Communication Systems Projects with LabVIEW

By:
Ed Doering

Communication Systems Projects with LabVIEW

By:
Ed Doering

Online:
< http://cnx.org/content/col10610/1.2/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: December 15, 2009
PDF generated: February 5, 2011
For copyright and attribution information for the modules contained in this collection, see p. 131.

Table of Contents

Introduction 1

1 Simulation and Visualization of Fundamental Concepts

1.1 Digital Communication System Simulation and Visualization 3
1.2 Intersymbol Interference (ISI) and the Eye Diagram 8
1.3 PAM Transmitter and Receiver Implementing Coherent Detection 15

2 Channel Coding and Error Control

2.1 Hamming Block Code Channel Encoder 27
2.2 Hamming Block Code Channel Decoder 30

3 FSK Demodulation

3.1 Caller ID Decoder 35

4 Bandpass Communications Over the Speaker-Air-Microphone Channel

4.1 Speaker-Air-Microphone (SAM) Channel Characterization 43
4.2 Binary ASK Transmitter 48
4.3 Texting Over the Speaker-Air-Microphone (SAM) Channel 55
4.4 Introduction to the LabVIEW Modulation Toolkit 63

5 SubVI Specifications

5.1 General-Purpose Utilities 69
5.2 Baseband Modulation and Pulse Amplitude Modulation (PAM) 85
5.3 Bandpass Modulation 94
5.4 Demodulation and Bitstream Regeneration 99
5.5 Hamming Block Coding 111
5.6 Speaker - Air - Microphone (SAM) Channel 119
5.7 Caller ID Decoder 124

Index 129
Attributions 131

Introduction[1]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[2] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 1

Introduction

Welcome to **Communication Systems Projects with LabVIEW**, a multimedia-enhanced series of projects that explore digital communication systems through LabVIEW simulations, visualizations, and implementations of practical systems.

Communication systems play an exciting role in our increasingly interconnected society. Digital communication systems form the heart of computer data networks, satellite communications, mobile telephones, and wireless hand-held devices. All electrical and computer engineering programs emphasize communication systems as part of the core curriculum.

Communication systems analysis and design requires a firm grasp of mathematical models, and demands mathematical skill with signals, systems, probability, and random variables. Insight and intuition, also important for the successful study of communication systems, do not always follow immediately from the mathematical presentations of traditional textbooks, however. Hands-on construction of real communication systems and interactive simulations that supplement the mathematics help to more quickly achieve insightful understanding of the myriad details involved in designing and optimizing a communications link for a given application.

Communication Systems Projects with LabVIEW features ten laboratory projects based on the LabVIEW graphical dataflow programming environment. LabVIEW offers an unparalleled way to directly translate communication system diagrams and mathematical descriptions into a LabVIEW program called a **block diagram**. The LabVIEW **front panel** GUI (graphical user interface) that emerges automatically as part of the programming activity enables real-time interaction with the communication system and visualization of the signals as waveforms, binary patterns, and text. This real-time interaction reveals connections, patterns, and often unexpected relationships – the basis of strong intuition and insight. Many of the projects emphasize listening to the signals as sound, further enhancing one's insight. Some of the laboratory projects simulate and visualize fundamental concepts such as baseband modulation, pulse shaping, intersymbol interference (ISI) and eye diagrams, while other projects result in fully-operational systems such as a Caller ID decoder and a text messaging system between a speaker and a microphone.

[1] This content is available online at <http://cnx.org/content/m18826/1.2/>.
[2] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>

Each project begins with an explanation of the background theory necessary to complete the project. These introductions feature narrated videos called **screencasts** that simulate a classroom lecture with a whiteboard visual aid. Continue by constructing a set of **subVIs** (LabVIEW reusable function blocks) according to precise specifications. Each subVI includes a screencast video that demonstrates the LabVIEW tool in operation to introduce and explain relevant LabVIEW programming techniques for the given subVI. Once the subVIs have been built and tested individually, assemble them into a working "top-level" VI (literally a **Virtual Instrument**, the name of a LabVIEW program). The project directions provide guidance through the complete development process, each step of the way.

To the Instructor

Communication Systems Projects with LabVIEW has been designed to augment existing communication systems laboratory projects, or to serve as the complete laboratory component of an introductory engineering communication systems course. Seven guiding principles motivate the design and organization of **Communication Systems Projects with LabVIEW**:

1. Build the concept for deepest learning – transforming a set of ideas into a working system clearly demonstrates a firm grasp of the concepts
2. Engage the senses to develop intuition and insight – seeing signals as waveform plots, listening to signals as sound, and changing the way signals are processed through virtual knobs and slider controls all work together to enhance understanding of the system under study
3. Interact with the system to develop understanding – LabVIEW offers an unparalleled means to automatically generate an interactive graphical user interface as part of the programming activity
4. Motivate with "real life" activities – many of the projects culminate in practical, working systems
5. Experience impairments – once the deleterious effects of real-world constraints such as finite channel bandwidth and noise become clear through direct experience, the standard methods to mitigate those effects can be appreciated more deeply
6. Integrate teaching and instruction with project activities – each project includes numerous narrated videos to explain concepts and to demonstrate task-specific LabVIEW programming techniques; each project also includes "textbook linkages" to many popular communication systems textbooks
7. Offer learning materials in a modular and open format – each project builds on a well-defined set of building blocks; the projects can easily be modified, extended, and tailored to specific needs

Each project requires four activities on the part of the student: (1) Study the introductory material that explains theory and concepts, (2) implement several subVIs as low-level building blocks, (3) assemble the subVIs into an application VI, and (4) interact with the finished VI to explore the theory and concepts. Constructing the subVIs helps students to develop skills with a wide variety of LabVIEW programming techniques, and also helps them to establish a firm grasp on the various LabVIEW data types. The subVIs are carefully specified around standard datatypes, i.e., Boolean array for bitstreams, waveform data type for "analog" signals; successful completion of the subVIs reduces the debugging effort required for the application VIs. Many of the subVIs are reused across multiple projects. The modularity of the projects – 10 projects total with a library of over 40 subVIs – allows the projects to be easily customized as necessary.

An instructor's manual and complete set of application VIs and subVIs is available; please contact the author for details.

Chapter 1

Simulation and Visualization of Fundamental Concepts

1.1 Digital Communication System Simulation and Visualization[1]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[2] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 1.1

NOTE: Visit LabVIEW Setup[3] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

1.1.1 Summary

Simulation and visualization enhance understanding of communication system behavior and performance. In this project, develop a simple model for a transmitter, channel, and receiver, and study the performance of the system in terms of bit error rate (BER). Channel errors are visualized as images and "auralized" as sound to further develop insight into the relationships between bit error rate and message length.

1.1.2 Objectives

1. Learn how to simulate a "black box" model of a binary communication system and to evaluate its performance
2. Develop an understanding of the relationships between bit error rate (BER) and message length

[1] This content is available online at <http://cnx.org/content/m18660/1.2/>.
[2] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[3] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

3. Develop a qualitative appreciation for BER and its impact on the received signal
4. Learn several ways to observe bitstreams

1.1.3 Deliverables

1. Summary write-up of your results
2. Hardcopy of all LabVIEW code that you develop (block diagrams and front panels)
3. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

1.1.4 Setup

1. LabVIEW 8.5 or later version
2. Computer soundcard
3. Speaker

1.1.5 Textbook Linkages

Refer to the following textbooks for additional background on the **binary symmetric channel** (also known as the **discrete memoryless channel**) used in this project; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 16
- Couch – Ch 7
- Haykin – Ch 10
- Lathi – Ch 15
- Proakis and Salehi (FCS) – Ch 12
- Proakis and Salehi (CSE) – Ch 9

1.1.6 Prerequisite Modules

If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[4] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

1.1.7 Introduction

Figure 1.1 illustrates a generic communication system (transmitter, channel, and receiver) and a comparator to compare the original source bitstream to the output bitstream and report bit errors.

[4] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

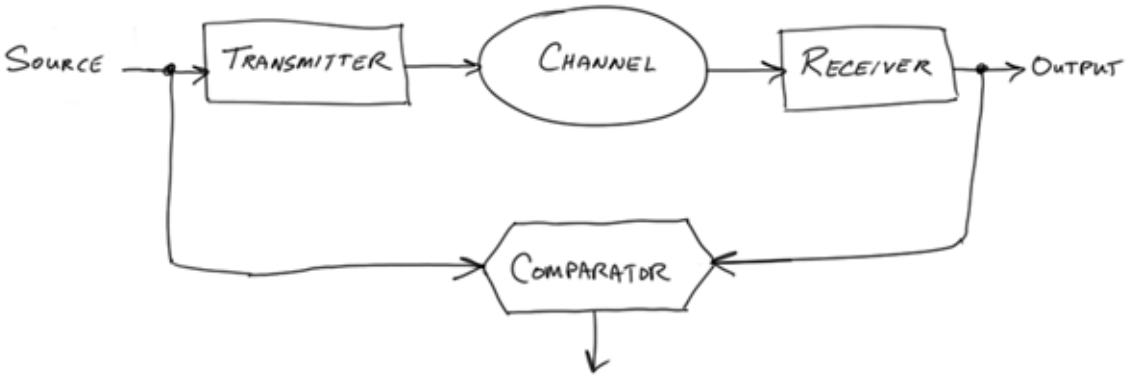


Figure 1.1: Generic communication system with comparator

This project implements Figure 1.1 at an elementary level:

1. The source is a bitstream with equiprobable 0s and 1s
2. The transmitter, channel, and receiver are lumped together as a single "black box," i.e., the internal details are hidden and only the source and received bitstreams are visible
3. The channel introduces errors according to the specified bit error rate (BER)
4. The comparator detects mismatches between the input and output bitstreams (bit errors) and reports measured BER, the ratio of the total number of actual bit errors to the total number of bits observed

1.1.8 Procedure

1.1.8.1 Build the subVIs

Build the subVIs listed below. You may already have some of these available from previous projects. Demonstrate that each of these subVIs works properly before continuing to the next part.

1. util_BitstreamFromRandom.vi (Section 5.1.1.1)
2. util_BinarySymmetricChannel.vi (Section 5.1.3.1)
3. util_MeasureBER.vi (Section 5.1.4.1)

1.1.8.2 Construct base system

1. Create the application VI `SystemOne.vi` pictured in Figure 1.2 by assembling the subVIs you built in the previous step. Use the default control and indicator styles for now. Expand the Boolean array indicators to show 20 elements (click on the outer frame of the indicator and drag either horizontally or vertically).
2. Try small values for `length` (say, 10 or 20) and various levels of bit error rate. Remember that the keyboard shortcut "Ctrl+R" runs the VI.
3. Submit a screenshot of your front panel with handwritten calculations that confirms the correct operation of the `measured BER` output.

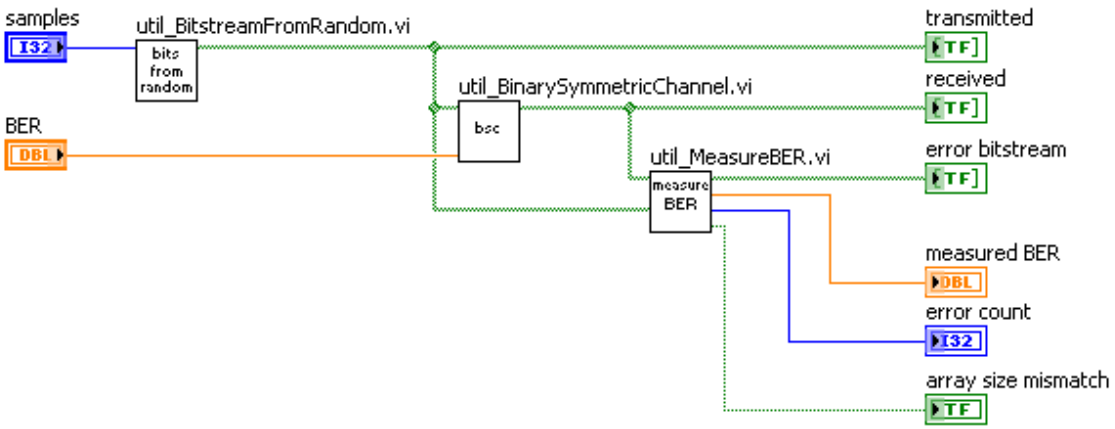


Figure 1.2: Generic communication system implemented as "SystemOne.vi"

1.1.8.3 Improve usability of front panel

The default numerical controls and indicators are useful to examine details such as the BER calculation. However, changing many of the controls and indicators to other forms greatly improves the usability of the front panel and facilitates greater interactivity.

