural consideration is the width of the Buffer RAM and of the CPU bus transfers supported. Table I shows the options considered. The 16-bit Buffer memory and 32-bit NuBus transfer option was chosen as the best compromise on data throughput and component count/cost.

General Hardware Overview

Figure 2 shows a more detailed block diagram of the Adapter. This shows each block, and details the chips used to implement each block.

The Ethernet to buffer memory interface is implemented using National Semiconductor chip set. The DP8390 Network Interface Controller is a CMOS VLSI device designed to ease interfacing with IEEE 802.3 Ethernet type Local Area Networks. It implements all Media Access Control MAC functions (a subset of the ISO data link layer) for transmission and reception of packets in accordance with the IEEE 802.3 standard. Its dual DMA channels and internal FIFO provide a simple yet efficient packet management design. All bus arbitration and memory support logic required by its dual DMA channels are integrated into the NIC.

The DP83910 Serial Network Interface is a CMOS combined analog and digital device which provides the Manchester encoding and decoding functions of IEEE 802.3 Ethernet type Local Area Networks. It contains ECL like balanced drivers and receivers, collision signal translator and a diagnostic loopback circuit.

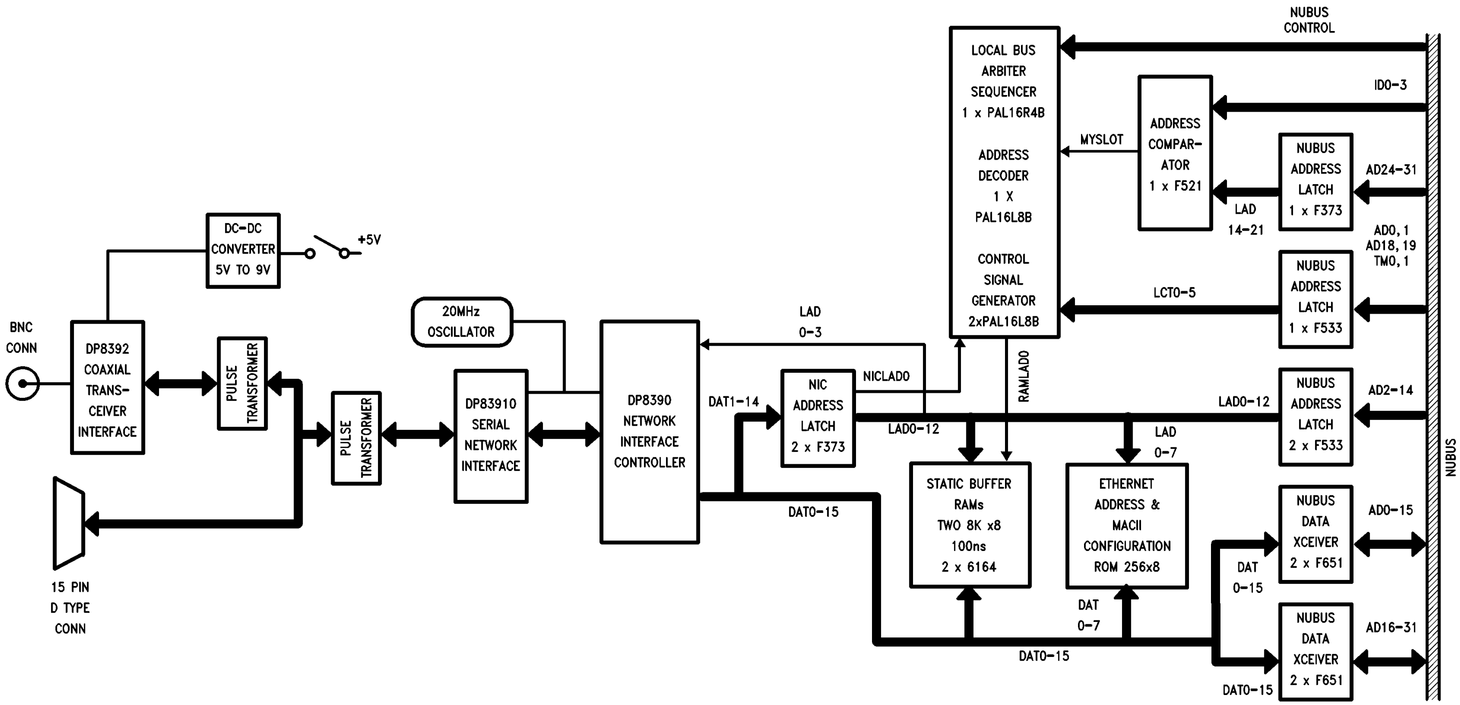
The DP8392 Coaxial Transceiver Interface is a bipolar device used as a coaxial cable line driver/receiver for IEEE 802.3 Ethernet Local Area Networks. In Ethernet applications the transceiver is usually mounted within a dedicated enclosure (Media Access Unit) and connected to the SNI via a drop cable, while for Thin-wire Ethernet (low cost version of Ethernet) the CTI is mounted on the same board as the SNI. Signal and power isolation requirements are met by placing a set of pulse transformers between the SNI and the CTI, and using a DC to DC converter to provide the CTI's −9V supply.

The adapter card supports a 32-bit NuBus interface to the host CPU, implemented using synchronous sequencer logic in the registered PAL 16R4. This interface supports transfers to the Network Interface Controller registers, the "Ethernet address/Mac configuration" ROM and the buffer memory. The two card interfaces must request use of the local bus before they can initiate a transfer to any of the on card devices. These requests are processed by arbitration logic which gives priority to the Ethernet interface.

The 256 x 8 ROM (LS471) contains the unique Physical Address assigned to each Etherent board and the Configuration data required on each NuBus board which supplies identifying information about the board. This ROM can also contain device driver data.

The address/data interface to NuBus consists of four F651s data transceivers to transfer 32-bit data from/onto the NuBus, and three F533s and an F373 used to latch the address and the transfer mode signals onto the adapter card. Data on NuBus is inverted and byte swapped, so inverting transceivers and latches are used. An exception to this is the F373 which is a non-inverting version of the F533 used to latch AD24–31 which are then compared with the ID lines of the particular NuBus slot. Also the card performs a hardware byte swap on the NuBus data.

On NuBus transfers, an F521 8-bit comparator aids in the address decode function by determining whether or not the transfer is intended for the adapter card.



TL/F/10805–2

FIGURE 2. Macintosh II Ethernet Adapter Card

TABLE I. Adapter Card NuBus RAM Width Options

NuBus Transfer Width	Buffer Memory Width	NuBus Clock Beats	NuBus Maximum Data Rate	Percent Bus Usage	Extra Devices Required
16 Bits	16 Bits	4	32 Mbits/sec	31%	None
32 Bits	16 Bits	5	53 Mbits/sec	19%	2-Transceivers 1-PAL16L8
32 Bits	32 Bits	4	64 Mbits/sec	15.5%	2-Transceivers 2-RAM 8k x 8
32 Bits	32 Bits	3	80 Mbits/sec	12.5%	2-Transceivers 2-RAM 8k x 8 (80 ns)

TABLE II. Adapter Card Cycle Type Decoding

AD0	AD1	TM0	TM1	AD18	AD19	Cycle Type
X	X	X	L	H	L	RAM Read
X	L	L	H	H	L	RAM Write Byte 0, 1
L	L	H	H	H	L	RAM Write (Byte 0)
H	L	H	H	H	L	RAM Write (Byte 1)
X	H	X	H	H	L	No Action
X	X	X	L	H	H	ROM Read
X	X	X	H	H	H	Bus Error (ROM Write)
X	X	X	X	L	L	Bus Error (No Device)
X	X	H	X	L	H	NIC Write
X	X	L	X	L	H	NIC Read

NUBUS BACKGROUND

This section describes the NuBus implemented in the Macintosh II expansion slots, and the Ethernet adapter card implementation of its interface. It covers NuBus' main features and signals as used by the card, followed by a description of the address space and addressing modes and ending with a description of the NuBus interface protocol.

NuBus is the bus chosen by Apple to drive the expansion slots of the Macintosh II. Its main features are:

- 32-bit wide multiplexed address data lines
- Synchronous 10 MHz clock cycle (75% duty cycle)
- Read and Write cycles (Mac II does not support block transfers)
- I/O and interrupts are memory mapped
- Geographical addressing lines

Each slot has its own geographical addressing lines onto the adapter board. This is illustrated in *Figure 3* by the super slot space in which each card has its own 256 Mbytes of memory space. Therefore no board configuration is required. (Described later.)

The Ethernet adapter card only implements a NuBus slave interface and therefore arbitration logic to gain bus mastership has not been implemented. Note also that no parity generating/checking logic has been implemented either.

The Ethernet adapter board uses the following NuBus signals:

- Clock
- Reset
- Card Slot Identification
- Non-Master Request
- Address/Data Signals/AD<31..0>
- Control Signals /TM<1..0>, /START, /ACK(nowledge)

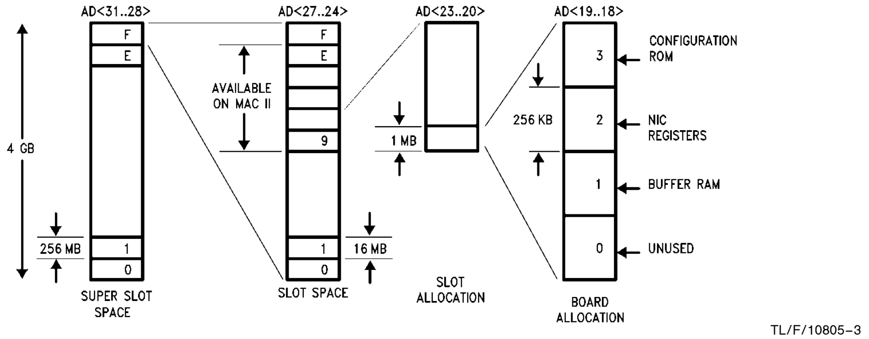
The evaluation board supports single word transfers to the NIC registers, Address/Configuration ROM and the Buffer RAM. During the Start clock of a NuBus cycle the NuBus address and transfer mode lines are decoded on the card as shown in *Figure 3,* and as follows:

- TM<1..0> determine the type of transfer (read/write),
- AD<24..31> determine which NuBus slot is accessed.
- AD<19..18> determine which adapter card device is accessed (NIC, ROM or RAM)
- AD<15..2> are used to access a particular location within the device.
- AD<0..1>&<20..23> are ignored by the address decode

The adapter card responds to all NuBus cycles which address the card by generating an acknowledge signal ACK for one clock period and driving a status code on the transfer mode lines. Only two of the four NuBus defined transfer modes are supported by the card, transfer complete or bus error (see Table III).

TABLE III. NuBus Status Codes

TMO	TM1	Acknowledge
H	H	Transfer Complete
L	H	Bus Error



TL/F/10805–3

FIGURE 3. NuBus and Adapter Card Address Space Mapping

If the CPU requests a transfer to the adapter card NuBus slot which does not address a device on the card, or requests a write transfer to ROM, the board will respond with an error status encoding of the TM<1..0> lines during the ACK clock of the NuBus cycle. Otherwise a "Bus Transfer Complete" code is returned.

Typical NuBus read and write cycle timings are shown in *Figures 4* and *5.* The first NuBus cycle asserts the START line going low and presenting the address and the transfer mode. A number of clock cycles may follow before the last cycle presents the data and status on the bus, and asserting the ACK signal.

The adapter card does not implement the two other status codes supported by NuBus, "bus time out error", and "try again later". The design of the adapter card ensures that all NuBus cycles will be acknowledged within the NuBus time-out period.

NuBus Address Space

With a 32-bit architecture, the NuBus provides a 4 Gigabytes of address space, *Figure 3.* The 4 Gigabyte space is divided into sixteen 256 Megabyte Super Slots. The Super Slot being accessed is determined by decoding AD<31..28>. The top Super Slot is divided into sixteen Slot spaces by 16 Megabytes each determined by decoding AD<27..24>. Six of these slots ($9 to $E) are implemented as NuBus expansion board connectors on the MAC II. The interface adapter board may be plugged into any of these connectors. No hardware configuration on the adapter card is required.

24/32 Bit Addressing Modes

The adapter card, by ignoring address bits AD<23..20>, supports both 32- and 24-bit addressing modes.

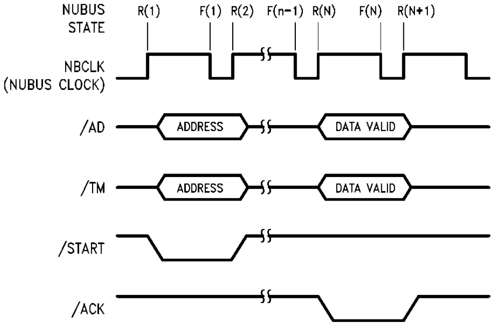
When addressing the card in 24-bit mode, addresses of the form "$sx xxxx" where s is the slot number can be used. The Mac II hardware translates this into a 32-bit address of the form "$Fs0x xxxx".

When addressing the card in 32-bit mode addresses of the form "$Fsxx xxxx" can be used. Note that as the adapter card ignores address bits AD<23..20>, addresses of the form "$Fssx xxxx" access the same adapter card location in both 32- and 24-bit modes, and as Apple have indicated that to ensure compatibility with future versions of the Macintosh designers should not rely on 24-bit mode addressing, it is suggested that addresses of the form "$Fssx xxxx" are always used.

Although supporting 24-bit mode addressing limits the memory range of each slot from 16M to 1M, this sufficiently covers the need of an Ethernet adapter card and simplifies software development. The Macintosh slot manager puts the system in 24-bit addressing mode by default and the memory manager plus some toolbox routines do not currently function properly in 32-bit mode.

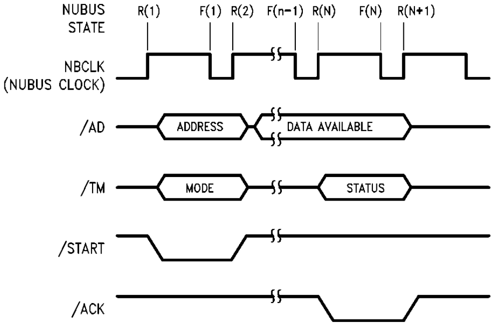
Adapter Card Address Space

Once the slot is selected, the Network Interface Adapter's memory space is subdivided into four 256 kbyte blocks of memory determined by decoding AD<19..18>.



TL/F/10805–4

FIGURE 4. NuBus Read Cycle



TL/F/10805–5

FIGURE 5. NuBus Write Cycle

Figure 3 shows the sub-division of the Super Slot, Slot, and Adapter Board address space.

It is important to note the following points when developing high level software to address the adapter card. If the NIC registers or the ROM are accessed as 8-bit devices, (by declaring a pointer to a character in Macintosh Programmers Workshop (MPW) for example) or as 16-bit devices (by declaring a pointer to a short integer), incrementing these pointers will usually only modify the transfer mode (by incrementing AD<1..0>) rather than increment the address to the device (NIC or RAM). It is therefore recommended that if this form of addressing is used, the RAM, ROM and NIC are declared as 32-bit devices (by accessing them with a pointer to an integer). This will ensure only word transfers take place and each NuBus address increment will also increment the address to the onboard device.

NuBus Timing Diagrams

The NuBus clock has a 100 ns period (10 MHz) with a 75% duty cycle (75 ns "high" and 25 ns "low"). NuBus signals are driven at the rising edge and sampled at the falling edge of the clock. A transfer (read/write cycle) is initiated when the master asserts /START, drives the address on AD<31..0>, and drives the transfer mode signals TM<1..0> with the appropriate code to indicate the desired transfer. A transfer is completed when the slave responds by asserting ACK and driving the transfer mode signals with the appropriate status code. Please refer to *Figure 4* and *Figure 5* for the NuBus read/write cycles.

For a read operation, once the master has acquired the bus, a read bus transaction involves the following steps:

R(1): The bus master asserts/START and the appropriate /ADx and /TMx lines to initate the transfer.

F(1): The bus slaves sample the /ADx and /TMx lines.

R(2): The bus master releases the /ADx, /TMx, and /START lines and waits for /ACK.

R(N): The bus slave places the requested data onto the /ADx lines, asserts /ACK, and places the appropriate status code on /TM0 and TM1 lines. (Note N may be from 2 to 256)

F(N): The bus master samples the /ADx, /ACK, and TMx lines to receive the data and note and error condition.

R(N+1): The bus slave releases the /ADx and /ACK lines and the /TMx lines. This may be the R(1) transition of the next transaction.

For a write operation, once the master has acquired the bus, a write bus transaction involves the following steps:

R(1): The bus master assert/START and the appropriate /ADx and /TMx lines to initiate the transfer.

F(1): The bus slaves sample the /ADx and /TMx lines.

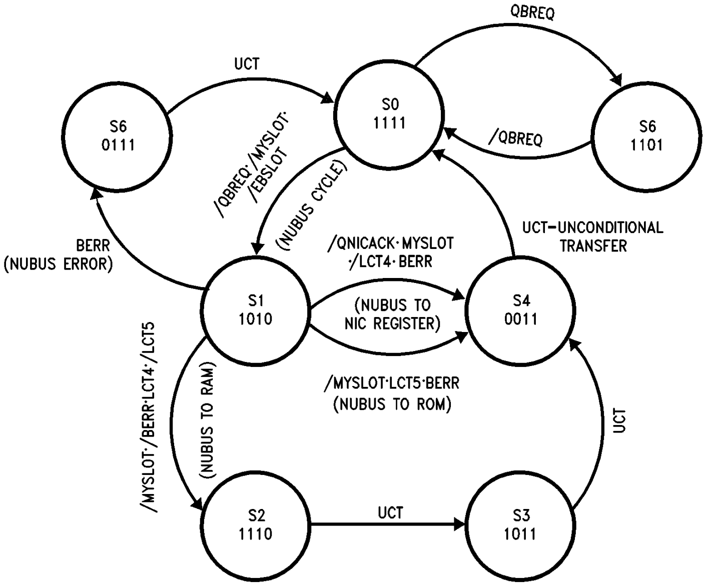
R(2): The bus master places the data on the /ADx lines, releases /TMx, and /START lines and waits for /ACK.

F(2)–F(N): The bus slave samples the /ADx lines to capture the data. The data may be sampled before or during the assertion of /ACK.

R(N): The bus slave asserts /ACK, and places the appropriate status code on /TM0 and TM1 lines when the data is accepted. (Note N may be from 2 to 256)

F(N): The bus master samples the /ACK and TMx lines to determine the end of a transaction.

R(N+1): The bus master releases the /ADx while the bus slave releases the /ACK lines and the /TMx lines.



TL/F/10805–6

FIGURE 6. State Diagram Sequencer

TABLE IV. Adapter Card Cycle State Sequence for Various Cycle Types and Corresponding Diagrams

Cycle Type	State Sequence	Functional Diagram	Timing Diagram
NuBus to RAM Read	S0 → S1 → S2 → S3 → S4 → S0	*Figure 12*	*Figure A-4*
NuBus to RAM Write	S0 → S1 → S2 → S3 → S4 → S0	*Figure 12*	*Figure A-5*
NuBus to ROM Read	S0 → S1 → S4 → S0	*Figure 8*	*Figure A-1*
NuBus to NIC Read	S0 → S1 → S4 → S0	*Figure 10*	*Figure A-2*
NuBus to NIC Write	S0 → S1 → S4 → S0	*Figure 10*	*Figure A-3*
NuBus Bus Error	S0 → S1 → S5 → S0	*Figure 8*	
NIC to RAM Read/Write	S0 → S6 → S0	*Figure 14*	

DETAILED HARDWARE DESCRIPTION

The card's main function is to transfer Ethernet packet data from the NuBus interface to the Ethernet cable during packet transmission, and from the Ethernet cable to the NuBus interface during packet reception, via an 8k x 16 Buffer RAM, expandable to 32k x 16. In addition to this the NuBus interface is allowed direct read and write access to the NICs registers to control and monitor the NIC's operations, and read access to the ‘‘Ethernet address/Mac configuration’’ ROM.

This transfer of packet data from Ethernet to host CPU is executed in two distinct stages, transfers between host CPU and buffer Memory, and transfers between buffer memory and Ethernet. The former is performed by the on-card NuBus slave interface whereas the latter is performed by the NIC chip set.

A synchronous sequencer/arbiter implemented in a PAL16R4 running on the 10 MHz NuBus clock controls all transfers supported by the adapter card. The state diagram for its operation is shown in *Figure 6.* States S1 to S5 support the NuBus slave interface, and state S6 supports the NICs interface. State S0 is the idle state. The sequence of states for each bus cycle type is shown in Table IV.

Arbitration for Local Card Bus by NIC/NuBus

All addressable devices on the card (Buffer RAM, NIC registers and ROM) share the common non-multiplexed local address and data buses. The two potential masters of this bus, the NuBus interface and the NIC, request access to the bus. Arbitration logic and the state sequencer resolve these requests. The sequencer only responds to master's requests during the idle state (S0). Therefore cycles already in progress are always allowed to complete before the bus is reallocated. Cycles always complete by returning to the idle state (S0). This prevents bus contention on the common local address bus at switchover time (NuBus/NIC) and enables arbitration to take place after every NuBus cycle or NIC local DMA burst. The NIC is given priority over the NuBus interface, so that if a NIC and a NuBus interface request are active when the sequencer is idle, the NIC cycle will be serviced first. The following bus latency discussion shows there is no real need to give one master priority over the other.

Bus Latency Requirements

This is defined as the time between a master issuing a request and receiving an acknowledge.

The NuBus interface bus latency allowable is determined by its 25.6 μs bus timeout period. The maximum NuBus bus latency that can be expected on this card, that is, the longest Bus Request from the NIC, occurs if a packet ends just as the NIC performs its last FIFO burst. The local DMA burst plus the End of Packet processing operations add up to just over 4 μs, well within the allowable 25.6 μs.

The NIC bus latency allowable is determined by the need to prevent its internal FIFO from overflowing during packet reception. The worst case bus latency the NIC can accommodate running on a 20 MHz clock is a little more than 1 μs (refer to the DP8390 Datasheet addendum). The maximum NIC bus latency that can be expected on this card, that is, the longest NuBus cycle to the card, is 0.5 μs (five NuBus Clocks), well within the allowable 1 μs.

Again, Table IV shows all the cycles supported by the card, and the state sequence followed by each one. The figures quoted display the timing diagram for each case.

NUBUS MASTER CYCLES

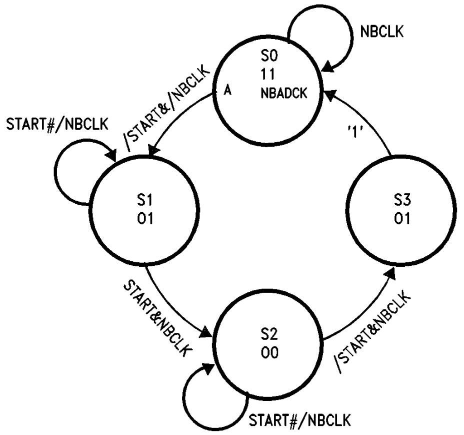
The CPU initiates NuBus cycles by driving the address and transfer mode onto NuBus and asserting the Start signal during the first clock of the bus access.

The adapter card latches the address onto the local bus on the falling edge of the first clock. A transparent latch is used so that the address is available on the local bus as soon as it is driven onto NuBus and address recognition can begin. A small asynchronous state machine in a control PAL® generates the address latching signal, which opens the latch when START becomes active and latches the address on the next falling edge of the clock. *Figure 7* shows the state machine diagram.

The address is enabled onto the local bus by NBADOE provided that the NIC is not already the bus master nor has issued a request while the sequencer is in the idle state.

The top 8 bits of the address are compared with the slot ID driven from NuBus.

The card, in response to the start signal, generates an enable signal (EBSLOT) which allows the sequencer to proceed onto the next NuBus cycle state (S1), provided the NIC bus request is not active and the above address comparison is successful (MYSLOT is active). This enable signal is cleared at the end of the NuBus cycle when ACK becomes active and prevents addresses not generated with a START signal from triggering the sequencer onto the S1 state. *Figure A-4* shows a detailed timing diagram of the address recognition operation. Note that FAST devices have been selected for the address latches and comparator to enable address recognition to meet the set up time of the sequencer.



TL/F/10805–7

A = State Variable (Not Used)

NBADCK = NuBus Address Clock

FIGURE 7. State Diagram for Clock to Latch NuBus Address onto Ethernet Card

States S1 to S4 cover the data portion of a NuBus transfer. The signal DASB (Data Strobe) is generated to qualify all the data enabling signals (NICCS, ROMCS, RAMOE, NBDBOE, DBNBOE). Note that during S2 DASB has to be released so that the state has a separate state number. This only affects NuBus to RAM cycles. Therefore during S2 the signal TOP is used to qualify the data enabling signals (RAMOE, NBDBOE, DBNBOE).

During S1 the address is further decoded with four possible outcomes. Each of the possible transfers initiated by NuBus are described.

Bus Error Transfer

If the address is not recognized by any of the on-card devices or it is recognized by the ROM with transfer mode defining a write cycle, a bus error condition is flagged to the CPU. The sequencer enters state S5 where the DASB signal is cleared, ACK is generated to signal the end of the cycle, and a bus error code is driven onto the NuBus transfer mode lines. The sequencer then returns to idle on the next clock beat. *Figure 8* shows a timing diagram for a NuBus error cycle.