View the Figure 1.3 screencast video to learn how to convert the numerical front panel controls to sliders and to visualize the bitstreams as waveforms. In addition, learn how to set the BER slider control to use logarithmic mapping to more conveniently select values over a wide range. Modify your application VI front panel accordingly to produce `SystemTwo.vi`.

Image not finished

Figure 1.3: [video] Improve the usability of the front panel controls

1.1.8.4 Investigate the relationship between BER and bitstream length

You have noticed by now that the measured bit error rate value is rarely (if ever) the same value as the specified BER of the channel. Moreover, the measured BER can change considerably from one run to the next. Continual operation of a VI greatly improves one's ability to see patterns and relationships emerge. In this section, modify the VI to run continually and observe the relationship between specified and measured BER value as a function of the bitstream length. View the Figure 1.4 screencast video to learn how to add a

while-loop structure to operate the system continually, and then modify your application VI accordingly to produce `SystemThree.vi`.

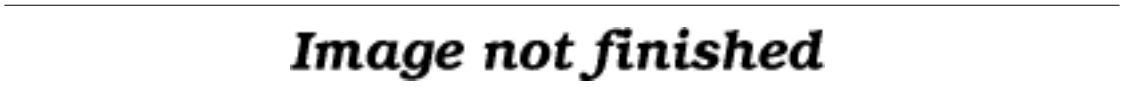


Figure 1.4: [video] Modify base system to run continually

Experiment with `SystemThree.vi`:

1. Estimate the average value (mean) of the measured BER as the specified channel BER varies over the range 0 to 1.
2. Estimate the variance of the measured BER as the bitstream length changes over the range 10 to 10,000 (a rough guess of the spread around the mean is adequate). Feel free to increase the "millisecond multiple" constant if the loop goes too fast to see the numerical displays.

Discuss your results:

1. How is the average value of the measured BER related to the specified channel BER?
2. How is the variance of the measured BER related to the bitstream length?

1.1.8.5 Visualize the bitstreams as images

Visualizing the error bitstream as 2-D image develops a qualitative feel for the impact of bit error rate on the data output of a binary communication system. That is, what value of BER corresponds to a "high quality" image transmission? Or, what value of BER makes the received image "poor quality"?

View the Figure 1.5 screencast video to learn how to reshape the error bitstream into a two-dimensional array suitable for display as a binary (2-level) image using the LabVIEW subVIs "Flatten Pixmap" and "Draw Flattened Pixmap." In addition, learn how to programmatically control the size of the front-panel image indicator using a "property node." Modify your application VI accordingly to produce `SystemFour.vi`.

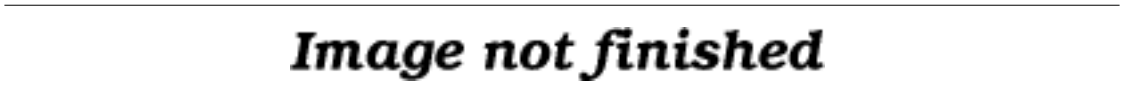


Figure 1.5: [video] Visualize the error bitstream as a binary image

Experiment with `SystemFour.vi` to study the relationship between BER and image size. To begin, set the bitstream length to 1,024 to produce a 32x32 image. Set the bit error rate to 0.0001. Describe the appearance of the error bitstream as an image, and state the relative "quality" of the image (remember that an ideal error image would always be uniformly black).

Now, gradually increase the bitstream length to 200,000 while watching the image. Would you still consider the image to be at the same quality level as before? What BER value do you need to obtain the same quality level you stated for the short bitstream length?

Explain why a specific BER value can be considered acceptable for some types of transmitted messages and not for others.

1.1.8.6 Listen to the error bitstream as sound

"Auralizing" the error bitstream as sound also develops your qualitative feel for bit error rate.

 Download and run bit_errors_as_sound.vi[5].

This application VI continually generates "the sound of silence" (bitstream of 0s) at the source with channel bit errors inserted according to the "BER" slider. Sound is generated in blocks (**frames**), and total errors within a frame are reported. The average bit errors per second is also reported. Note that the circular panel indicators use logarithmic mapping.

1. Change the bit error rate (BER) slider and listen to the bit errors.
2. Try different values of soundcard sampling frequency. Your soundcard may or may not support arbitrary values, but should definitely support CD-quality (44,100 Hz) and lower sampling rates of $\frac{44,100}{2^N}$, where N is an integer greater than zero.
3. If possible, use a media player to play music or speech through your soundcard while the VI is running. Determine the BER values you would associate with the following qualitative labels for the noise level's impact on the music signal: **none**, **just noticeable**, **tolerable**, **annoying**, and **overwhelming**. Tabulate your value/label pairs.
4. For BER=0.001, describe the character of the bit error click sound as a function of sampling frequency. Propose an explanation for the change in sound.
5. Explain the role of data rate (samples per second) on the impact of bit errors. In other words, does BER tell the whole story?

1.1.9 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1
2. Couch, Leon W. II, "Digital and Analog Communication Systems," 7th ed., Pearson Prentice Hall, 2007. ISBN-10: 0-13-142492-0
3. Haykin, Simon. "Communication Systems," 4th ed., Wiley, 2001. ISBN-10: 0-471-17869-1
4. Lathi, Bhagwandas P., "Modern Digital and Analog Communication Systems," 3rd ed., Oxford University Press, 1998. ISBN-10: 0-19-511009-9
5. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
6. Proakis, John G., and Masoud Salehi, "Communication Systems Engineering," 2nd ed., Pearson Prentice Hall, 2002. ISBN-10: 0-13-061793-8

1.2 Intersymbol Interference (ISI) and the Eye Diagram[6]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[7] module for tutorials and documentation that will help you:

continued on next page

[5] See the file at <http://cnx.org/content/m18660/latest/bit_errors_as_sound.vi>
[6] This content is available online at <http://cnx.org/content/m18662/1.1/>.

• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 1.2

NOTE: Visit LabVIEW Setup[8] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

1.2.1 Summary

This project studies intersymbol interference (ISI) in an intuitive way by using a LabVIEW VI to simulate a pulse transmitter, finite bandwidth channel, and received signaling waveform. Rectangular pulses are considered first to demonstrate the ISI problem, and then two alternative pulse shapes are explored as a way to minimize ISI. The eye diagram is also introduced in this project as a visual aid to present the time-domain signaling waveform to promote understanding of the ISI phenomenon.

1.2.2 Objectives

1. Understand the root cause of intersymbol interference (ISI)
2. Explain the significance of the sinc pulse and raised cosine pulse as a means to eliminate ISI
3. Understand the construction of an eye diagram
4. Be able to measure performance metrics (peak ISI, noise margin, jitter, and timing sensitivity) directly from the eye diagram

1.2.3 Deliverables

1. Summary write-up of your results
2. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

1.2.4 Setup

1. LabVIEW 8.5 or later version

1.2.5 Textbook Linkages

Refer to the following textbooks for additional background on the project activities of this module; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 11
- Couch – Ch 3

[7] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[8] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

- Haykin – Ch 4
- Haykin and Moher – Ch 6
- Proakis and Salehi (FCS) – Ch 9
- Proakis and Salehi (CSE) – Ch 8
- Stern and Mahmoud – Ch 4

1.2.6 Prerequisite Modules

If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[9] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

1.2.7 Introduction

Introductory digital logic courses present digital waveforms as essentially rectangular pulses. Indeed, the internal signals of a digital integrated circuit ideally exist at one of two voltage levels (high and low), with minimal time spent changing from one state to the other. Waveform displays from digital circuit simulators further emphasize the two-level rectangular shape of ideal digital signals.

Rectangular pulses are not ideal for transmission through communication links, however, since communication channels always restrict the bandwidth available between the transmitter and the receiver. Rectangular signaling pulses contain significant spectral energy across a wide frequency range due to the step-like transition between levels, and yet most communication systems do not allocate nearly enough bandwidth to faithfully transmit these abrupt changes. Passing a rectangular pulse through a limited-bandwidth channel distorts the pulse by "smearing" it – that is, the pulse stretches out in time. The transmitter sends a series of pulses to convey the message, therefore this time smearing causes **interference** between adjacent time slots (or **bit slots**). This **intersymbol interference** (abbreviated **ISI**) adds extraneous signal energy at the exact moments when a receiver's bit sampler decides whether a received bit should be called a logic "1" or a logic "0." ISI is not the same as additive random noise, but plays a similar role by reducing the **noise margin**, i.e., the room for error before the receiver's bit sampler makes an error.

This project studies intersymbol interference in an intuitive way by using a LabVIEW VI to simulate a pulse transmitter, finite bandwidth channel, and received signaling waveform. Rectangular pulses are considered first to demonstrate the ISI problem, and then two alternative pulse shapes are explored as a way to minimize ISI.

The **eye diagram** is also introduced in this project as a visual aid to present the time-domain signaling waveform to promote understanding of the ISI phenomenon. The eye diagram also reveals other key performance measures such as **noise margin**, **timing jitter**, and **timing sensitivity**.

1.2.8 ISI_and_EyeDiagram.vi

 Download the LabVIEW VI ISI_and_EyeDiagram.vi[10] , an interactive tool to study various pulse shapes as they pass through a band-limited channel.

Open the VI which starts running automatically, and then view the Figure 1.6 screencast video for a short orientation tour of the VI.

[9] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

[10] http://cnx.org/content/m18662/latest/ISI_and_EyeDiagram.vi

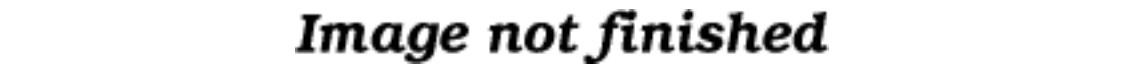


Figure 1.6: [video] Orientation tour of the "ISI_and_EyeDiagram.vi" LabVIEW VI

1.2.9 Rectangle Pulse Shape

Restore the front panel controls of "ISI_and_EyeDiagram.vi" to their default values by selecting "Edit | Reinitialize Values to Defaults."

Set the `symbols` control to 1 to produce a single rectangular pulse. The channel bandwidth should already be set to its maximum value of 0.49, which corresponds to essentially unlimited bandwidth. Note that this VI uses **normalized frequency**, therefore the sampling frequency corresponds to 1 and the Nyquist frequency is 0.5.

Compare the "transmitted waveform" and the "received waveform" plots in the lower-right front panel. How well does the received pulse match the transmitted pulse? Also, to what extent does the received pulse "spill out" of its designated time slot?

Decrease the channel bandwidth until you begin to observe noticeable pulse shape distortion. At what bandwidth does this occur?

Continue decreasing the channel bandwidth. What effects do you begin to observe?

Make a series of plots that show the progressive degradation of the rectangular pulse shape as the channel bandwidth is restricted. Right-click on the plot and choose "Export Simplified Image" to copy the graph to the clipboard for pasting into your report. Be sure to indicate the channel bandwidth for each plot.

1.2.10 Sinc Pulse Shape

Restore the front panel controls of "ISI_and_EyeDiagram.vi" to their default values.

Set the `symbols` control to 1 to produce a single pulse, and set the `bandwidth` control to 0.02. The received pulse should show noticeable distortion.

Now set the `pulse shape` control to "Sinc." How much distortion is evident at the receiver? How much lower can you restrict the bandwidth while still preserving the basic sinc waveform shape?

The sinc function's ability to maintain its basic shape through a restricted channel bandwidth is important, but its true significance extends beyond this fact, as explored in the next section.

1.2.11 Multiple Pulses

A transmitter converts a message, or sequence of bits, into a series of analog pulses to create the signaling waveform. A receiver recovers the bitstream by periodically sampling the signaling waveform and comparing the sample to a threshold value to decide "1" or "0." Sinc-shaped pulse do not interfere with adjacent bit slots, **provided** that the bit slots are sampled at the correct instant in time.

To see this, reinitialize the front panel control values to their default settings, choose the "Sinc" pulse shape, and choose 2 symbols. Look carefully at the transmitted and received pulses on the lower-left front panel plots. The white trace shows the first pulse in the sequence, while the red trace shows the second pulse in the sequence. The first pulse has an amplitude of +1, while the second pulse has an amplitude of -1, corresponding to a bit sequence "10"; refer to the `message bitstream` indicator to confirm that the first bit is `T` (green LED indicator active) and the second bit is `F` (inactive LED indicator).