NuBus ROM Transfer

If the address and transfer mode are decoded as a read cycle to ROM, the address decoding PAL generates a ROM enabling signal to the ROM chip select input which drives its data onto the local bus. The control PALs generate the "Data bus to NuBus output enable" signal DBNBOE to enable this data from the local bus onto NuBus. The sequencer transfers to state S4 on the next clock beat where ACK is driven onto NuBus together with the "transfer complete" code on the transfer mode signals. *Figures 8* and *A-1* show a basic and detailed timing diagram of the ROM read operation. Note that ROM set up times are easily met. Very slow ROMs can be used on this design, up to 135 ns data enable time or 210 ns address access time.

NuBus NIC Transfer

If the address and transfer mode are decoded as a read or write cycle to the NIC register, the address decoding PAL generates the NIC chip select signal NICCS, and the bottom four bits of the local address are sent to the NIC to select one of sixteen possible NIC registers. The sequencer, *Figure 6,* remains in state S1 until the NIC generates acknowledge signal NICACK. This signal is synchronized to the NuBus clock before it is fed into the synchronous state sequencer PAL. The sequencer then proceeds onto state S4.

See *Figure 9* for a functional timing diagram of NuBus to NIC read and write cycles.

If the cycle is a write, the control PAL generates the NBDBOET signal to enable the NuBus write data onto the local bus, and a small asynchronous state machine in the PAL generates the write enable signal to the NIC, SWR. *Figure 10* shows the state machine diagram. This signal is cleared on the falling edge of the clock during the S4 state to provide the necessary write data hold time to the NIC. See *Figure A-3* detailed timing diagram.

If the cycle is a read, the control PAL generates DBNBOE to enable the NIC read data from the local bus onto NuBus. See *Figure A-2* for a detailed timing diagram.

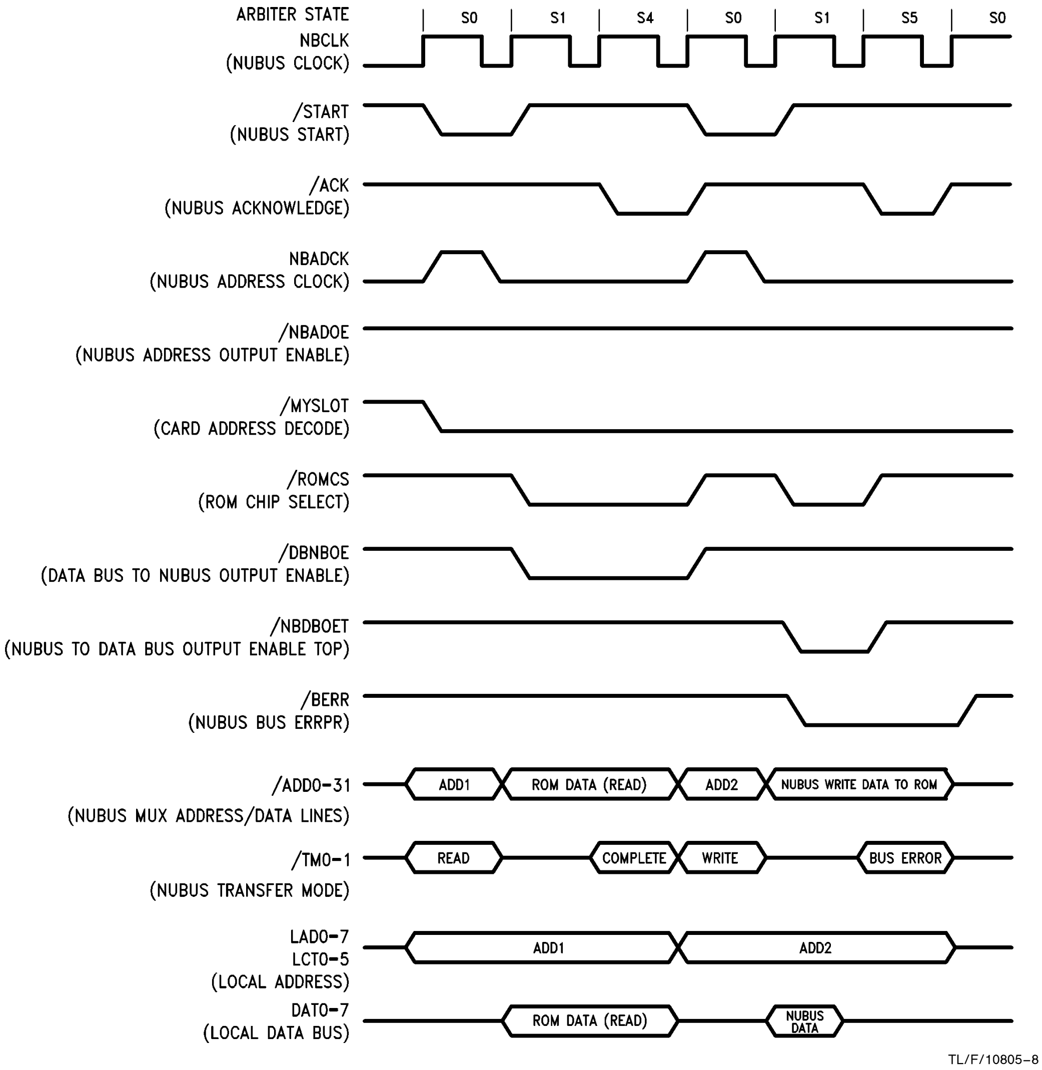
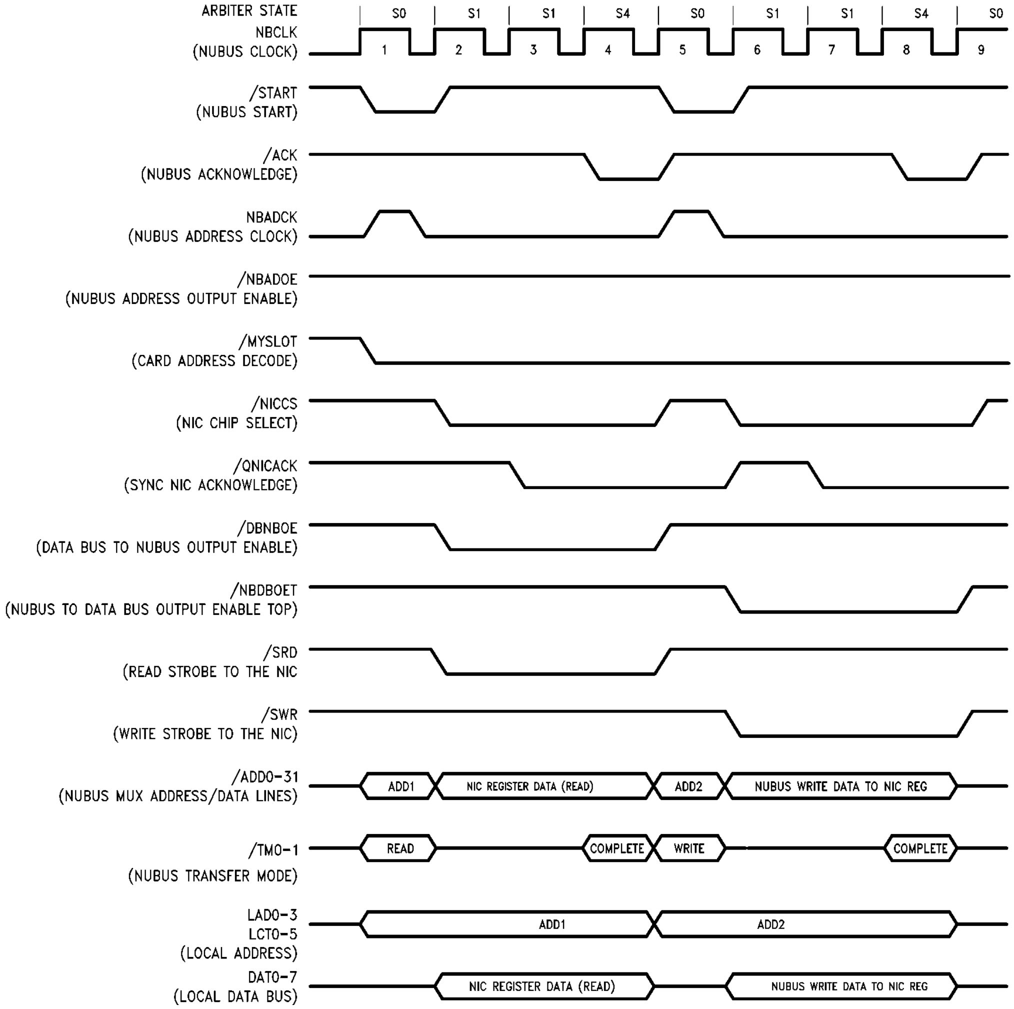
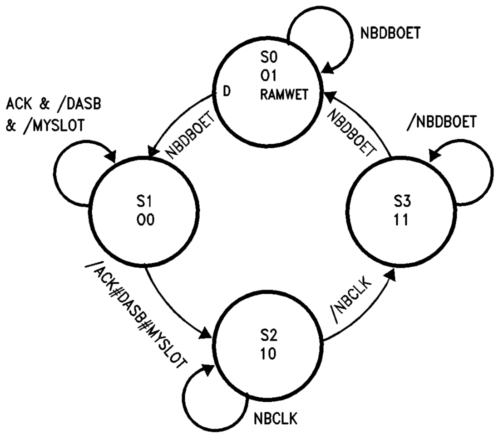


FIGURE 8. NuBus to ROM Read and Bus Error Cycles



TL/F/10805–10

FIGURE 9. NuBus to NIC Register Read and Write Cycle

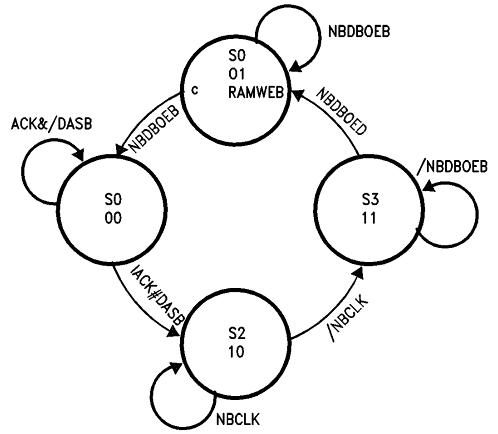


TL/F/10805-9

D = State Variable

RAMWET = RAM Write Enable Top

FIGURE 10. Write Enable Top State Diagram



TL/F/10805-11

C = State Variable (Not Used)

RAMWEB = RAM Write Enable Bottom

FIGURE 11. Write Enable Bottom State Diagram

NuBus to RAM Transfer

The address and transfer mode are decoded as a read or write cycle to the buffer RAM. The adapter card supports 32-bit NuBus transfers to the 16-bit buffer RAM. This is done by reading/writing to the RAM twice on every NuBus to RAM access, once during states S1 and S2 to access the least significant 16 bits of the NuBus word, and again during states S3 and S4 to access the most significant 16 bits of the NuBus word, after having incremented the bottom local address bit to the RAM (see the state diagram *Figure 6*). Therefore the NIC sees the buffer memory as an 8k x 16 RAM whereas the NuBus sees it as a 4k x 32 RAM. *Figure 12* shows a basic timing diagram for a NuBus to RAM read and write cycle.

For read cycles' data from the RAM read during states S1 and S2 is stored in the top two NuBus transceivers by setting the transceivers in storage mode and clocking them on the falling edge of the NuBus clock during state S2 with the TBCK (Top Bus Clock) signal. This data is enabled onto NuBus bits 16–31 with the signal NBDBOET. During states S3 and S4 the next RAM location is read and its data driven onto NuBus bits 0–15 through the bottom two NuBus transceivers, which are not set in storage mode (real time data mode).

Therefore by the time the adapter card drives ACK back to NuBus during state S4, the least significant 16 bits of data, corresponding to the first RAM location read, which were stored during S2, are being enabled onto NuBus bits 16–31 through the top two transceivers, and the most significant 16 bits of data, corresponding to the second RAM location read, are being enabled onto NuBus bits 0–15 through the bottom two NuBus transceivers. *Figure A-4* at the end of this note, shows a detailed timing diagram of a NuBus to RAM read cycle.

Note that the adapter card performs a hardware byte swap of NuBus data through the transceivers, so that the least significant byte of data from the RAM (Bits 0–7 on the local data bus of the first RAM location read) are driven onto byte 3 of NuBus (bits 24–31). This byte will be carried on byte lane 3 in the MACII system onto byte 3 of the MC68020 (data line bits 0–7).

For write cycles, during states S1 and S2, the top two NuBus transceivers (NuBus bits 16–31) are enabled onto the local data bus and their NuBus write data written into the Buffer RAM, with the bottom local address bit clear. During the next two states S3 and S4 the bottom two NuBus transceivers (NuBus bits 0–15) are enabled onto the local data bus and their NuBus write data written into the next Buffer RAM location with the bottom local address bit set. Two separate write enable signals are generated (RAMWET and RAMWEB) and ANDed together on the card to generate RAMWE. Two small asynchronous state machines are used to generate these signals. *Figure 10* and *11* show their state diagram. *Figure A-5* shows a detailed timing diagram of a NuBus write cycle to RAM.

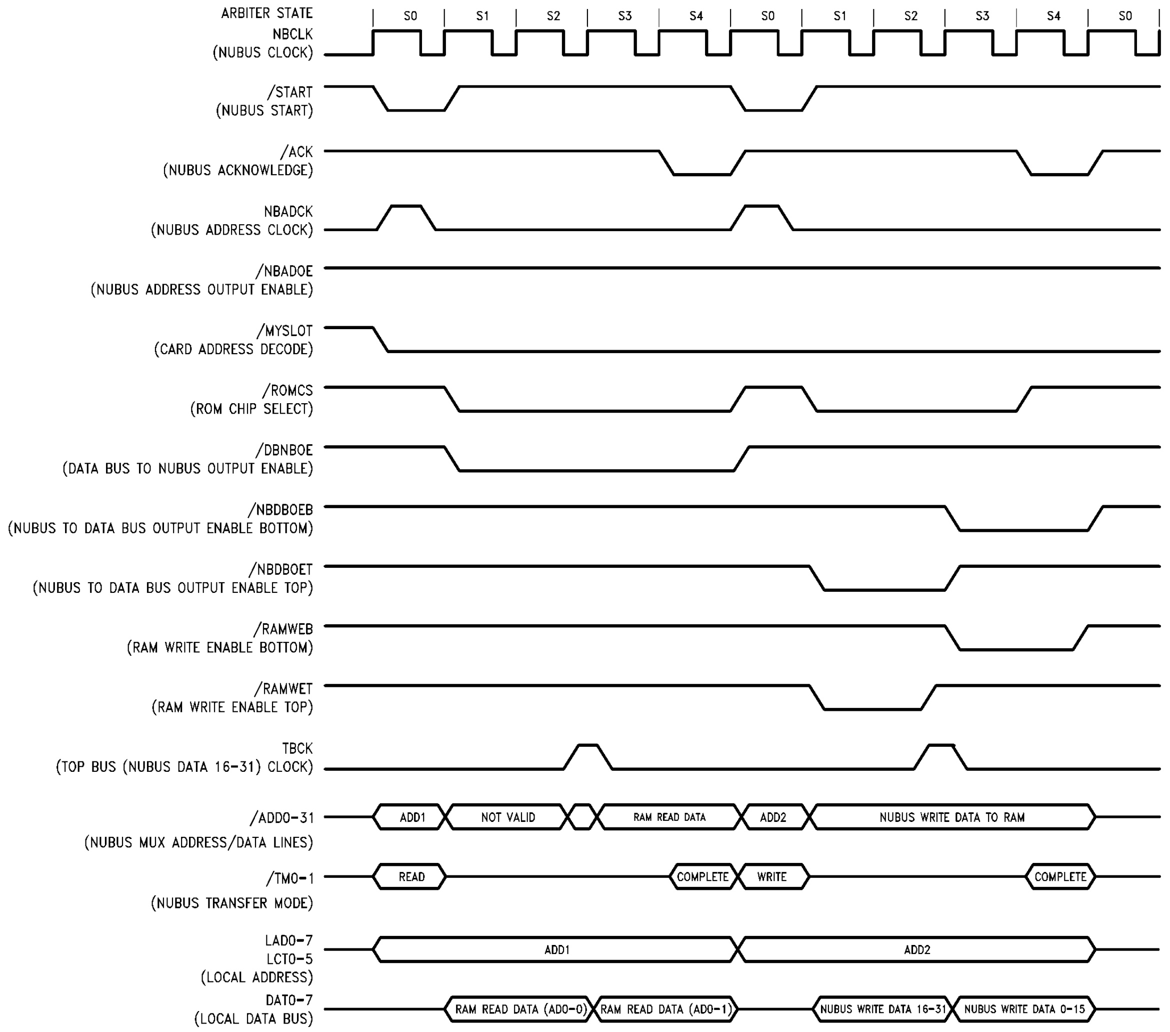
Supporting 32-bit transfers on NuBus rather than 16 only introduces one extra wait state per NuBus cycle to the RAM while doubling the data throughput per transfer.

NIC MASTER CYCLE

The NIC initiates local DMA cycles by driving its Bus Request line active. The sequencer/arbiter PAL, if in idle state S0, will enter state S6 where it acknowledges the request and hands over control of the local bus to the NIC. Any requests from the NuBus interface will be held until the NIC completes its local DMA burst and clears its request line allowing the sequencer PAL to return to the idle state S0.

Figure 14 shows an NIC to RAM cycle, its request coinciding with the start of a NuBus cycle, thus illustrating the arbitration process.

Note the NIC runs on a separate 20 MHz clock, asynchronous to the NuBus clock. Therefore NIC signals to the arbiter sequencer are first synchronized to the NuBus clock with a D-type latch (F175) before they are used by the synchronous sequencer PAL running on the NuBus clock. The signals affected are BREQ and NICACK.



TL/F/10805-12

FIGURE 12. NuBus to RAM Read and Write Cycles

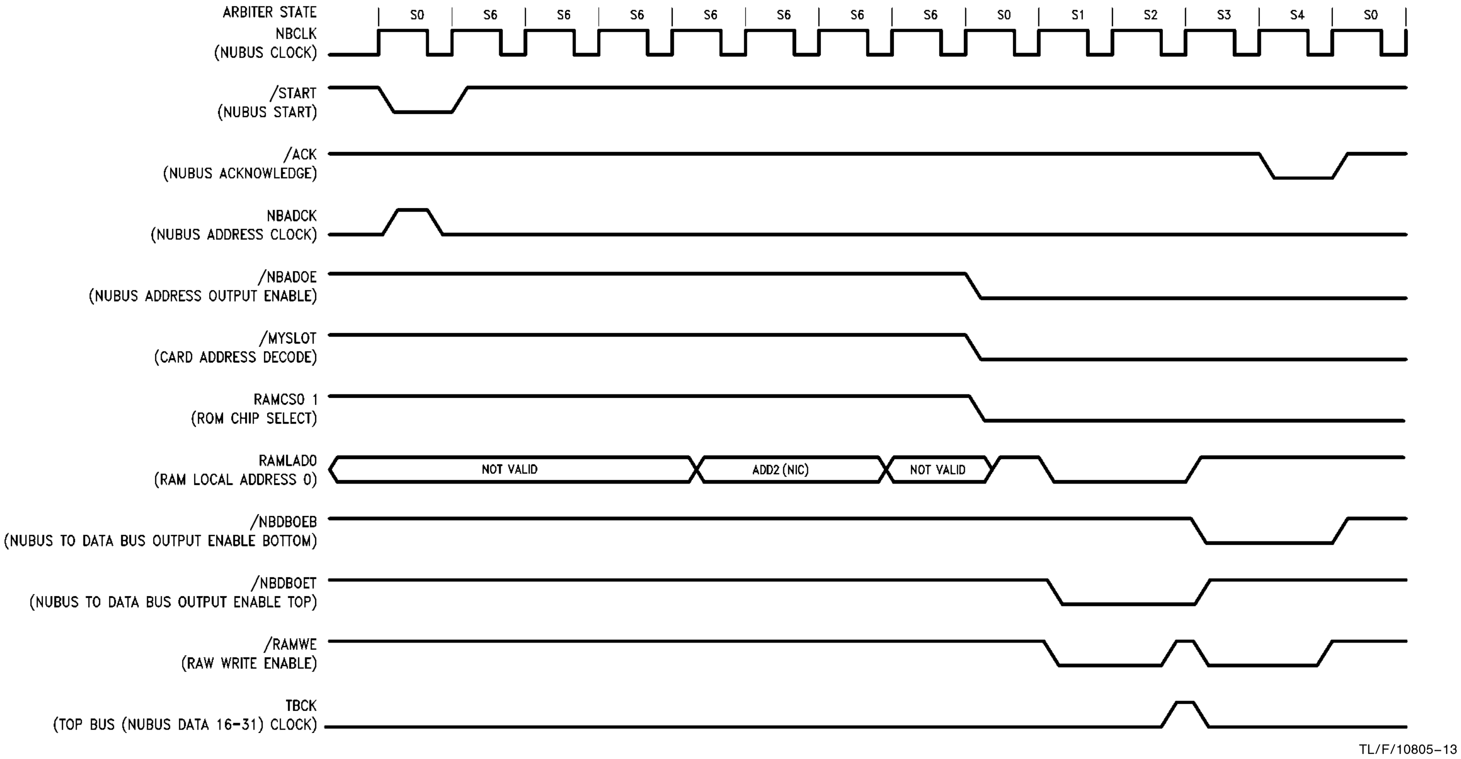
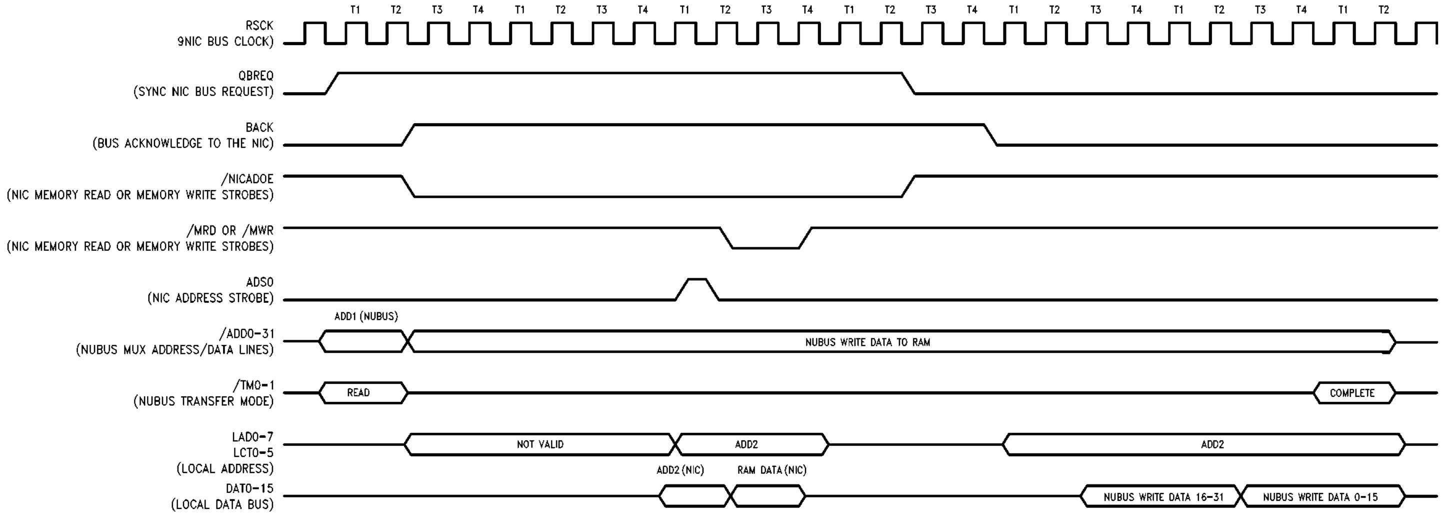
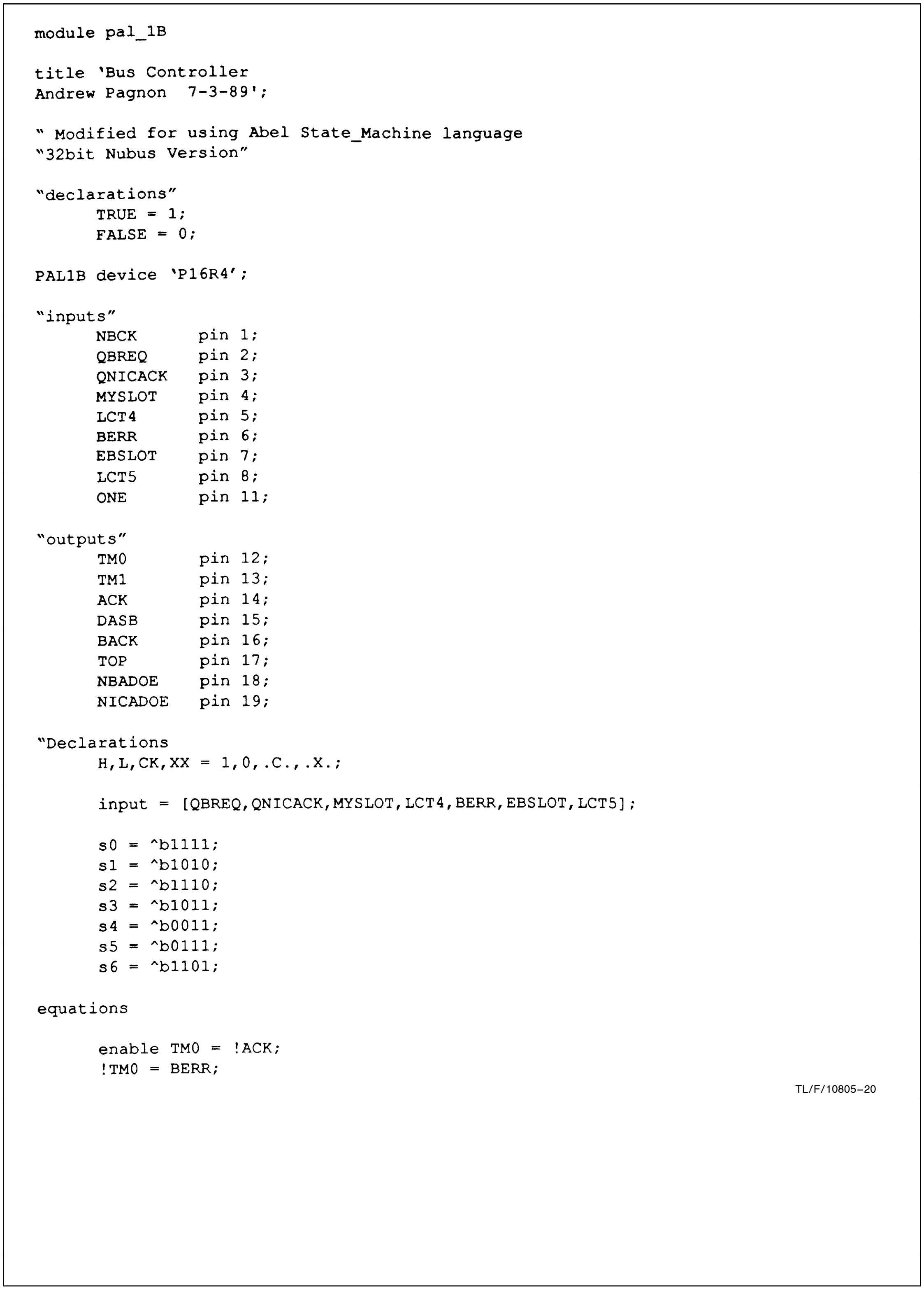