The waveform plots on the lower-right front panel show the actual transmitted and received waveforms, which superimpose (i.e., add) the individual pulses together. The plots on the lower-left front panel illustrate the contribution of each individual pulse.

Look carefully at the time 450 samples, in which the second (red) pulse is at its most negative value. What is the value of the first (white) pulse at this instant? Hopefully you can see that it is **zero**, indicating that the first (white) pulse produces **zero interference** at the instant the second (red) pulse attains its maximum absolute value.

Now, increase the number of symbols to 3, and also to higher values. Study the waveforms to convince yourself that even though a single sinc pulse extends over many bit time intervals, the contribution of all adjacent pulses is always zero at the moment that a given sinc pulse attains its maximum absolute value. Therefore, the sinc pulse shape achieves zero ISI when properly sampled.

Make a series of plots from the "received pulses" waveform display and explain your understanding of the sinc pulse shape and its ability to achieve zero ISI.

Restrict the channel bandwidth to 0.02, and confirm that the sinc pulses remain essentially unchanged. Now, set the pulse shape to "Rectangular." Set the `symbols` control to 2 and study the lower-left front panel plots. Identify where the second (red) pulse attains its maximum absolute value. How much interference is present from the first (white) pulse?

Make a series of plots from the "received pulses" waveform display and explain your understanding of the rectangle pulse shape and its susceptibility to intersymbol interference.

1.2.12 Eye Diagram

Study the transmitted and received waveform plots on the lower-right front panel as you increase the number of symbols to 40, and try this for the two pulse shapes considered so far. Also try varying the channel bandwidth. The received signaling waveform is reasonably easy to understand for rectangle pulses, but is rather difficult to interpret when sinc pulses are used. For example, you should find that you can easily correlate the "message bitstream" sequence with the high and low regions in the received waveform when rectangular pulse shapes are transmitted; the correlation is more difficult when sinc pulses are used.

The **eye diagram** provides a powerful visualization tool to interpret the behavior of the received waveform regardless of the pulse shape. The Figure 1.7 screencast video continues the discussion by explaining the eye diagram plot on the upper-right front panel of "ISI_and_EyeDiagram." Follow along with video, matching the front panel controls of "ISI_and_EyeDiagram.vi" to those of the video.

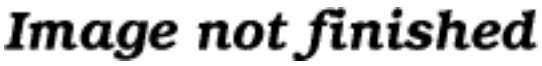


Figure 1.7: [video] Explanation of the eye diagram plot in the "ISI_and_EyeDiagram.vi" LabVIEW VI

The eye diagram reveals important performance metrics for a communication link, including **noise margin**, **ISI**, **timing sensitivity**, and **zero-crossing jitter**. In addition, the eye diagram shows the optimum sampling time for the receiver to regenerate a bitstream from the received signaling waveform. View the Figure 1.8 screencast video to learn how to measure these performance metrics and how to determine the optimum sampling time.

Image not finished

Figure 1.8: [video] Measuring noise margin, ISI, timing sensitivity, zero-crossing jitter, and optimum sampling time using an eye diagram

1.2.13 Eye Diagram Measurements

Figure 1.9 illustrates a generic eye pattern superimposed on a measured eye diagram plot and summarizes the definition of the various performance metrics discussed earlier. Use these definitions for the following measurements.

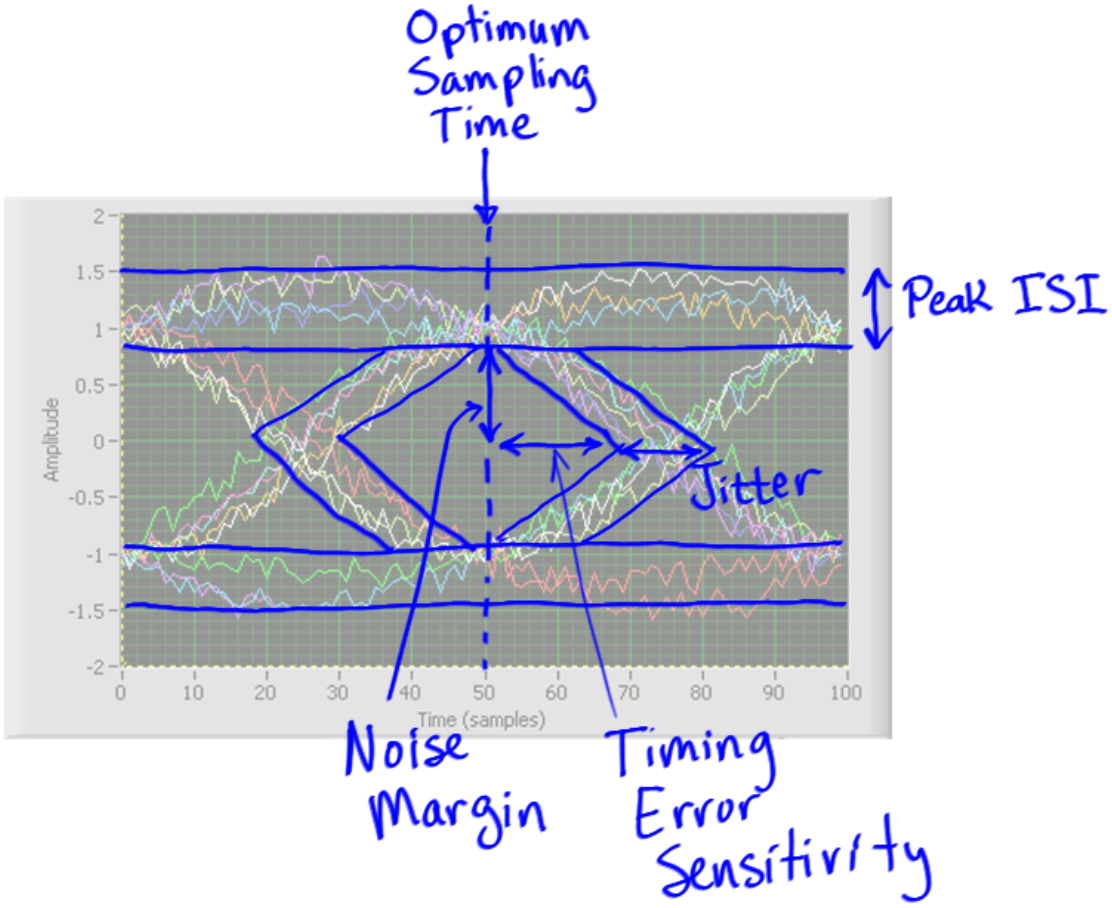


Figure 1.9: Generic eye pattern and definition of performance metrics

1.2.13.1 Rectangle Pulse

Restore the front panel controls of "ISI_and_EyeDiagram.vi" to their default values, and set the `symbols` control to 40. Vary the channel bandwidth and observe its effect on the eye diagram plot, and then set the channel bandwidth to 0.05. Increase the eye diagram `start time` to 245 samples to center the eye in the plot window.

Export the eye diagram plot to a piece of paper, and then use the eye diagram cursor as a tool to measure the following (show and label the relevant distances you measured on your hardcopy plot):

1. Optimum sampling time; report this as the number of samples from the nearest zero crossing
2. Peak ISI
3. Zero crossing jitter; report this as the maximum variation in time samples
4. Noise margin

1.2.13.2 Sinc Pulse

Ensure that the front panel controls of "ISI_and_EyeDiagram.vi" are the same as in the previous step, and then select the "Sinc" pulse shape. Adjust the eye diagram `start time` and `time span` to maximize the number of displayed bit intervals and also to avoid the initial startup transient that causes lines to cross through the center of the eye; also make adjustments to place the maximum eye opening at the center of the plot window.

As in the previous step, export the eye diagram plot to a piece of paper, and then use the eye diagram cursor as a tool to measure the following (show and label the relevant distances you measured on your hardcopy plot):

1. Peak ISI
2. ISI at the optimum sampling time
3. Zero crossing jitter; report this as the maximum variation in time samples
4. Noise margin
5. Timing error sensitivity; report this in terms of time samples

1.2.13.3 Raised Cosine Pulse

Keep the front panel controls of "ISI_and_EyeDiagram.vi" at the same settings you used for the previous "Sinc" pulse measurements, and then select the "Raised Cosine" pulse shape. You should expect to see the maximum eye opening remain centered in the eye diagram plot.

As in the previous steps, export the eye diagram plot to a piece of paper, and then use the eye diagram cursor as a tool to measure the the same five metrics as for the "Sinc" pulse. Show and label the relevant distances you measured on your hardcopy plot.

Compare your results for the raised cosine pulse and the sinc pulse. What appears to be advantageous about the raised cosine pulse shape?

See the video screencast in pam_RaisedCosinePulse.vi (Section 5.2.1.1) for more background about the raised cosine pulse, the most widely-used pulse shape in digital communication systems.

1.2.14 Noise

Add random channel noise to the received waveform by moving the `noise standard deviation` control away from zero. Note how the eye pattern begins to close as the noise level increases.

Report the noise standard deviation value at which the eye just begins to close completely for each of the three pulse shapes. Make hardcopy plots of the eye diagram for each value that you report.

1.2.15 Channel Filter

"ISI_and_EyeDiagram.vi" uses a 10th-order IIR lowpass filter to model the limited-bandwidth channel. This type of filter is fairly realistic, and produces **delay distortion**, an effect caused by the filter's nonlinear phase response. Delay distortion causes the various signal frequency components to arrive at the receiver at different times, and can severely distort the originally-transmitted pulse shape.

Engage the `FIR` pushbutton control to select a linear-phase FIR filter that does not introduce delay distortion.

1.2.16 Final Comments

The `seed` front-panel control in the "Pulses" section primes the random number generator that creates the message bitstream. Feel free to try other seed values to produce other bit sequences for the message.

"ISI_and_EyeDiagram.vi" uses two programming techniques that you may wish to investigate further, namely, an **event structure** and **property nodes**. Type Ctrl+E to show the block diagram window, and observe the event structure within the while-loop structure. The event structure only "fires" when a front-panel control value changes; the CPU does not do any work except to instantly respond to front-panel activity. The event structure makes the VI very responsive to user input, but does not burden the CPU as would a plain while-loop.

The property nodes allow front panel controls and indicators to be programmatically controlled. For example, changing the value of the `samples` front-panel control causes the upper limit of `start time` to automatically change to the same value.

1.2.17 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1
2. Couch, Leon W. II, "Digital and Analog Communication Systems," 7th ed., Pearson Prentice Hall, 2007. ISBN-10: 0-13-142492-0
3. Haykin, Simon. "Communication Systems," 4th ed., Wiley, 2001. ISBN-10: 0-471-17869-1
4. Haykin, Simon, and Michael Moher, "Introduction to Analog and Digital Communication Systems," 2nd ed., Wiley, 2007. ISBN-13: 978-0-471-43222-7
5. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
6. Proakis, John G., and Masoud Salehi, "Communication Systems Engineering," 2nd ed., Pearson Prentice Hall, 2002. ISBN-10: 0-13-061793-8
7. Stern, Harold P.E., and Samy A. Mahmoud, "Communication Systems," Pearson Prentice Hall, 2004. ISBN-10: 0-13-040268-0

1.3 PAM Transmitter and Receiver Implementing Coherent Detection[11]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[12] module for tutorials and documentation that will help you:

continued on next page

[11] This content is available online at <http://cnx.org/content/m18652/1.2/>.

• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 1.3

NOTE: Visit LabVIEW Setup[13] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

1.3.1 Summary

The integrate-and-dump detector is fundamental to coherent detection, the optimal receiver technique that minimizes bit error rate (BER) for a given signal-to-noise ratio Eb/No. In this project develop a pulse amplitude (PAM) transmitter based on a transmit filter to map a bitstream onto a signaling waveform (rectangular and Manchester pulse shapes), an additive white Gaussian noise (AWGN) channel, and a receiver that implements integrate-and-dump detection. All waveforms throughout the signal processing chain are presented as a stacked chart indicator with a speed control to permit generated waveforms to be studied slowly (i.e., the integrator output ramping up or down) or quickly to process long message bitstreams. Visualizing the critical system signals as waveforms facilitates exploration of the effects of specific values of BER and Eb/No, and promotes deeper understanding of coherent detection.