FIGURE 13. NuBus to RAM—Write Cycles with Arbitration



TL/F/10805-14

FIGURE 14. NIC to RAM—Write Cycles with Arbitration



```
module pal_1B

title 'Bus Controller
Andrew Pagnon  7-3-89';

" Modified for using Abel State_Machine language
"32bit Nubus Version"

"declarations"
      TRUE = 1;
      FALSE = 0;

PAL1B device 'P16R4';

"inputs"
      NBCK      pin 1;
      QBREQ     pin 2;
      QNICACK   pin 3;
      MYSLOT    pin 4;
      LCT4      pin 5;
      BERR      pin 6;
      EBSLOT    pin 7;
      LCT5      pin 8;
      ONE       pin 11;

"outputs"
      TM0       pin 12;
      TM1       pin 13;
      ACK       pin 14;
      DASB      pin 15;
      BACK      pin 16;
      TOP       pin 17;
      NBADOE    pin 18;
      NICADOE   pin 19;

"Declarations
      H,L,CK,XX = 1,0,.C.,.X.;

      input = [QBREQ,QNICACK,MYSLOT,LCT4,BERR,EBSLOT,LCT5];

      s0 = ^b1111;
      s1 = ^b1010;
      s2 = ^b1110;
      s3 = ^b1011;
      s4 = ^b0011;
      s5 = ^b0111;
      s6 = ^b1101;

equations

      enable TM0 = !ACK;
      !TM0 = BERR;
```

TL/F/10805-20

```
      enable TM1 = !ACK;
      !TM1 = !ACK;

      enable NBADOE = TRUE;
      !NBADOE = BACK & !ACK # BACK & !DASB # BACK & !QBREQ # BACK & !TOP;

        " NBADOE = !BACK* + BREQ . (ACK* . DASB* . BACK* . TOP*)"
  "NBADOE IS NOT ACTIVE IF BACK IS ACTIVE OR IF BREQ IS ACTIVE DURING S0"

      enable NICADOE = TRUE;
      !NICADOE = QBREQ & !BACK;

state_diagram [ACK,DASB,BACK,TOP]

      State s0:   case (input == [1,XX,XX,XX,XX,XX,XX])  :s6;
                       (input == [0,XX, 0,XX,XX, 0,XX])  :s1;
                       (input == [0,XX, 1,XX,XX,XX,XX])  :s0;  "hold
                  endcase;

      State s1:   case (input == [XX,XX,XX,XX, 0,XX,XX]) :s5;  "BERR
                       (input == [XX, 0, 0, 0, 1,XX,XX]) :s4;  "NIC
                       (input == [XX,XX, 0, 1, 1,XX, 1]) :s4;  "ROM
                       (input == [XX,XX, 0, 1, 1,XX, 0]) :s2;  "RAM
                       (input == [XX, 1, 0, 0, 1,XX,XX]) :s1;  "hold
                  endcase;

      State s2:   goto s3;

      State s3:   goto s4;

      State s4:   goto s0;

      State s5:   goto s0;

      State s6:   case (input == [0,XX,XX,XX,XX,XX,XX])  :s0;
                       (input == [1,XX,XX,XX,XX,XX,XX])  :s6; "hold
                  endcase;
```

TL/F/10805–21

```
test_vectors
([NBCK,QBREQ,QNICACK,MYSLOT,LCT4,BERR,EBSLOT,LCT5,ONE] ->
[TM0,TM1,ACK,DASB,BACK,TOP,NBADOE,NICADOE])
  [.C.,0,1,1,1,1,1,1,0]   -> [.X.,.X.,.X.,.X.,.X.,.X.,.X.,.X.];
  [.C.,0,1,1,1,1,1,1,0]   -> [.X.,.X.,.X.,.X.,.X.,.X.,.X.,.X.];
  [.C.,0,1,1,1,1,1,1,0]   -> [.X.,.X.,.X.,.X.,.X.,.X.,.X.,.X.];
  [.C.,0,1,1,1,1,1,1,0]   -> [.Z.,.Z.,1,1,1,1,0,1]; "TEST FOR IDLE         S0"
  [.C.,1,1,0,0,1,0,0,0]   -> [.Z.,.Z.,1,1,0,1,1,0]; "SET NIC BREQ          S6"
                                                    "NB TO NIC  START CYCLE  "
  [.C.,1,1,0,0,1,0,0,0]   -> [.Z.,.Z.,1,1,0,1,1,0]; "HOLD NIC BREQ         S6"
  [.C.,0,1,0,0,1,0,0,0]   -> [.Z.,.Z.,1,1,1,1,0,1]; "CLEAR NIC BREQ        S0"
  [.C.,0,1,0,0,1,0,0,0]   -> [.Z.,.Z.,1,0,1,0,0,1]; "NB TO NIC -           S1"
  [.C.,0,1,0,0,1,0,1,0]   -> [.Z.,.Z.,1,0,1,0,0,1]; "WAIT FOR NICACK       S1"
  [.C.,0,0,0,0,1,0,1,0]   -> [ 0 , 0 ,0,0,1,1,0,1]; "NICACK SETS           S4"
  [.C.,0,1,0,0,1,1,1,0]   -> [.Z.,.Z.,1,1,1,1,0,1]; "RETURN TO IDLE        S0"
  [.C.,0,1,0,0,1,1,1,0]   -> [.Z.,.Z.,1,1,1,1,0,1]; "IDLE WITH NO EBSLOT   S0"
  [.C.,0,1,0,1,1,0,0,0]   -> [.Z.,.Z.,1,0,1,0,0,1]; "NB TO RAM -           S1"
  [.C.,0,1,0,1,0,0,0,0]   -> [ 1 , 0 ,0,1,1,1,0,1]; "BERR SETS             S5"
  [.C.,1,1,0,1,0,1,1,0]   -> [.Z.,.Z.,1,1,1,1,1,1]; "IDLE NIC BREQ SETS    S0"
  [.C.,1,1,0,1,0,1,1,0]   -> [.Z.,.Z.,1,1,0,1,1,0]; "NIC MASTER            S6"
  [.C.,1,1,0,1,1,0,1,0]   -> [.Z.,.Z.,1,1,0,1,1,0]; "NB TO MEM -NIC MASTER S6"
  [.C.,0,1,0,1,1,0,1,0]   -> [.Z.,.Z.,1,1,1,1,0,1]; "RETURN TO IDLE        S0"
  [.C.,0,1,0,1,1,0,1,0]   -> [.Z.,.Z.,1,0,1,0,0,1]; "NB TO ROM -           S1"
  [.C.,1,1,0,1,1,0,1,0]   -> [ 0 , 0 ,0,0,1,1,0,1]; "NIC BREQ - SET NB ACK S4"
  [.C.,1,1,0,1,1,1,1,0]   -> [.Z.,.Z.,1,1,1,1,1,1]; "RETURN TO IDLE        S0"
  [.C.,1,1,1,1,1,1,1,0]   -> [.Z.,.Z.,1,1,0,1,1,0]; "NIC MASTER            S6"
  [.C.,0,1,1,1,1,1,1,0]   -> [.Z.,.Z.,1,1,1,1,0,1]; "RETURN TO IDLE        S0"
  [.C.,0,1,0,1,1,0,0,0]   -> [.Z.,.Z.,1,0,1,0,0,1]; "NB TO RAM             S1"
  [.C.,1,1,0,1,1,0,0,0]   -> [.Z.,.Z.,1,1,1,0,0,1]; "NIC BREQ-LTCH DA16-31 S2"
  [.C.,1,1,0,1,1,0,0,0]   -> [.Z.,.Z.,1,0,1,1,0,1]; "EB DA0-15             S3"
  [.C.,1,1,0,1,1,0,0,0]   -> [ 0 , 0 ,0,0,1,1,0,1]; "ACK to NUBUS          S4"
  [.C.,1,1,0,1,1,0,0,0]   -> [.Z.,.Z.,1,1,1,1,1,1]; "RETURN TO IDLE        S0"
  [.C.,1,1,1,1,1,1,1,0]   -> [.Z.,.Z.,1,1,0,1,1,0]; "NIC MASTER            S6"
  [.C.,0,1,1,1,1,1,1,0]   -> [.Z.,.Z.,1,1,1,1,0,1]; "RETURN TO IDLE        S0"
  [.C.,0,1,1,1,1,1,1,0]   -> [.Z.,.Z.,1,1,1,1,0,1]; "STAY IN IDLE          S0"


end  pal_1B
```

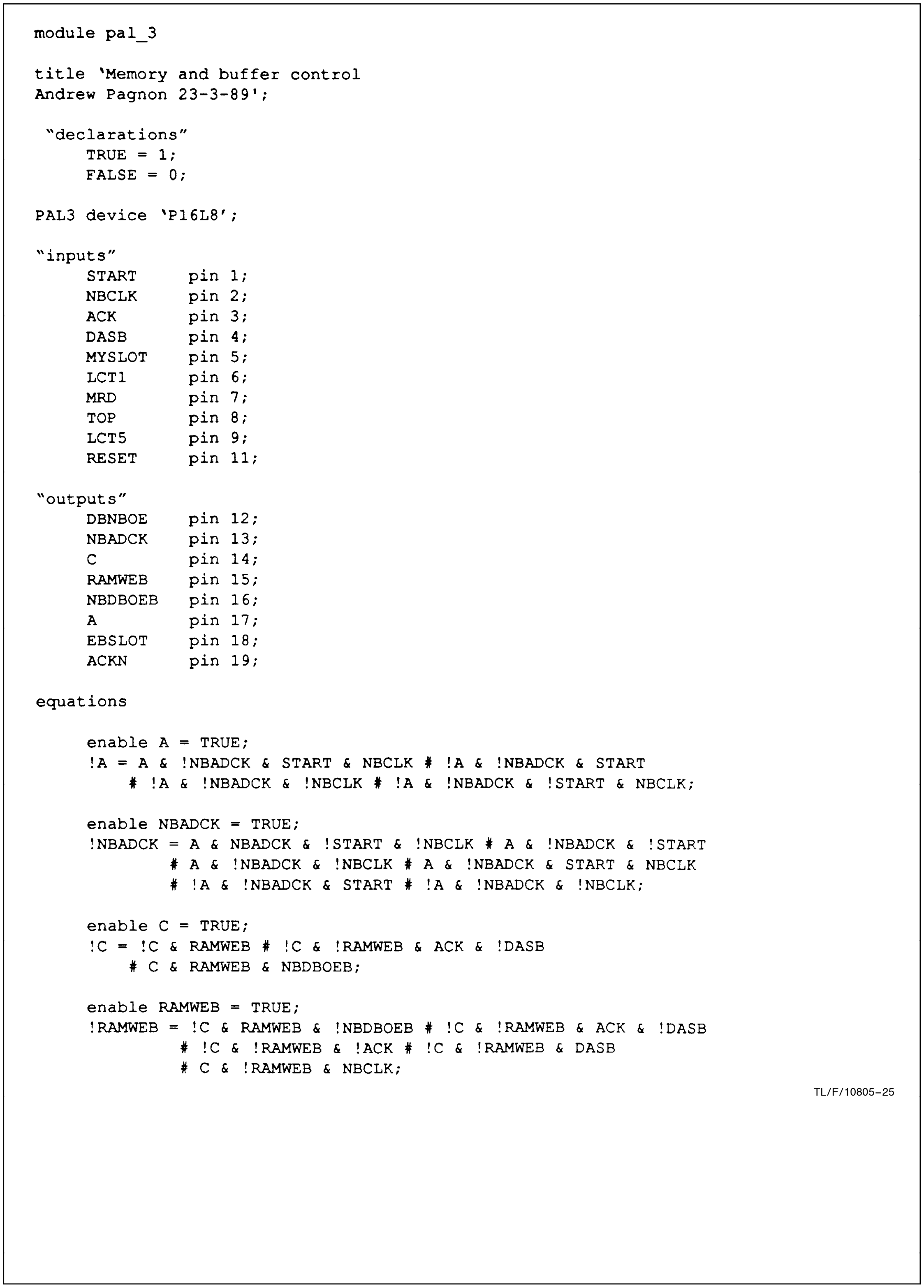
TL/F/10805-22

```
module pal_2

title 'Memory decoder
Andrew Pagnon  8-3-89';

 "declarations"
     TRUE = 1;
     FALSE = 0;

PAL2 device 'P16L8';

"inputs"
     LCT0      pin 1;
     LCT1      pin 2;
     LCT2      pin 3;
     LCT3      pin 4;
     LCT4      pin 5;
     LCT5      pin 6;
     MYSLOT    pin 7;
     DASB      pin 8;
     NICADOE   pin 9;
     MSRAMSL   pin 11;

"outputs"
     RAMCS1    pin 12;
     ROMCS     pin 13;
     NICCS     pin 14;
     RAMCS0    pin 15;
     RAMCS3    pin 16;
     RAMCS2    pin 17;
     BERR      pin 18;
     SRD       pin 19;

equations

     enable RAMCS0 = TRUE;

     !RAMCS0 = !NICADOE & !MSRAMSL
     #  LCT0 &  LCT1 & !LCT2 & !LCT3 & LCT4 & !LCT5 & !MYSLOT " Write byte 0 "
     # !LCT0 &  LCT1 &  LCT2 & !LCT3 & LCT4 & !LCT5 & !MYSLOT " Write hw 0 "
     # !LCT0 &  LCT1 & !LCT2 & !LCT3 & LCT4 & !LCT5 & !MYSLOT " Write word "
     # !LCT1 &  LCT4 & !LCT5 & !MYSLOT; " Read "

     enable RAMCS1 = TRUE;

     !RAMCS1 = !NICADOE & !MSRAMSL
     #  LCT0 &  LCT1 &  LCT2 & !LCT3 & LCT4 & !LCT5 & !MYSLOT " Write byte 1 "
     # !LCT0 &  LCT1 &  LCT2 & !LCT3 & LCT4 & !LCT5 & !MYSLOT " Write hw 0 "
     # !LCT0 &  LCT1 & !LCT2 & !LCT3 & LCT4 & !LCT5 & !MYSLOT " Write word "
     # !LCT1 &  LCT4 & !LCT5 & !MYSLOT; " Read "
```

TL/F/10805–23

```
     enable RAMCS2 = TRUE;

     !RAMCS2 = !NICADOE & MSRAMSL
     #  LCT0 &  LCT1 & !LCT2 &  LCT3 & LCT4 & !LCT5 & !MYSLOT " Write byte 2”
     # !LCT0 &  LCT1 &  LCT2 &  LCT3 & LCT4 & !LCT5 & !MYSLOT " Write hw 1 "
     # !LCT0 &  LCT1 & !LCT2 & !LCT3 & LCT4 & !LCT5 & !MYSLOT " Write word "
     # !LCT1 & LCT4 & !LCT5 & !MYSLOT; " Read "

     enable RAMCS3 = TRUE;

     !RAMCS3 = !NICADOE & MSRAMSL
     #  LCT0 &  LCT1 &  LCT2 &  LCT3 & LCT4 & !LCT5 & !MYSLOT  " Write byte 3 "
     # !LCT0 &  LCT1 &  LCT2 &  LCT3 & LCT4 & !LCT5 & !MYSLOT  " Write hw 1 "
     # !LCT0 &  LCT1 & !LCT2 & !LCT3 & LCT4 & !LCT5 & !MYSLOT  " Write word "
     # !LCT1 &  LCT4 & !LCT5 & !MYSLOT; " Read "

     enable ROMCS = TRUE;
     !ROMCS = !LCT1 & LCT4 & LCT5 & !MYSLOT & !DASB; " Read "

     enable NICCS = TRUE;
     !NICCS = !DASB & !MYSLOT & !LCT4 & LCT5; " NIC Register Read or Write "

     enable SRD = TRUE;
     !SRD = !NICCS & !LCT1; " NIC register read "

     enable BERR = TRUE;
     !BERR = LCT1 & LCT4 & LCT5 & !MYSLOT     "ROM WRITE”
           # !LCT4 & !LCT5 & !MYSLOT;         "NOT IN CARD”

test_vectors
([LCT0,LCT1,LCT2,LCT3,LCT4,LCT5,MYSLOT,DASB,NICADOE,MSRAMSL] ->
[RAMCS0,RAMCS1,RAMCS2,RAMCS3,ROMCS,NICCS,BERR,SRD])
[.X.,.X.,.X.,.X.,.X.,.X.,1,.X.,0,0] -> [0,0,1,1,1,1,1,1];”NIC RD/WT RAM HW0"
[.X.,.X.,.X.,.X.,.X.,.X.,1,.X.,0,1] -> [1,1,0,0,1,1,1,1];”NIC RD/WT RAM HW1"
[1,1,0,0,0,1,0,0,1,.X.]             -> [1,1,1,1,1,0,1,1];”NB WT NIC BYTE0"
[0,0,0,0,0,1,0,0,1,.X.]             -> [1,1,1,1,1,0,1,0];”NB RD NIC WD”
[0,0,0,0,1,1,0,.X.,1,.X.]           -> [1,1,1,1,0,1,1,1];”NB RD ROM WD”
[0,0,0,0,1,0,0,.X.,1,.X.]           -> [0,0,0,0,1,1,1,1];”NB RD RAM WD”
[1,1,0,0,1,0,0,.X.,1,.X.]           -> [0,1,1,1,1,1,1,1];”NB WT RAM BYTE0"
[1,1,1,0,1,0,0,.X.,1,.X.]           -> [1,0,1,1,1,1,1,1];”NB WT RAM BYTE1"
[1,1,0,1,1,0,0,.X.,1,.X.]           -> [1,1,0,1,1,1,1,1];”NB WT RAM BYTE2"
[1,1,1,1,1,0,0,.X.,1,.X.]           -> [1,1,1,0,1,1,1,1];”NB WT RAM BYTE3"
[0,1,1,0,1,0,0,.X.,1,.X.]           -> [0,0,1,1,1,1,1,1];”NB WT RAM HW0"
[0,1,1,1,1,0,0,.X.,1,.X.]           -> [1,1,0,0,1,1,1,1];”NB WT RAM HW1"
[0,1,0,0,1,0,0,.X.,1,.X.]           -> [0,0,0,0,1,1,1,1];”NB WT RAM WD”
[1,0,.X.,.X.,1,0,0,.X.,1,.X.]       -> [0,0,0,0,1,1,1,1];”NB RD RAM BY    BERR”
[0,0,.X.,1,1,0,0,.X.,1,.X.]         -> [0,0,0,0,1,1,1,1];”NB RD RAM H1,BL BERR”
[0,0,1,0,1,0,0,.X.,1,.X.]           -> [0,0,0,0,1,1,1,1];”NB RD RAM H0    BERR”
[.X.,1,.X.,.X.,1,1,0,.X.,1,.X.]     -> [1,1,1,1,1,1,0,1];”NB WT ROM       BERR”
[.X.,.X.,.X.,.X.,0,0,0,.X.,1,.X.] -> [1,1,1,1,1,1,0,1];”ADD NOT IN CARD BERR”

end pal_2
```

TL/F/10805–24



```
module pal_3

title ‘Memory and buffer control
Andrew Pagnon 23-3-89';

 “declarations”
     TRUE = 1;
     FALSE = 0;

PAL3 device ‘P16L8’;

“inputs”
     START     pin 1;
     NBCLK     pin 2;
     ACK       pin 3;
     DASB      pin 4;
     MYSLOT    pin 5;
     LCT1      pin 6;
     MRD       pin 7;
     TOP       pin 8;
     LCT5      pin 9;
     RESET     pin 11;

“outputs”
     DBNBOE    pin 12;
     NBADCK    pin 13;
     C         pin 14;
     RAMWEB    pin 15;
     NBDBOEB   pin 16;
     A         pin 17;
     EBSLOT    pin 18;
     ACKN      pin 19;

equations

     enable A = TRUE;
     !A = A & !NBADCK & START & NBCLK # !A & !NBADCK & START
         # !A & !NBADCK & !NBCLK # !A & !NBADCK & !START & NBCLK;

     enable NBADCK = TRUE;
     !NBADCK = A & NBADCK & !START & !NBCLK # A & !NBADCK & !START
             # A & !NBADCK & !NBCLK # A & !NBADCK & START & NBCLK
             # !A & !NBADCK & START # !A & !NBADCK & !NBCLK;

     enable C = TRUE;
     !C = !C & RAMWEB # !C & !RAMWEB & ACK & !DASB
         # C & RAMWEB & NBDBOEB;

     enable RAMWEB = TRUE;
     !RAMWEB = !C & RAMWEB & !NBDBOEB # !C & !RAMWEB & ACK & !DASB
              # !C & !RAMWEB & !ACK # !C & !RAMWEB & DASB
              # C & !RAMWEB & NBCLK;
```

TL/F/10805-25

```
     enable NBDBOEB = TRUE;
     !NBDBOEB = LCT1 & !DASB & !MYSLOT & TOP & !LCT5;

     enable DBNBOE = TRUE;
     !DBNBOE = !LCT1 & !DASB & !MYSLOT
             # !LCT1 & !TOP & !MYSLOT;

     enable EBSLOT = TRUE;
     !EBSLOT = EBSLOT & !START & RESET # !EBSLOT & ACK & RESET;

     enable ACKN = !ACK;
     !ACKN = !ACK;


test_vectors
([START,NBCLK,ACK,DASB,MYSLOT,LCT1,TOP,LCT5,RESET] ->
[A,NBADCK,C,RAMWEB,NBDBOEB,DBNBOE,EBSLOT,ACKN])
[1,1,1,1,1,.X.,1,.X.,0] -> [.X.,.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[1,0,1,1,1,.X.,1,.X.,1] -> [.X.,.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[1,1,1,1,1,.X.,1,.X.,1] -> [.X.,.X.,.X.,.X.,1,1,1,.Z.];
[1,0,1,1,1,.X.,1,.X.,1] -> [.X.,.X.,.X.,.X.,1,1,1,.Z.];
[0,1,1,1,1,.X.,1,.X.,1] -> [.X.,1,0,1,1,1,0,.Z.];"S0,START LOW NBCKL HIGH"
[0,1,1,1,0,1,1,0,1]     -> [1,1,0,1,1,1,0,.Z.];"S0,MYSLOT* LOW,S0"
[0,0,1,1,0,1,1,0,1]     -> [1,0,0,1,1,1,0,.Z.];"S0,NBCLK1=0,S0"
[1,1,1,0,0,1,0,0,1]     -> [0,0,0,1,1,1,0,.Z.];"S1,NBCLK2=1,START*=1"
[1,1,1,0,0,1,0,0,1]     -> [0,0,0,1,1,1,0,.Z.];"S1"
[1,0,1,0,0,1,0,0,1]     -> [0,0,0,1,1,1,0,.Z.];"S1,NBCLK2=0"
[1,1,1,1,0,1,0,0,1]     -> [0,0,0,1,1,1,0,.Z.];"S2 NBCLK3=1"
[1,1,1,1,0,1,0,0,1]     -> [0,0,0,1,1,1,0,.Z.];"S2"
[1,1,1,1,0,1,0,0,1]     -> [0,0,0,1,1,1,0,.Z.];"S2"
[1,0,1,1,0,1,0,0,1]     -> [0,0,0,1,1,1,0,.Z.];"S2,NBCLK3=0"
[1,1,1,0,0,1,1,0,1]     -> [0,0,0,0,0,1,0,.Z.];"S3,NBCLK4=1"
[1,1,1,0,0,1,1,0,1]     -> [0,0,0,0,0,1,0,.Z.];"S3"
[1,0,1,0,0,1,1,0,1]     -> [0,0,0,0,0,1,0,.Z.];"S3,NBCLK=0"
[1,1,0,0,0,1,1,0,1]     -> [0,0,1,0,0,1,1,0];  "S4"
[1,0,0,0,0,1,1,0,1]     -> [0,0,1,1,0,1,1,0];  "S4"
[1,1,1,1,0,1,1,0,1]     -> [0,0,0,1,1,1,1,.Z.];"S0"
[1,0,1,1,0,1,1,0,1]     -> [0,0,0,1,1,1,1,.Z.];"S0"
[0,1,1,1,0,1,1,1,1]     -> [1,1,0,1,1,1,0,.Z.];"S0,START*=0,NBCLK5=1"
[0,1,1,1,0,1,1,1,1]     -> [1,1,0,1,1,1,0,.Z.];"S0,NIC OR ROM WRITE"
[0,0,1,1,0,1,1,1,1]     -> [1,0,0,1,1,1,0,.Z.];"S0,NBCLK5=0"
[0,0,1,1,0,1,1,1,1]     -> [1,0,0,1,1,1,0,.Z.];"S0,MYSLOT*=1"
[1,1,1,0,0,1,0,1,1]     -> [0,0,0,1,1,1,0,.Z.];"S1,NBCLK6=1"
[1,0,1,0,0,1,0,1,1]     -> [0,0,0,1,1,1,0,.Z.];"S1"
[1,1,0,0,0,1,1,1,1]     -> [0,0,0,1,1,1,1,0];  "S4"
[1,0,0,0,0,1,1,1,1]     -> [0,0,0,1,1,1,1,0];  "S4"
[1,1,1,1,0,1,1,1,1]     -> [0,0,0,1,1,1,1,.Z.];"S0"
[1,0,1,1,.X.,.X.,1,.X.,1]->[0,0,0,1,1,1,1,.Z.];"S0"

end pal_3
```