1.3.2 Objectives

1. Implement a binary pulse amplitude modulation (PAM) transmitter
2. Model an additive white Gaussian noise (AWGN) channel impairment with a random number generator
3. Implement a PAM receiver based on the integrate-and-dump form of coherent detection
4. Study the signal processing chain from the source message bitstream to the regenerated bitstream
5. Evaluate system performance using a plot of bit error rate (BER) vs. signal-to-noise ratio (Eb/No)
6. Learn how to use the LabVIEW point-by-point signal processing design pattern

1.3.3 Deliverables

1. Summary write-up of your results
2. Hardcopy of all LabVIEW code that you develop (block diagrams and front panels)
3. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

1.3.4 Setup

1. LabVIEW 8.5 or later version

[12] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[13] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

1.3.5 Textbook Linkages

Refer to the following textbooks for additional background on the project activities of this module; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 11
- Couch – Ch 6
- Haykin – Ch 5
- Haykin and Moher – Ch 10
- Lathi – Ch 14
- Proakis and Salehi (FCS) – Ch 8
- Proakis and Salehi (CSE) – Ch 7
- Stern and Mahmoud – Ch 4

1.3.6 Prerequisite Modules

If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[14] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

1.3.7 Introduction

Noise represents the most widely-known **channel impairment** in a communication system. No doubt you have heard "static" while listening to AM radio during a thunderstorm, soft hissing during a telephone conversation, and other types of background noise. Digital communication system noise causes errors in the recovered (regenerated) bit stream at the receiver.

In general, digital receivers rely on one of two detection techniques to regenerate the transmitted bit stream: **coherent detection** and **non-coherent detection**. "Coherent" means the receiver maintains synchronism with the transmitter, normally by using special subsystems that extract timing signals directly from the transmitted bit stream. Transmitting timing pulses in a separate channel is usually too expensive for long-haul comm links. The synchronizer establishes the precise beginning and ending of each bit interval. A synchronizer increases the receiver's cost and complexity, but also achieves the lowest bit error rate (BER) of the two techniques for a given signal-to-noise ratio (SNR). Incoherent detection, on the other hand, uses a lower-complexity approach to recover the bit stream, but does not perform as well in terms of BER. In this project the **correlation detector** scheme is studied in detail.

Figure 1.10 illustrates a generic communication system (transmitter, channel, and receiver) and a comparator to compare the original source bitstream to the output bitstream and report bit error.

[14] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

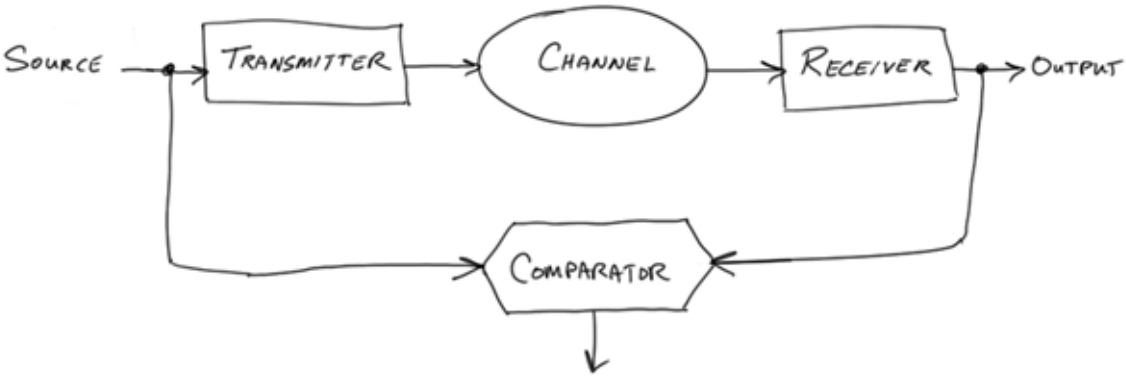


Figure 1.10: Generic communication system with comparator

This project implements Figure 1.10 at a moderately realistic level:

1. The source is a bitstream with equiprobable 0s and 1s.
2. The pulse amplitude modulation (PAM) transmitter maps the two source symbols onto rectangular signaling waveforms; these discrete-time waveforms approximate the true analog signaling waveforms that would be applied to a radio transmitter's modulator for wireless communications or a laser diode for fiber optic communications, for example.
3. The channel impairs the transmitted signal with additive white Gaussian noise (AWGN).
4. The receiver is a coherent receiver implemented as a correlation receiver.

1.3.8 PAM Transmitter

Figure 1.11 illustrates the detailed block diagram of the binary pulse amplitude modulation (PAM) transmitter.

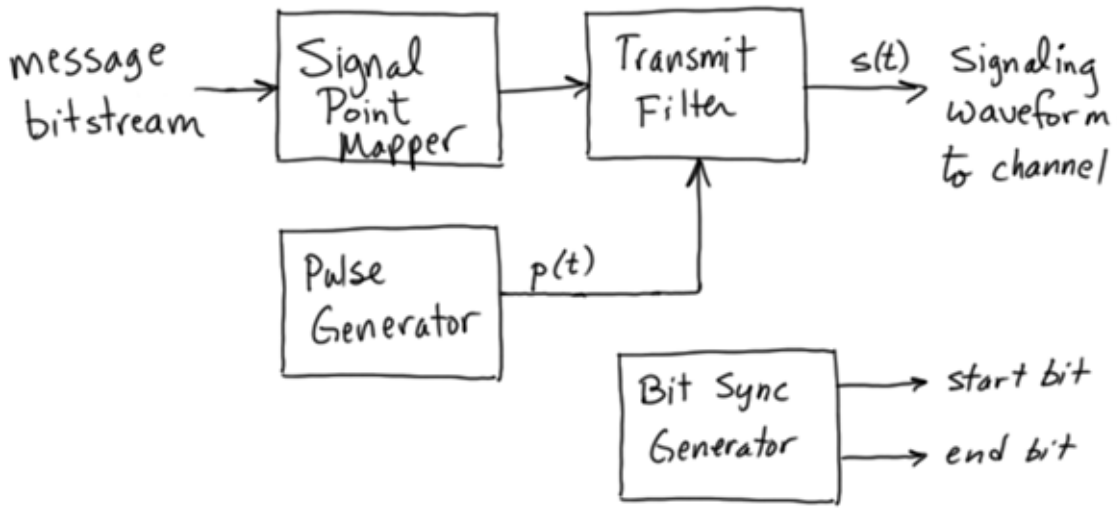


Figure 1.11: PAM transmitter block diagram

The bitstream 1 and 0 values map to the amplitudes $\sqrt{E_b/T_b}$ and $-\sqrt{E_b/T_b}$, where E_b is the energy per bit and T_b is the bit interval. The amplitudes are applied to the prototype pulse shape $p(t)$ with unit amplitude to generate a pair of signaling waveforms $s_1(t) = \sqrt{E_b/T_b}p(t)$ and $s_0(t) = -\sqrt{E_b/T_b}p(t)$. This signaling scheme is called **binary antipodal signaling**.

Many different pulse shapes are used in practice, based on the application. This project considers two specific pulse shapes, namely, **rectangular** and **Manchester**. Both of the pulse shapes are of the **polar NRZ** (non return to zero) type. The Figure 1.12 screencast video continues the discussion by describing these two pulse shapes in more detail.

Image not finished

Figure 1.12: [video] Rectangular and Manchester polar NRZ pulse shapes

The **signal point mapper** and **pulse generator** of Figure 1.11 describe the desired amplitudes and pulse shape, while the transmit filter converts the message bitstream into a sequence of signaling waveforms. The transmit filter is an FIR filter driven by an impulse train derived from the signal point mapper amplitudes; the FIR filter coefficients are the pulse shape values. Refer to the screencast video in pam_TransmitFilter.vi (Section 5.2.2.2) for full implementation details.

The **bit sync generator** block sends pulses to the receiver to indicate the beginning and ending a bit interval.

1.3.9 AWGN Channel

Additive white Gaussian noise (**AWGN**) impairs signals as they pass through an electromagnetic medium, including the electronics in the transmitter and receiver. Adding the output of a Gaussian random number generator to the transmitted signal simulates the AWGN impairment of a real channel.

The degree of signal impairment is reported as a ratio of signal strength to noise ratio (SNR). Digital communication systems define signal-to-noise ratio as E_b/N_0 (pronounced "ebb know"), where E_b is the energy per bit and N_0 is twice the power spectral density of thermal noise at room temperature. The ratio is dimensionless, and is normally reported in decibels. Refer to the screencast video in util_AWGNchannel_PtByPt.vi (Section 5.1.3.2) to learn how to convert a specified E_b/N_0 ratio into the standard deviation parameter of a Gaussian random number generator.

1.3.10 Coherent Detection Receiver

Figure 1.13 shows the block diagram of a receiver that implements coherent detection with a correlator, also called an **integrate-and-dump** detector.

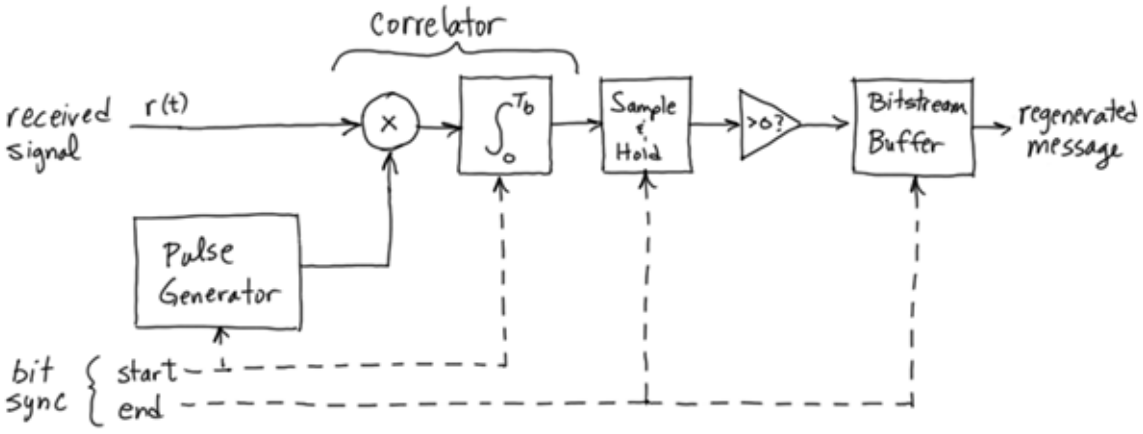


Figure 1.13: Block diagram of PAM receiver based on coherent detection

The **correlator** multiplies the received signal by the same pulse shape used by the transmitter, and then integrates this product signal over one bit interval. The correlator output is sampled at the end of the bit interval by the **sample-and-hold** device, and then compared to the zero threshold. If the sampled correlator output is greater than the threshold, the received bit is declared a 1, otherwise the received bit is declared a 0. The integrator is reset to zero at the beginning of each bit interval.

The receiver requires precise synchronization with the transmitter in two respects: the correlator must multiply the received signal by the pulse shape in the same time location, and the integrator must be reset precisely at the beginning of a new bit interval. These requirements are easy to achieve within a simulation, since the transmitter can send pulses to signal the beginning and ending of the bit interval. In a real system, synchronization subsystems extract these timing pulses directly from the received signal, adding cost and complexity to the receiver.

Digital communication system performance in the face of AWGN channel impairment is measured in terms of bit error rate (BER) for a given signal quality E_b/N_0. Coherent detection with binary antipodal

signaling as used in this project has a theoretical BER of

$$BER = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{1.1}$$

where the Q-function $Q(x)$ describes the area under a zero-mean unit-variance Gaussian probability density function from x to positive infinity, i.e., the area under the positive tail of the Gaussian. Equation (1.1) serves as the benchmark for the simulated BER of the system constructed in this project.

1.3.11 Procedure

1.3.11.1 Build the subVIs

Build the subVIs listed below. You may already have some of these available from previous projects. Demonstrate that each of these subVIs works properly before continuing to the next part.