TL/F/10805–26

```
module pal_4

title ’32 bit Nubus control
Andrew Pagnon  11-10-89';

 “declarations”
     TRUE = 1;
     FALSE = 0;

PAL6 device ‘P16L8’;

“inputs”
     TOP       pin 1;
     NBCLK     pin 2;
     DASB      pin 3;
     BACK      pin 4;
     LAD0      pin 5;
     MYSLOT    pin 6;
     ACK       pin 7;
     BERR      pin 8;
     LCT1      pin 9;
     LCT4      pin 11;
     LCT5      pin 18;

“outputs”
     RAMLAD0   pin 12;
     TBCK      pin 13;
     NBDBOET   pin 14;
     RAMWET    pin 15;
     D         pin 16;
     DBNBOEB   pin 17;

equations

     enable RAMLAD0 = TRUE;
     !RAMLAD0 = !BACK & !LAD0 # BACK & !TOP;

     enable TBCK = TRUE;
     !TBCK = TOP # NBCLK # !DASB;

     enable NBDBOET = TRUE;
     !NBDBOET = LCT1 & !MYSLOT & !TOP “RAM”
            # LCT1 & !MYSLOT & !DASB & BERR & !LCT4; “NIC”

     enable RAMWET = TRUE;
     !RAMWET = !D & RAMWET & !NBDBOET
             # !D & !RAMWET & !DASB & !MYSLOT & ACK
             # !D & !RAMWET & DASB
             # !D & !RAMWET & !ACK
             # !D & !RAMWET & MYSLOT
             # D & !RAMWET & NBCLK;

     enable D = TRUE;
     !D = !D & RAMWET
        # !D & !RAMWET & !DASB & ACK & !MYSLOT
       # D & RAMWET & NBDBOET;
```

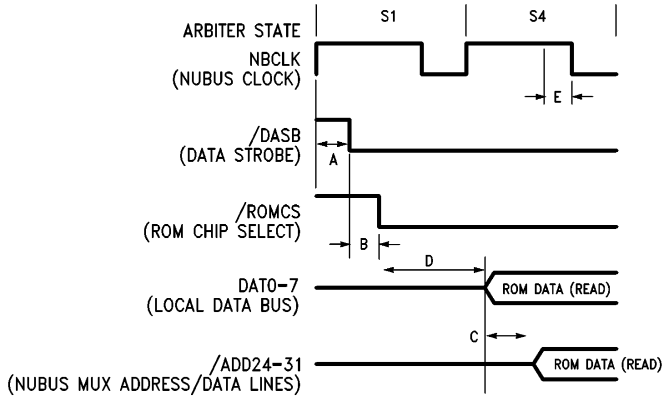
TL/F/10805–27

```
     enable DBNBOEB = TRUE;
     !DBNBOEB = !LCT1 & !DASB & !MYSLOT & !LCT5
              # !LCT1 & !TOP & !MYSLOT & !LCT5;

test_vectors
([TOP,NBCLK,DASB,BACK,LAD0,MYSLOT,ACK,BERR,LCT1,LCT4,LCT5] ->
[RAMLAD0,TBCK,NBDBOET,RAMWET,DBNBOEB])
[1,1,1,0,1,1,1,1,.X.,.X.,.X.] -> [.X.,.X.,.X.,.X.,.X.];”RESET”
[1,0,1,0,1,1,1,1,.X.,.X.,.X.] -> [.X.,.X.,.X.,.X.,.X.];”RESET”
[1,1,1,0,1,1,1,1,.X.,.X.,.X.] -> [1,0,1,1,1];”LAD0=1,BACK*=0"
[1,0,1,0,1,1,1,1,.X.,.X.,.X.] -> [1,0,1,1,1];
[1,1,1,0,0,1,1,1,.X.,.X.,.X.] -> [0,0,1,1,1];”LAD0=0,BACK*=0"
[1,0,1,0,0,1,1,1,.X.,.X.,.X.] -> [0,0,1,1,1];
[1,1,1,1,0,1,1,1,.X.,.X.,.X.] -> [0,0,1,1,1];”LAD0=0,BACK*=1,TOP=1->RAMLAD0=1"
[1,0,1,1,0,1,1,1,.X.,.X.,.X.] -> [0,0,1,1,1];
[0,1,0,1,0,0,1,1,1,0,1]     -> [1,0,0,0,1];”S1,NIC WT”
[0,0,0,1,.X.,0,1,1,1,0,1]   -> [1,0,0,0,1];”S1,NIC WT,NBCLK=0"
[1,1,0,1,.X.,0,0,1,1,0,1]   -> [0,0,0,0,1];”S4,NIC WT,ACK=0"
[1,0,0,1,.X.,0,0,1,1,0,1]   -> [0,0,0,1,1];”S4,NIC WT,ACK=0,NBCLK=0"
[1,1,1,1,.X.,0,1,1,1,0,1]   -> [0,0,1,1,1];”S0"
[1,0,1,1,.X.,0,1,1,0,1,1]     -> [0,0,1,1,1];”S0"
[0,1,0,1,.X.,0,1,1,0,1,1]     -> [1,0,1,1,1];”S1,ROM RD”
[0,0,0,1,.X.,0,1,1,0,1,1]     -> [1,0,1,1,1];”S1,ROM RD,NBCLK=0"
[1,1,0,1,.X.,0,0,1,0,1,1]     -> [0,0,1,1,1];”S4,ROM RD,ACK=0"
[1,0,0,1,.X.,0,0,1,0,1,1]     -> [0,0,1,1,1];”S4,ROM RD,ACK=0,NBCLK=0"
[1,1,1,1,.X.,0,1,1,0,1,1]     -> [0,0,1,1,1];”S0"
[1,0,1,1,.X.,0,1,1,0,1,0]     -> [0,0,1,1,1];”S0"
[0,1,0,1,.X.,0,1,1,0,1,0]     -> [1,0,1,1,0];”S1,RAM RD”
[0,0,0,1,.X.,0,1,1,0,1,0]     -> [1,0,1,1,0];”S1,RAM RD,NBCLK=0"
[0,1,1,1,.X.,0,1,.X.,0,1,0]   -> [1,0,1,1,0];”S2,RAM RD”
[0,0,1,1,.X.,0,1,.X.,0,1,0]   -> [1,1,1,1,0];”S2,RAM RD,NBCLK=0,TBCK=1"
[1,1,0,1,.X.,0,1,.X.,0,1,0]   -> [0,0,1,1,0];”S3,RAM RD,TOP=1"
[1,0,0,1,.X.,0,1,.X.,0,1,0]   -> [0,0,1,1,0];”S3,RAM RD,NBCLK=0"
[1,1,0,1,.X.,0,1,.X.,0,1,0]   -> [0,0,1,1,0];”S4,RAM RD,ACK=0"
[1,0,0,1,.X.,0,0,.X.,0,1,0]   -> [0,0,1,1,0];”S4,RAM RD,ACK=0,NBCLK=0"
[1,1,1,1,.X.,0,1,.X.,0,1,0]   -> [0,0,1,1,1];”S0"
[1,0,1,1,.X.,0,1,1,1,1,0]     -> [0,0,1,1,1];”S0"
[0,1,0,1,.X.,0,1,1,1,1,0]     -> [1,0,0,0,1];”S1,RAM WT”
[0,0,0,1,.X.,0,1,1,1,1,0]     -> [1,0,0,0,1];”S1,RAM WT,NBCLK=0"
[1,1,1,1,.X.,0,0,.X.,1,1,0]   -> [0,0,1,0,1];”S5,RAM WT,BERR=0"
[1,0,1,1,.X.,0,0,.X.,1,1,0]   -> [0,0,1,1,1];”S5,RAM WT,BERR=0,NBCLK=0"
[1,1,1,1,.X.,0,1,.X.,1,1,0]   -> [0,0,1,1,1];”S0"
[1,0,1,1,.X.,0,1,1,1,1,1]     -> [0,0,1,1,1];”S0"
[0,1,0,1,.X.,0,1,1,1,1,1]     -> [1,0,0,0,1];”S1,ROM WT”
[0,0,0,1,.X.,0,1,1,1,1,1]     -> [1,0,0,0,1];”S1,ROM WT,NBCLK=0"
[1,1,1,1,.X.,0,0,.X.,1,1,1]   -> [0,0,1,0,1];”S5,ROM WT,BERR=0"
[1,0,1,1,.X.,0,0,.X.,1,1,1]   -> [0,0,1,1,1];”S5,ROM WT,BERR=0,NBCLK=0"
[1,1,1,1,.X.,0,1,.X.,1,1,1]   -> [0,0,1,1,1];”S0"
[1,0,1,1,.X.,0,1,1,1,1,0]     -> [0,0,1,1,1];”S0"
[0,1,0,1,.X.,0,1,1,1,1,0]     -> [1,0,0,0,1];”S1,RAM WT”
[0,0,0,1,.X.,0,1,1,1,1,0]     -> [1,0,0,0,1];”S1,RAM WT,NBCLK=0"
[0,1,1,1,.X.,0,1,.X.,1,1,0]   -> [1,0,0,0,1];”S2,RAM WT”
[0,0,1,1,.X.,0,1,.X.,1,1,0]   -> [1,1,0,1,1];”S2,RAM WT,NBCLK=0,TBCK=1"
[1,1,0,1,.X.,0,1,.X.,1,1,0]   -> [0,0,1,1,1];”S3,RAM WT,TOP=1"
[1,0,0,1,.X.,0,1,.X.,1,1,0]   -> [0,0,1,1,1];”S3,RAM WT,NBCLK=0"
```

TL/F/10805–28

```
[1,1,0,1,.X.,0,1,.X.,1,1,0]   -> [0,0,1,1,1];"S4,RAM WT,ACK=0"
[1,0,0,1,.X.,0,0,.X.,1,1,0]   -> [0,0,1,1,1];"S4,RAM WT,ACK=0,NBCLK=0"
[1,1,1,1,.X.,0,1,.X.,1,1,0]   -> [0,0,1,1,1];"S0"
[1,0,1,1,.X.,0,1,.X.,0,1,1]   -> [0,0,1,1,1];"S0"
[0,1,0,1,.X.,0,1,.X.,0,1,1]   -> [1,0,1,1,1];"S1,ROM RD"
[0,0,0,1,.X.,0,1,1,0,1,1]     -> [1,0,1,1,1];"S1,ROM RD,NBCLK=0"
[1,1,0,1,.X.,0,0,.X.,0,1,1]   -> [0,0,1,1,1];"S4,ROM RD,ACK=0"
[1,0,0,1,.X.,0,0,.X.,0,1,1]   -> [0,0,1,1,1];"S4,ROM RD,ACK=0,NBCLK=0"
[1,1,1,1,.X.,0,1,.X.,0,1,1]   -> [0,0,1,1,1];"S0"
[1,0,1,1,.X.,1,1,.X.,.X.,.X.,.X.] -> [1,0,1,1,1];"S0"
[1,1,1,1,.X.,1,1,.X.,.X.,.X.,.X.] -> [1,0,1,1,1];"S0"
[.X.,.X.,.X.,0,0,.X.,.X.,.X.,.X.,.X.,.X.] ->
[0,.X.,.X.,.X.,.X.];"BACK*=0,LAD&RLAD=0"
[.X.,.X.,.X.,0,1,.X.,.X.,.X.,.X.,.X.,.X.] ->
[1,.X.,.X.,.X.,.X.];"BACK*=0,LAD&RLAD=1"

end pal_6
```