1. pam_SignalPointMapper.vi (Section 5.2.2.1)
2. pam_RectanglePulse.vi (Section 5.2.1.2)
3. pam_ManchesterPulse.vi (Section 5.2.1.3)
4. pam_TransmitFilter.vi (Section 5.2.2.2)
5. pam_TransmitSync.vi (Section 5.2.2.3)
6. regen_Correlator.vi (Section 5.4.3.1)
7. regen_SampleHold.vi (Section 5.4.4.1)
8. regen_BitstreamBuffer.vi (Section 5.4.4.3)
9. util_BitstreamFromRandom.vi (Section 5.1.1.1)
10. util_AWGNchannel_PtByPt.vi (Section 5.1.3.2)
11. util_MeasureBER.vi (Section 5.1.4.1)
12. util_Qfunction.vi (Section 5.1.5.3)

1.3.11.2 Build the transmitter

Assemble the transmitter by translating Figure 1.11 into a LabVIEW application VI called `Transmitter.vi`. Create front panel controls with default values as follows:

1. `message length` – I32 – 5 bits
2. `Eb, energy per bit interval [J/bit]` – DBL – 1.0
3. `Tb, bit interval [s]` – DBL – 1.0
4. `pulse shape` – enumerated data type – Rectangle
5. `fs, sampling frequency [Hz]` – DBL – 10.0

Use an enumerated front-panel control to select the pulse shape, and a case structure on the block diagram to select the desired pulse shape. The Figure 1.14 screencast video explains how to configure the front-panel control and how to use the control as the selector on the case structure.

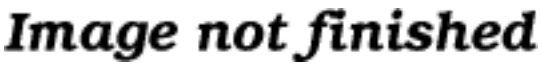


Figure 1.14: [video] Enumerated control as a case selector

Plot the transmitted signal waveform for both the polar NRZ and Manchester pulse shapes, and confirm that the signal waveform amplitude and samples per bit interval respond correctly to various selections for sampling frequency, bit interval, energy per bit, and message length.

1.3.11.3 Build the channel and receiver

Visualizing the signal processing chain through the receiver is the main objective of this section. The **stacked chart** waveform indicator works best because it allows timescale adjustments while maintaining synchronism among all of the displayed signals. The stacked chart emulates a strip chart recorder or oscilloscope display, and is designed to accumulate and display one sample point generated each pass through a repetitive structure such as a for-loop or while-loop. The Figure 1.15 screencast video introduces the stacked chart waveform indicator, explains how to display multiple signals, and describes how to interact with the indicator to view selected time intervals.

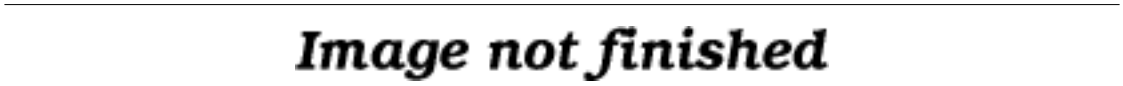


Figure 1.15: [video] Display multiple synchronized signals on stacked chart

Copy `Transmitter.vi` to a new file called `TransmitterReceiver.vi`. Remove the waveform graph indicator. Add the AWGN channel and coherent receiver to this VI by translating the Figure 1.13 receiver block diagram. Make a front panel control for the channel Eb/No. Embed the entire channel and receiver into a for-loop structure. Include "Programming | Timing | Wait Until Next ms Multiple" inside the for-loop and create a front-panel control called `loop delay [ms]` to adjust the delay. Place the control inside the for-loop structure so that the processing rate of the receiver can be easily adjusted. Display the following signals on a stacked chart:

1. transmitted signal, s(t)
2. received signal, s(t)+n(t)
3. transmitter bit interval start pulse
4. transmitter bit interval end pulse
5. correlator output
6. sample-and-hold output
7. comparator output

Include a BER measurement (with util_MeasureBER.vi (Section 5.1.4.1)) to compare the transmitted and received message bitstreams.

Include Boolean indicators for the transmitted bitstream, the regenerated (received) bitstream, and the error bitstream.

Reserve space for the BER vs. Eb/No plot to be added later.

Figure 1.16 illustrates a suggested front-panel layout for `TransmitterReceiver.vi`.

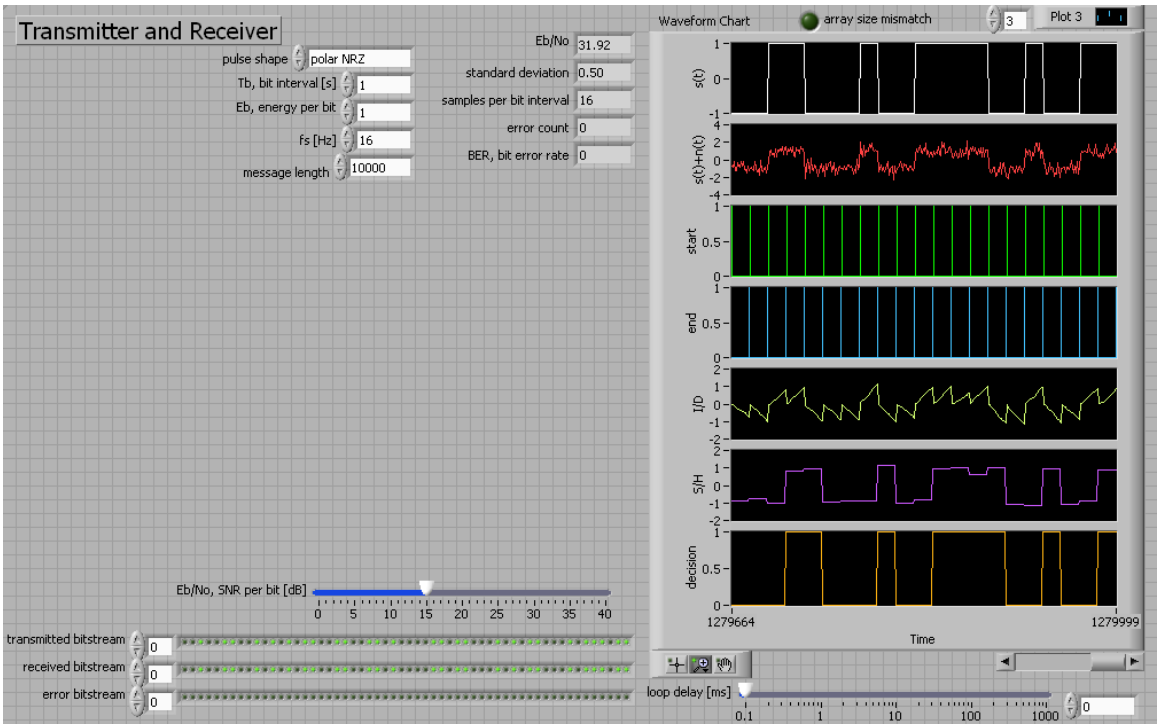


Figure 1.16: Suggested front-panel layout for `TransmitterReceiver.vi`

Debug the combined transmitter and receiver with a high value of Eb/No such as 40dB to effectively eliminate channel noise. Ensure that the received message is the same as the transmitted message. The BER should remain zero or nearly so, even for relatively long messages.

To confirm that the AWGN channel works properly, set the front panel controls to these exact values:

1. `message length` = 10,000 bits
2. `Eb` = 1 J/bit
3. `Tb` = 1 s
4. `Eb/No` = 0 dB
5. `pulse shape` = Polar NRZ
6. `fs` = 32 Hz
7. `loop delay` = 0 ms

The BER should be very close to 0.079 each time the VI is run; the theoretical value is 0.07865.

1.3.11.4 Experiment with the transmitter, channel, and receiver

Set Eb/No to 40dB to generate a clean transmitter signal at the receiver, and study the correlator output for the polar NRZ pulse shape. Describe the effect of the "integrate-and-dump" operation as applied to the transmitted signal. Use a loop delay of in the range 10 to 50 ms to observe the waveform unfold slowly.

Switch to the Manchester pulse shape, and study the correlator output again. The correlator output should look exactly the same as observed for the polar NRZ pulse shape, even though the two pulse shapes are significantly different. Explain why.

Try message lengths from 10 bits to 10,000 bits and higher. Confirm that BER is zero or nearly so for each message.

Set the message length to 10 bits. Gradually decrease Eb/No and observe the effect on the receiver signals. What level of Eb/No causes the received signal to look noisy and yet still be intelligible to the eye? What level of Eb/No causes the received signal to look essentially unusable, and yet the BER remains small (say, 1 percent)? From these observations, explain how coherent detection is able to recover a very useable signal from such a noisy input.

1.3.11.5 BER vs. Eb/No performance measure

Add a structure to retain the Eb/No and measured BER in arrays at the end of each simulation run. Plot BER vs. Eb/No as a scatter plot over the domain Eb/No = 0 dB to 10 dB. Include a Boolean control to reset the plot by reinitializing the arrays. See the Figure 1.17 screencast for implementation details.

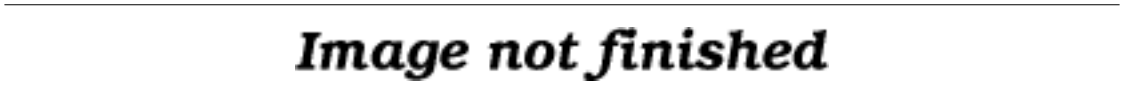


Figure 1.17: [video] Retain values across multiple runs of a VI and visualize values as a scatter plot

Engage the "Run Continuously" mode (the circulating arrows icon next to the "Run" button) to continually add points to the plot. Vary Eb/No from 0 dB to 10 dB for a message length of 100 bits. Make note of the spread of BER values for a particular Eb/No value, as well as the minimum BER. Increase the message length to 1,000 bits and then clear the accumulated plot points. Observe the BER spread and minimum value as Eb/No varies over the same range.

Repeat the previous step for a message length of 10,000 bits. Consider your results for various message lengths, and then explain the relationship between the minimum recorded BER and message length. In addition, describe the relationship between the spread (variance) of BER values as a function of Eb/No. Explain why the spread decreases as the noise level increases, or equivalently, as Eb/No decreases. :w Add the theoretical BER vs. Eb/No curve for binary antipodal signaling as a solid trace to the scatter plot; refer to the Figure 1.18 screencast video to learn how to overlay two plots. How well does the simulated scatter plot match theory? What is the critical parameter that causes the measured BER to more closely follow the theoretical value for higher-quality signals, i.e., when Eb/No is closer to 10 dB? What penalty is incurred to achieve a more accurate estimate of BER for higher quality signals?

Include representative plots in your report.

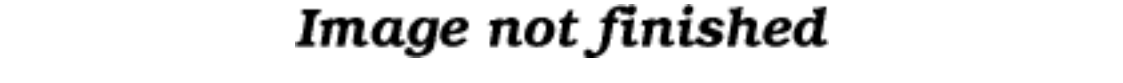


Figure 1.18: [video] Overlay two plots

1.3.12 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1

2. Couch, Leon W. II, "Digital and Analog Communication Systems," 7th ed., Pearson Prentice Hall, 2007. ISBN-10: 0-13-142492-0
3. Haykin, Simon. "Communication Systems," 4th ed., Wiley, 2001. ISBN-10: 0-471-17869-1
4. Haykin, Simon, and Michael Moher, "Introduction to Analog and Digital Communication Systems," 2nd ed., Wiley, 2007. ISBN-13: 978-0-471-43222-7
5. Lathi, Bhagwandas P., "Modern Digital and Analog Communication Systems," 3rd ed., Oxford University Press, 1998. ISBN-10: 0-19-511009-9
6. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
7. Proakis, John G., and Masoud Salehi, "Communication Systems Engineering," 2nd ed., Pearson Prentice Hall, 2002. ISBN-10: 0-13-061793-8
8. Stern, Harold P.E., and Samy A. Mahmoud, "Communication Systems," Pearson Prentice Hall, 2004. ISBN-10: 0-13-040268-0

Chapter 2

Channel Coding and Error Control

2.1 Hamming Block Code Channel Encoder[1]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[2] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 2.1

NOTE: Visit LabVIEW Setup[3] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

2.1.1 Summary

Channel encoding inserts additional information into a transmitted bitstream to facilitate **error detection** and **error correction** at the receiver. **Block coding** breaks up a bitstream into words of length k bits and appends check bits to form a **codeword** of length n bits. A corresponding **channel decoder** examines the complete codeword, and detects and even corrects certain types of erroneous bits caused by the channel.

In this project, develop a channel encoder using a special class of block code called a Hamming code. In a follow-on project, develop a companion channel decoder, and then evaluate the performance of the complete encoder/decoder system.

2.1.2 Objectives

1. Develop an (n,k) Hamming block code channel encoder
2. Examine the behavior of the encoded bitstream before and after passing through a binary symmetric channel (BSC)

[1] This content is available online at <http://cnx.org/content/m18663/1.1/>.
[2] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[3] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

3. Learn how to use LabVIEW matrix-oriented subVIs

2.1.3 Deliverables

1. Summary write-up of your results
2. Hardcopy of all LabVIEW code that you develop (block diagrams and front panels)
3. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

2.1.4 Setup

1. LabVIEW 8.5 or later version

2.1.5 Textbook Linkages

Refer to the following textbooks for additional background on the project activities of this module; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 13 (basis for notation used in this module)
- Haykin – Ch 10
- Lathi – Ch 16
- Proakis and Salehi (FCS) – Ch 13
- Proakis and Salehi (CSE) – Ch 9
- Stern and Mahmoud – Ch 10

2.1.6 Prerequisite Modules

If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[4] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

2.1.7 Introduction

Error control coding describes a class of techniques that prepare a digital message bitstream to pass through a noisy channel so that the receiver can detect transmission errors and in some cases correct these errors.

The Figure 2.1 screencast video introduces error control coding, including visualization of codewords, **Hamming distance**, **minimum distance** of a code, and error detection and correction power of a code.

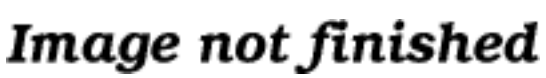


Figure 2.1: [video] Error control coding basic concepts

[4] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

(n,k) block codes break up a message bitstream into blocks of k bits and insert additional blocks of **checkbits**. The checkbit information permits a receiver to diagnose the received bitstream for errors, and to correct some types of errors automatically.

The Figure 2.2 screencast video introduces (n,k) block codes, **code rate**, the special case of **linear block codes**, and illustrates the trade-off between code rate and error control power.

Figure 2.2: [video] (n,k) block coding basic concepts

(n,k) Hamming block codes represent a popular type of block code. The Figure 2.3 screencast video introduces the (n,k) Hamming block code, explains how to construct the **generator matrix** to transform message blocks into codewords rate, and presents a detailed example to illustrate the encoding process.

Figure 2.3: [video] (n,k) Hamming code construction rules and example

2.1.8 Procedure

2.1.8.1 Manual calculations

Work through the Hamming code construction process by hand to lay a good foundation for developing a correct and understandable computer implementation. Write up this work on a separate page.

1. Construct two distinct (7,4) Hamming code "G" matrices.
2. For each "G" matrix, calculate the codewords that emerge from the following message words: 0000, 1010, and 1111.

2.1.8.2 SubVI construction

Build the subVIs listed below. You may already have some of these available from previous projects.

Demonstrate that each of these subVIs works properly before continuing to the next part.

1. hamming_HammingCodeParameters.vi (Section 5.5.3)
2. hamming_GeneratorMatrix.vi (Section 5.5.2)
3. hamming_Mod2MatrixMultiply.vi (Section 5.5.4)
4. util_BitstreamFromRandom.vi (Section 5.1.1.1)
5. util_BitsToWords.vi (Section 5.1.2.1)
6. util_WordsToBits.vi (Section 5.1.2.2)
7. util_BinarySymmetricChannel.vi (Section 5.1.3.1)

2.1.8.3 Hamming block code channel encoder

Review again the background theory presented earlier for the Hamming block code channel encoder, and then assemble your subVIs into a top-level application VI that creates a message bitstream, encodes the bitstream using Hamming coding, passes the bitstream through a noisy channel (the binary symmetric channel), and displays selected results of the channel encoding process.

 Download the LabVIEW VI Front_Panel_Indicators.vi[5] . This VI contains pre-formatted front panel indicators suitable for convenient display of binary values.

Debug your application until it works properly. Include a front-panel screenshot with hand-written annotations that demonstrates correct operation of your encoder.

2.1.9 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1
2. Haykin, Simon. "Communication Systems," 4th ed., Wiley, 2001. ISBN-10: 0-471-17869-1
3. Lathi, Bhagwandas P., "Modern Digital and Analog Communication Systems," 3rd ed., Oxford University Press, 1998. ISBN-10: 0-19-511009-9
4. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
5. Proakis, John G., and Masoud Salehi, "Communication Systems Engineering," 2nd ed., Pearson Prentice Hall, 2002. ISBN-10: 0-13-061793-8
6. Stern, Harold P.E., and Samy A. Mahmoud, "Communication Systems," Pearson Prentice Hall, 2004. ISBN-10: 0-13-040268-0

2.2 Hamming Block Code Channel Decoder[6]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[7] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 2.2

NOTE: Visit LabVIEW Setup[8] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[5] http://cnx.org/content/m18663/latest/Front_Panel_Indicators.vi
[6] This content is available online at <http://cnx.org/content/m18665/1.2/>.
[7] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[8] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

2.2.1 Summary

Channel encoding inserts additional information into a transmitted bit stream to facilitate **error detection** and **error correction** at the receiver. **Block coding** breaks up a bit stream into words of length k bits and appends check bits to form a **codeword** of length n bits. A corresponding **channel decoder** examines the complete codeword, and detects and even corrects certain types of erroneous bits caused by the channel.

In the prerequisite project Hamming Block Code Channel Encoder (Section 2.1) you developed a channel encoder using a special class of block code called a Hamming code. In this project, develop the companion channel decoder, and then evaluate the performance of the complete encoder/decoder system.

2.2.2 Objectives

1. Develop an (n,k) Hamming block code channel decoder capable of error detection and correction
2. Examine the behavior of the encoded bitstream before and after passing through the decoder
3. Evaluate the performance of the complete encoder/decoder system

2.2.3 Deliverables

1. Summary write-up of your results
2. Hardcopy of all LabVIEW code that you develop (block diagrams and front panels)
3. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

2.2.4 Setup

1. LabVIEW 8.5 or later version

2.2.5 Textbook Linkages

Refer to the following textbooks for additional background on the project activities of this module; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 13 (basis for notation used in this module)
- Haykin – Ch 10
- Lathi – Ch 16
- Proakis and Salehi (FCS) – Ch 13
- Proakis and Salehi (CSE) – Ch 9
- Stern and Mahmoud – Ch 10

2.2.6 Prerequisite Modules

If you have not done so already, please complete the prerequisite module Hamming Block Code Channel Encoder (Section 2.1). If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[9] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

[9] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

2.2.7 Introduction

Error control coding describes a class of techniques that prepare a digital message bitstream to pass through a noisy channel so that the receiver can detect and in some cases correct transmission errors. The prerequisite project Hamming Block Code Channel Encoder (Section 2.1) describes how to create a specific type of channel encoder based on the **(n,k) Hamming code**. The codeword length "n" and message length "k" are specific values calculated from the user-defined number of checkbits "q". As discussed in the prerequisite module, the code rate of the Hamming code approaches 1 (100% efficiency) as "q" increases, but the minimum Hamming distance "dmin" is fixed at 3. Therefore, the Hamming code can detect up to two bit errors in a received codeword, and can correct up to one bit error.

The **channel decoder**, a subsystem of the receiver, serves as a complement to the channel encoder in the transmitter. The channel decoder examines each received codeword, indicates detectable errors, fixes correctable errors, and extracts the message. Not all types of errors are detectable nor correctable, therefore the channel decoder can certainly emit garbled messages. Fortunately the channel noise must be rather severe before this becomes a problem.

The channel decoder developed in this project is called a **table lookup syndrome decoder**. View the Figure 2.4 screencast video to learn how to calculate the **syndrome** of a codeword, how to develop a **lookup table** of most-likely error patterns indexed by syndrome value, and how to use these results as the basic components of a channel decoder capable of detecting and correcting some types of error patterns.

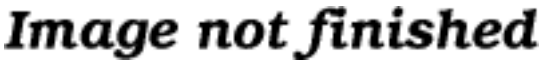


Figure 2.4: [video] Table lookup syndrome channel decoder for Hamming block code

2.2.8 Procedure

2.2.8.1 Manual calculations

Work through the syndrome calculation process by hand to lay a good foundation for developing a correct and understandable computer implementation. Write up this work on a separate page.

The end of the Figure 2.4 screencast video presents an example of a specific Hamming code generator matrix "G", a specific message vector "M" and associated codeword vector "X", and three received versions of the same transmitted codeword with varying severity of bit errors.

1. Determine the parity check matrix "HT" (the transpose of the matrix "H") that corresponds to the generator matrix "G".
2. Write the three received codeword vectors.
3. Calculate the syndrome for each of the three received codewords. Remember to use modulo-2 arithmetic for the matrix calculations.
4. Discuss your results in terms of the potential to detect and correct errors for each of the three received codewords based on their calculated syndromes.

2.2.8.2 SubVI construction

Build the subVIs listed below. You may already have some of these available from previous projects.

Demonstrate that each of these subVIs works properly before continuing to the next part.

1. hamming_ParityCheckMatrix.vi (Section 5.5.5)
2. hamming_SyndromeTable.vi (Section 5.5.6)
3. hamming_DetectorCorrector.vi (Section 5.5.1)
4. util_BitstreamFromText.vi (Section 5.1.1.2)
5. util_BitstreamToText.vi (Section 5.1.2.3)

2.2.8.3 Hamming block code channel decoder

Use your top-level application VI from the prerequisite channel encoder project as a starting point for this project.

Review again the background theory presented earlier for the Hamming block code channel decoder, then extend the top-level application VI to decode the output of the channel. Follow the block diagram described near the end of the Figure 2.4 screencast video.

Display Boolean array front-panel indicators for the following values:

- message – original message words
- encoded message – message words with appended checkbits (transmitted codewords)
- received message – received codewords after passing through noisy channel
- pre-decoding errors – bit error locations in received codewords
- corrected message – received codewords with error correction applied
- post-decoding errors – bit error locations in corrected codewords
- error detected (1-D array) – error detected (non-zero syndrome)

2.2.8.4 Combined channel encoder/decoder performance

1. Generate 50 words, and begin with 3 checkbits. Run the VI repeatedly and observe the channel decoder output indicators. What bit error rate tends to limit the received codeword errors to single-bit errors?
2. Increase to 4 checkbits, then to 5 checkbits, and so on while holding the bit error rate fixed. Recalling the positive effect of increasing the number of checkbits (increased code rate), what appears to be the negative effect of an increased number of checkbits? Explain.
3. Return to 3 checkbits. Run the VI until you observe a two-bit error in a received codeword. Does the "error detected" indicator work properly? How about the corrected codeword? Explain these results.
4. Set the number of checkbits to 2 and run the VI several times. What is another name (hopefully familiar to you) for this code?

2.2.8.5 Text messaging

Replace the random number generator in the transmit section with the text data source util_BitstreamFromText.vi (Section 5.1.1.2). Use the companion subVI util_BitstreamToText.vi (Section 5.1.2.3) to display the receiver output. Experiment with short messages and long messages, making sure that the intermediate Boolean displays make sense.

Experiment with intelligibility in the received message as a function of bit error rate (BER). Determine specific BER values you associate with the following qualitative labels: **excellent**, **good**, **barely acceptable**, and **unintelligible**.

2.2.9 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1
2. Haykin, Simon. "Communication Systems," 4th ed., Wiley, 2001. ISBN-10: 0-471-17869-1

3. Lathi, Bhagwandas P., "Modern Digital and Analog Communication Systems," 3rd ed., Oxford University Press, 1998. ISBN-10: 0-19-511009-9
4. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
5. Proakis, John G., and Masoud Salehi, "Communication Systems Engineering," 2nd ed., Pearson Prentice Hall, 2002. ISBN-10: 0-13-061793-8
6. Stern, Harold P.E., and Samy A. Mahmoud, "Communication Systems," Pearson Prentice Hall, 2004. ISBN-10: 0-13-040268-0

Chapter 3

FSK Demodulation

3.1 Caller ID Decoder[1]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[2] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 3.1

NOTE: Visit LabVIEW Setup[3] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

3.1.1 Summary

The telephone company's "Caller ID" service provides the calling party's directory information as well as the time and date of the call as an FSK (frequency shift keying) signal between the first and second rings of a telephone call. In this project, develop a complete Caller ID decoder that analyzes an audio recording of the FSK signal to extract the directory and date information for display.