TL/F/10805–29



TL/F/10805–15

FIGURE A-1. Detailed NuBus to ROM Read Timing

TIMINGS

A = 15 ns, 16R4B prop delay, NbClk to DaSb

B = 15 ns, 16L8B prop delay, DaSb to ROMCS

C = 8 ns, F651 prop delay, Data Bus to NuBus

D = 35 ns, Max ROM enable access time, ROMCS to Data Bus

E = 21 ns, NuBus data set up time, NuBus data to NbClk low (S4)

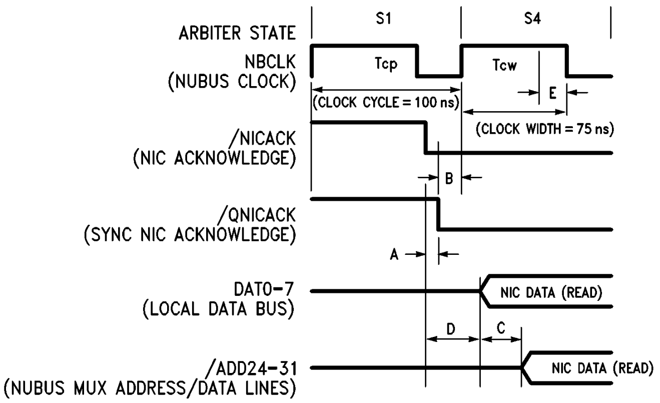
Note: The address to the ROM is valid midway through the S0 state. Therefore the ROM address access time (70 ns) is not in the timing critical path.

ROM READ DATA SET UP TIME

Spec Times To Be Met
Read data set up time to NuBus
(NbClk(S4)low) = 21 ns
(Tsu in Nubus spec)

Adapter Card Times
Read data set up time
= Tcp(S1) + Tcw(S4) − A − B − C − D
= 100 + 75 + 15 + 15 − 8 − 35
= 102 ns (81 ns to spare)

Data hold times are as per RAM read cycles



TL/F/10805–16

FIGURE A-2. Detailed NuBus to DP8390 Read Timing

TIMINGS

A = 4 ns, Min F175 prop delay, NICACK to QNICACK

B = 10 ns, Typ 16R4B Set Up, QNICACK to NBClk

C = 8 ns, F651 prop delay, Data Bus to NuBus

D = 55 ns, NIC Register access time, NICACK to Data Bus

E = 21 ns, NuBus Data set up time, NuBus data to NbClk

Data hold times are as per RAM read cycles

NIC REGISTER READ DATA SET UP TIME

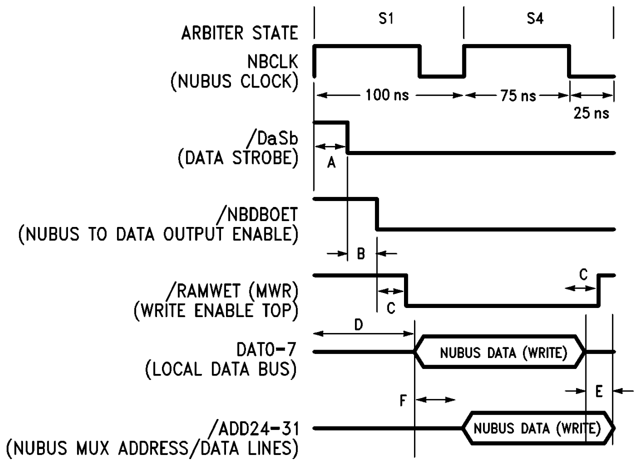
Spec Times To Be Met

Read data set up time to NuBus(NbClk(S4)low) = 21 ns (Tsu in NuBus spec)

Adapter Card Times

Read data set up time =
Tcw(S2) + A + B − C − D = 75 + 4 + 10 − 8 − 55
= 26 ns (5 ns to spare)

(Note B is a typical value)



TL/F/10805–17

FIGURE A-3. NuBus to NIC Write Timing

TIMINGS

A+B+C = 45 ns, 16R4B and 16L8B delay,
NbClk to RAMWET

D = 52 ns, NuBus data enable (Ton+2Tpd), DaSb to ROMCS

E = 8 ns, F651 prop delay, Data Bus to NuBus

F = 25 ns, Nubus data hold time (Tcp–Tcw)

G = 2 ns, F651 min prop delay, Local bus to NuBus

For Tcp, Tcw, Ton, Tpd see RAM read timings

NIC REGISTER WRITE DATA SET UP TIME

Spec Times To Be Met

Write data set up time to end of MWR = 20 ns (NIC spec)

Adapter Card Times

Write data set up time to end of MWR =
Tcp(S1)+Tcw(S2)−D−E = 115 ns (95 ns to spare).

NIC REGISTER WRITE DATA HOLD TIME

Spec Times To Be Met

Write data hold time after MWR = 17 ns (NIC spec)

Adapter Card Times

Write data hold time after MWR = F+E−C = 17 ns
(C = 10 ns (D series PAL))

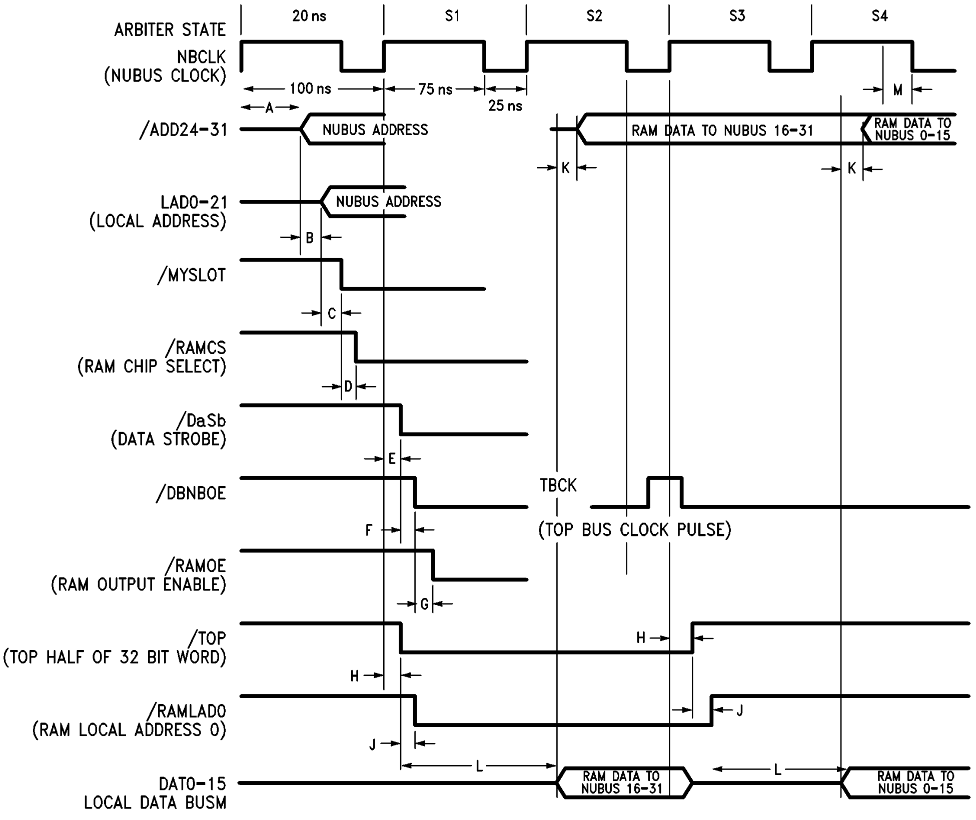
NIC REGISTER WRITE WIDTH FROM ACK

Spec Times To Be Met

Write width from ACK = 50 ns min (NIC spec)

Adapter Card Times

As RAMWET (MWR) is set before S4,
ACK to RAMWET > 75 ns (>25 ns to spare).



TL/F/10805-18

FIGURE A-4. NuBus to RAM Read Timing and Address Recognition Timing

K = 8 ns, F651 prop delay, Local data bus to NuBus

P = 15 ns, PAL16L8B prop delay, NbClk(S2) low to TBCK

Spec Times To Be Met

RAM Read data set up time to NuBus
(NbClk(S4)low) =
M = 21 ns (Tsu in NuBus spec)

RAM Read data set up time in TBCK = 7 ns
(F651 spec)

TBCK minimum pulse width = 7 ns
(F651 spec)

Adapter Card Times

NbClk(S1) to RAM Read data (bottom 16 bits) on local data bus = H+J+L = 130 ns

Set up time to TBCK = Tcp(S1+S2)−130 = 70 ns
(63 ns to spare)

NbClk(S3) to RAM Read data (32 bits) on NuBus = H+J+L+K = 138 ns

Adapter Card Times

Data buffer enabling signals are not cleared until after NbClk(S3) goes high

Hold time on Local data bus after TBCK > Tcp(S2)−Tcw(S2)−P = 10 ns
(> 10 ns to spare)

Data must be held on NuBus while NbClk(S4) is low and as buffer enabling signals are not cleared until the next rising edge of NbClk, NuBus hold time is met

Address Recognition

= 52 ns, Nubus address valid (Ton+2Tpd),
NbClk to NuBus

= 10 ns, F533 prop delay, Nubus to Local Address

= 11 ns, F521 prop delay, Local address to MySlot

TIMINGS

A = 15 ns, B series PAL prop delay, NbClk to DaSb or TOP

B = 15 ns, PAL16L8B prop delay, DaSb or TOP to NBDBOE

C = 15 ns, PAL16L8B prop delay, NBDBOE to RAMWET/B

D = 6.5 ns, F11 prop delay, RAMWET/B to RAMWE

E = 52 ns, NuBus data turn on time, NbClk(S1) to Local data bus

F = 8 ns, F651 prop delay, Local data bus to NuBus

RAM CE TO END OF WE

Spec Times To Be Met

RAM CE to end of RAMWE = 85 ns (RAM spec)

Adapter Card Times

RAMCE is driven during S0 (See RAM read timing)

RAMCE to end of RAMWE>Tcp(S1)+Tcw(S2) = 175 ns
(>90 ns to spare)

RAM WE PULSE WIDTH

Spec Times To Be Met

RAMWE pulse width = 70 ns min (RAM spec)

Adapter Card Times

RAMWE pulse width >Tcp(S1)+Tcw(S2)−A−B−C−D = 123.5 ns (53.5 ns to spare)

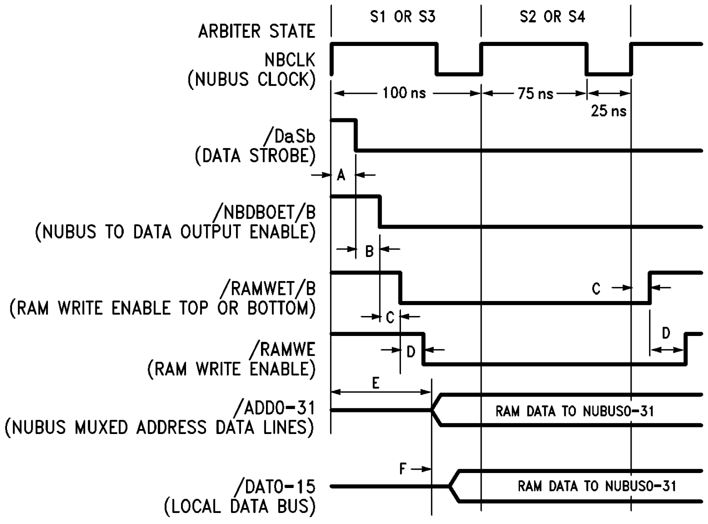
DATA VALID TO END OF WE

Spec Times To Be Met

Data valid to end of RAMWE = 50 ns min (RAM spec)

Adapter Card Times

Data valid to end of RAMWE = Tcp(S1)+Tcw(S2)−E−F = 115 ns (65 ns to spare)



TL/F/10805-19

FIGURE A-5. NuBus to RAM Write Timing

DATA HOLD AFTER END OF WE

Spec Times To Be Met

Data hold after end of RAMWE = 0 ns min (RAM spec)

Adapter Card Times

Data buffer enabling signals are not cleared unitl after NbClk(S3) goes high

Data hold after end of RAMWE >

Tcp(S2 or S4) + Tcw(S2 or S4) − C − D = 3.5 ns
(>3.5 ns to spare)

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.