3.1.2 Objectives

1. Describe the Caller ID standard at the signal level
2. Express a set of Caller ID information as a series of message bytes
3. Decode by hand a Caller ID data block bitstream
4. Implement a complete Caller ID decoder application in LabVIEW

[1] This content is available online at <http://cnx.org/content/m18708/1.1/>.
[2] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[3] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

3.1.3 Deliverables

1. Summary write-up of your results
2. Hardcopy of all LabVIEW code that you develop (block diagrams and front panels)
3. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

3.1.4 Setup

1. LabVIEW 8.5 or later version
2. Modulation Toolkit 4.0 or later version
3. Computer soundcard
4. Speaker

3.1.5 Textbook Linkages

Refer to the following textbooks for additional background on the project activities of this module; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 14
- Couch – Ch 5
- Haykin and Moher – Ch 7
- Lathi – Ch 13
- Proakis and Salehi (FCS) – Ch 10
- Stern and Mahmoud – Ch 5
- Wheeler – Ch 14 (an excellent reference, provides much detail about Caller ID)

3.1.6 Prerequisite Modules

If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[4] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

3.1.7 Introduction

You are no doubt familiar with **Caller ID**, the telephone company service that provides the name and phone number of your incoming caller. The Caller ID service transmits the calling party's **directory information** (name and telephone number) as well as the date and time of the call between the first and second ring as a **binary FSK** (**frequency shift keying**) signal. Click CallerID_audio_example.mp3[5] to listen to a typical Caller ID FSK signal embedded between the first and second ringer pulses.

After successfully completing this project, your LabVIEW application will be able to read audio recordings such as CallerID_audio_example.mp3[6] and then extract the Caller ID message for display.

Figure 3.1 illustrates the Caller ID decoder system to be constructed in this project.

[4] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

[5] http://cnx.org/content/m18708/latest/CallerID_audio_example.mp3

[6] http://cnx.org/content/m18708/latest/CallerID_audio_example.mp3

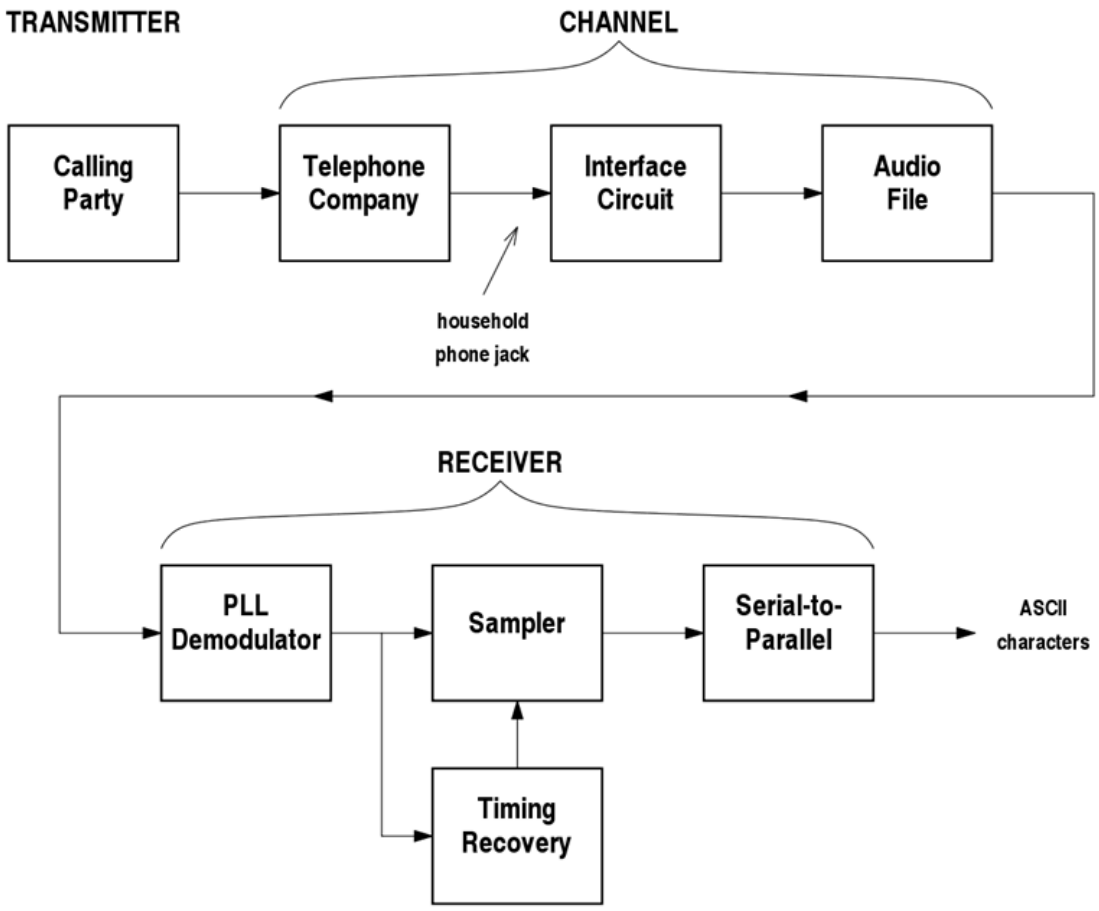


Figure 3.1: Caller ID decoder system to be constructed in this project

The process begins with a call placed by the calling party. The telephone company's **subscriber line interface card** (**SLIC**) in the telephone company **central office** (**CO**) signals the **customer premises equipment** (**CPE**) – telephone, modem, CallerID unit, etc. – with a **ringing pulse** (90 VRMS, 20 Hz, 2 seconds on, 4 seconds off). The CO repeats the ringing pulse as long as the CPE is **on hook**, that is, the phone has not been answered. Answering the phone call places the CPE in the **off hook** state, and the CO connects the calling party to the CPE. The terms "on hook" and "off hook" refer to the position of the ear piece or handset in early telephone equipment. The SLIC detects the CPE hook state by the amount of DC current in the customer loop: zero current means on-hook, and non-zero current (about 10 to 20mA) indicates off-hook. The Caller ID FSK signal is transmitted between the first and second ringing pulses provided the CPE is on-hook. For this reason, the interface circuit indicated in Figure 1 must be AC-coupled to the SLIC to prevent drawing DC current.

3.1.8 FSK Signal

Caller ID uses the Bell 202 modem standard:

- Binary FSK (two-level frequency shift keying)

- Symbol rate: 1,200 symbols/second
- Bit rate: 1 bit per symbol
- Logic 0 ("space"): 2,200 Hz
- Logic 1 ("mark"): 1,300 Hz

3.1.9 Caller ID Message Format

The complete Caller ID message is less than eight tenths of a second in duration. Listen to reduced_tempo_FSK.wav[7] , a reduced tempo version of just the FSK signal; the signal is stretched out in time by a factor of 4, but the original frequencies are preserved.

Hopefully you could discern three distinct regions of the signal:

1. Alternating 1's and 0's for 250 ms – this **channel seizure** region "wakes up" the demodulator and gives the symbol synchronization subsystem enough time to generate pulses synchronized to the FSK signal
2. Constant 1's for 150 ms – this **steady mark** region separates the channel seizure region from the datablock; the relatively long interval of constant "mark" level ensures that the first "space" symbol of the data block can be easily detected
3. Message bits for approximately 350 ms (the total time varies depending on the caller information) – this **data block** region contains the Caller ID information

The message consists of a sequence of 10-bit **frames**. A **start bit** of value 0 begins the frame, 8 bits of information follow, and the frame concludes with a **stop bit** of value 1. The 8 information bits begin with the **LSB** (least significant bit) and end with the **MSB** (most significant bit). The information bits form one byte.

The data block message bytes are organized as follows:

1. Message type, 1 byte – 0x80 (i.e., hexadecimal 80) for Multiple Data Message Format (MDMF), or 0x04 for Single Data Message Format (SDMF). Number-only Caller ID service uses SDMF, and number-plus-name service uses MDMF. Number-plus-name service is much more common today, and is used in this project.
2. Length of complete message, 1 byte – This length value **excludes** the single-byte checksum at the end of the data block
3. Data type, 1 byte – 0x01 = date and time, 0x02 = phone number, 0x04 = number not present, 0x07 = name, and 0x08 = name not present
4. Length of data, 1 byte
5. Data bytes, variable number according to length
6. Repeat Items 3, 4, and 5 as needed to complete the Caller ID message
7. Checksum, 1 byte – Add this to the modulo-256 sum of all the previous bytes in the message, including the message type and message length; a zero result indicates no errors detected

3.1.10 FSK Demodulator

The phase-lock loop (PLL) can easily discern the change in frequencies of an FSK signal. The LabVIEW Modulation Toolkit provides a PLL component that serves as an FSK demodulator for this project. Refer to the theory-of-operation screencast video in cid_Demodulator.vi (Section 5.7.1) learn how to use this PLL.

[7] http://cnx.org/content/m18708/latest/reduced_tempo_FSK.wav

3.1.11 Timing Recovery

The Caller ID message symbol rate is 1,200 symbols per second. The baseband output of the FSK demodulator must be compared to a threshold and sampled at the symbol rate to recover the serial bit stream. The timing recovery system ensures that the thresholded demodulator output is sampled near the midpoint of the symbol. In this project a "local oscillator" produces a squarewave at a nominal frequency of 1,200 Hz. The local oscillator phase is synchronized to the thresholded FSK demodulator output. That is, the local oscillator phase is reset each time a transition is detected on the FSK demodulator output.

3.1.12 Procedure

3.1.12.1 Download required project files

Download the required project files contained in CID_Decoder_Project_Files.zip[8] ; unpack the files to the same folder in which you plan to build your LabVIEW subVIs and top-level application VIs.

The .zip archive contains the following files:

- `cid_ParseMessage.vi` – Accepts a text string containing the Caller ID data block bytes, and parses the string to extract the Caller ID data fields, i.e., date, time, number, and name; also returns information about the data block itself, namely, message type (SDMF or MDMF), message length, checksum value, and result of checksum calculation. All of the values are returned in a single cluster.
- `cid_Display.ctl` – Custom front-panel control to display Caller ID data block information contained in the cluster generated by `cid_ParseMessage.vi`. Follow these steps to place the control on the front panel: Display the front panel, right-click and choose "Select a Control...", and then choose `cid_Display.ctl`.
- `CallerID-N.wav` – Two audio recordings of Caller ID signal embedded between the first and second ringer pulses. The recordings include three ringer pulses, are approximately 17 seconds in duration, and are sampled at 44.1 kHz.
- `CallerID-N_19.2kHz.wav` – The same audio recordings downsampled to 19.2 kHz to produce 16 samples per symbol, the default value for many of the LabVIEW Modulation Toolkit subVIs. Either audio file can be used for this project, although the downsampled versions shorten run times of the Caller ID decoder application.
- `cid_Recorder.vi` – LabVIEW VI to monitor the soundcard input for ringer activity; when detected, record for a fixed time interval and save to a .wav file. Useful to collect several Caller ID signals automatically. Requires a suitable interface circuit between the telephone wall jack and the computer sound card.

3.1.12.2 Study the Caller ID signal audio recordings

Open the `CallerID-1.wav` audio recording in Audacity[9] or an equivalent sound editor to view and listen to the signal. Use the zoom features to study the fine detail of the FSK signal, especially at the transitions between frequencies.

Repeat for the other .wav audio files included in the download distribution.

Are you able to discern any obvious differences among the various audio recordings?

3.1.12.3 SubVI construction

Possible approaches to analyze the signal generated by the telephone central office include: (1) process each sample as it arrives and generate the decoded message "on the fly," or (2) record the entire signal and then make repeated passes over the recording as needed to extract the message. Real-time implementation

[8] http://cnx.org/content/m18708/latest/CID_Decoder_Project_Files.zip
[9] http://audacity.sourceforge.net

requires the former approach, while the latter "off-line" approach is easier to implement as a sequence of subVI calls, and is therefore the method of choice for this project.

Build the subVIs listed below. You may already have some of these available from previous projects.

Demonstrate that each of these subVIs works properly before continuing to the next part. The order in which you build the subVIs does not matter, however, the order presented roughly corresponds to the general processing flow that begins with the audio recording and ends with a collection of bytes.

1. util_GetAudio.vi (Section 5.1.5.2)
2. cid_Demodulator.vi (Section 5.7.1)
3. regen_Sampler.vi (Section 5.4.4.2)
4. util_EdgeDetector.vi (Section 5.1.5.1)
5. regen_BitClock.vi (Section 5.4.1.1)
6. cid_DetectStartBit.vi (Section 5.7.2)
7. util_BitstreamToText.vi (Section 5.1.2.3)

3.1.12.4 Isolate and demodulate the FSK signal

Build a top-level VI that isolates and demodulates the FSK portion of the complete Caller ID audio signal. util_GetAudio (Section 5.1.5.2) to load the .wav audio file, and then pass this signal to cid_Demodulator.vi (Section 5.7.1). Connect front-panel controls to the four parameter inputs of cid_Demodulator.vi (Section 5.7.1). Create a mixed-signal waveform chart to display the four signals associated with cid_Demodulator.vi (Section 5.7.1), namely: `FSK signal` (the original audio signal applied to the demodulator input), `baseband signal` (the demodulated output), `phase error magnitude`, and `PLL locked`.

The Figure 3.2 screencast video describes how to set up a LabVIEW "Mixed Signal Graph" to plot the waveform data type and Boolean 1-D array on a common timescale, much like an oscilloscope display with multiple analog and digital signals.

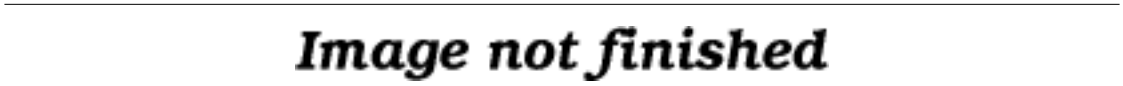


Figure 3.2: [video] Set up a "Mixed Signal Graph" to display both "analog" and "digital" signals on a common timescale

Set the demodulator VCO carrier frequency to the average value of the mark and space frequencies of the FSK signal.

Experiment with the remaining demodulator parameters `VCO gain` (start with values in the range 0.05 to 0.20), `phase error LPF cutoff frequency`, and `comparator threshold for PLL lock` to satisfy the following goals:

- Baseband signal in FSK region "looks good", i.e., reasonably quick rise time without high-frequency ringing or other non-ideal artifacts
- `PLL locked` is active (`T`) only during the FSK signal and is inactive during silence and ringer pulses.

Report your four demodulator parameter values and plot the mixed signal graph that demonstrates the ability of your system to identify the time region over which the FSK signal is active.

3.1.12.5 Sample the baseband signal

Use regen_Sampler.vi (Section 5.4.4.2) to extract the FSK signal region from the baseband signal produced by cid_Demodulator.vi (Section 5.7.1). Note that regen_Sampler.vi (Section 5.4.4.2) is used like a "gating" circuit here: the `PLL locked` signal is active over the entire time that the FSK signal is detected, therefore regen_Sampler.vi (Section 5.4.4.2) picks out every value from the original audio recording over which `PLL locked` is active.

Add the bit sync system to your VI using a zero-crossing comparator, util_EdgeDetector.vi (Section 5.1.5.1) (two instances), regen_BitClock.vi (Section 5.4.1.1) set to 1,200 Hz, and another instance of regen_Sampler.vi (Section 5.4.4.2). Use the zero-crossing comparator to convert the FSK signal to a Boolean 1-D array. This signal is **not** a bitstream yet, but rather serves to identify the beginning of bit intervals. Use one edge detector to produce indicator pulses for each zero crossing of the baseband signal (both rising edges and falling edges). These indicator pulses serve as the synchronization for the bit clock oscillator, which produces a square wave synchronized to the baseband signal. The square wave transitions low-to-high at the beginning of the symbol interval and transitions high-to-low at the midpoint of the signal interval. Use an edge detector to produce indicator pulses at the midpoint of the symbol interval which control when the sampler should take samples from the isolated FSK signal.

Create another mixed signal waveform graph to display the isolated FSK signal, the thresholded version of this signal, the indicator pulses that synchronize the bit clock, the bit clock output, and "actual sampling instants" produced by the sampler.

Study your results to ensure that the demodulated baseband signal is sampled properly.

The Figure 3.3 screencast video shows how you can conserve front-panel real estate by placing two waveform graphs inside a tabbed control.

Figure 3.3: [video] Conserve front-panel real estate with a tabbed control

3.1.12.6 Decode the message bitstream

Use a zero-crossing comparator (specifically of the "less than zero" type) to convert the sampled baseband signal to a bitstream. Process this bitstream with cid_DetectStartBit.vi (Section 5.7.2) to extract only the data block portion of the bitstream.

Use util_BitstreamToText.vi (Section 5.1.2.3) to convert the bitstream to a sequence of 8-bit values ("string" data type), and then use `cid_ParseMessage` to convert the text string into an information cluster to be displayed with the custom front-panel control `CID Display`.

Include front panel indicators for the `framing error` and `text out` outputs from util_BitstreamToText.vi (Section 5.1.2.3). The string indicator can be easily switched from ASCII display to hexadecimal display, as needed: right-click on the front panel indicator and choose either "Normal Display" or "Hex Display."

The framing error indicator should be dark for the entire data block region; some "left over" bits are likely after the data block ends due to the delay until the FSK demodulator's `PLL locked` output returns to `F`, and the resulting framing errors may be safely ignored.

3.1.12.7 Evaluate the completed system

Choose one of the Caller ID signal recordings, and run your VI to decode the FSK signal. When all goes well, the `CID Display` indicator will show "OK" for both the parsing process and the checksum calculation.

Confirm that the data block values are correctly extracted by manually decoding the hex values in the text string front panel indicator. More specifically, copy the hex values to a piece of paper, and work through the interpretation of each byte. For example, the first byte of the data block is 0x80, which indicates the Caller ID message is of the MDMF (Multiple Data Message Format) type. The next byte is an 8-bit unsigned integer that indicates message length; convert the hexadecimal value to decimal and report this value. Continue in this way to demonstrate that you understand the significance of each byte in the data block.

Try your Caller ID decoder on other signal recordings. Report the `CID Display` indicator values for each audio signal recording.

3.1.13 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1
2. Couch, Leon W. II, "Digital and Analog Communication Systems," 7th ed., Pearson Prentice Hall, 2007. ISBN-10: 0-13-142492-0
3. Haykin, Simon, and Michael Moher, "Introduction to Analog and Digital Communication Systems," 2nd ed., Wiley, 2007. ISBN-13: 978-0-471-43222-7
4. Lathi, Bhagwandas P., "Modern Digital and Analog Communication Systems," 3rd ed., Oxford University Press, 1998. ISBN-10: 0-19-511009-9
5. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
6. Stern, Harold P.E., and Samy A. Mahmoud, "Communication Systems," Pearson Prentice Hall, 2004. ISBN-10: 0-13-040268-0
7. Wheeler, Tom, "Electronic Communications for Technicians," 2nd ed., Pearson Prentice Hall, 2006. ISBN-10: 0-13-113049-8

Chapter 4

Bandpass Communications Over the Speaker-Air-Microphone Channel

4.1 Speaker-Air-Microphone (SAM) Channel Characterization[1]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[2] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Communications / Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 4.1

NOTE: Visit LabVIEW Setup[3] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

4.1.1 Summary

A speaker and microphone connected to a computer serve as an excellent communications channel because the transmitted information is audible. Listening to the channel while making parameter adjustments and viewing plots builds additional insight into the specific modulation scheme under study.

The passband limits and bandwidth of the **SAM channel** (Speaker-to-Air-to-Microphone) must be characterized to effectively choose modulation parameters such as carrier frequency, bit rate, and signal pulse shape. This project describes how to characterize a SAM channel for use in subsequent projects that use bandpass modulation techniques.

[1] This content is available online at <http://cnx.org/content/m18666/1.2/>.
[2] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[3] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

4.1.2 Objectives

1. Measure the passband limits and bandwidth of a speaker-air-microphone (SAM) channel
2. Learn how to use LabVIEW Express subVIs for audio and power spectrum measurement

4.1.3 Deliverables

1. Summary write-up of your results
2. Hardcopy of all LabVIEW code that you develop (block diagrams and front panels)
3. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

4.1.4 Setup

1. LabVIEW 8.5 or later version
2. Computer soundcard
3. Speaker
4. Microphone

4.1.5 Textbook Linkages

Refer to the following textbooks for additional background on the project activities of this module; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 9
- Couch – Ch 6
- Haykin – Ch 1
- Haykin and Moher – Ch 8
- Lathi – Ch 3
- Proakis and Salehi (FCS) – Ch 5
- Proakis and Salehi (CSE) – Ch 4
- Stern and Mahmoud – Ch 2

4.1.6 Prerequisite Modules

If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[4] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

4.1.7 Introduction

Figure 4.1 illustrates the speaker-to-air-to-microphone (SAM) channel.

[4] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

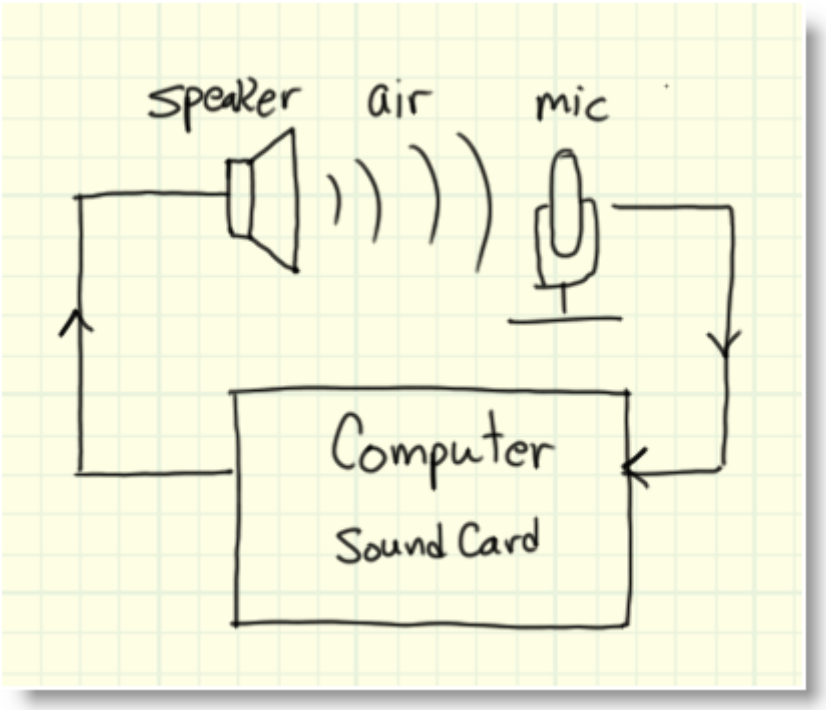


Figure 4.1: Speaker-air-microphone channel

The magnitude frequency response of the SAM channel can be easily measured by applying white noise to the speaker, recording the result, and calculating the power spectrum of the of the recorded signal. White noise contains equal contribution from all frequencies; therefore, any deviation from a flat power spectrum at the channel output must be the response of the channel.

The computer sound card forms part of the channel, as well. Sound card microphone inputs are often AC-coupled and do not pass DC. Microphone inputs are sometimes intentionally lowpass filtered to reduce high-frequency hiss, and all soundcard inputs include lowpass filters to serve as anti-aliasing filters.

Most modern sound cards are **full duplex**, meaning that they can simultaneously generate and record sound – a necessary feature for this project. Older **half duplex** sound cards can still be used but require two computers, one to transmit the sound and another to receive the sound.

4.1.8 Procedure

4.1.8.1 White noise source

Create a white noise sound source to serve as the SAM channel excitation:

1. Use the "Express | Output | Play Waveform" Express VI to generate audio for the speaker.
2. Use the "Express | Signal Analysis | Simulate Signal" Express VI to generate uniform white noise in the range -1 to +1.
3. Set the Express VI parameters to generate a signal with 44.1 kHz sampling frequency and duration of 10 seconds.

Refer to the video of Figure 4.2 for LabVIEW coding techniques for the white noise source.

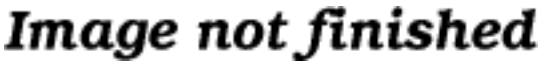


Figure 4.2: [video] LabVIEW coding techniques for white noise source

4.1.8.2 Recording device

Create a device to record the microphone signal:

1. Use the "Express | Input | Acquire Sound" Express VI configured for the same sampling frequency as the white noise source (44.1 kHz) and a 0.1 second duration.
2. Monitor the recorded signal as a time-domain plot.
3. Use the "Express | Signal Manipulation | Extract Portion of Signal" Express VI to discard a user-specified number of samples at the beginning of the waveform to account for transient startup.

Refer to the video of Figure 4.3 for LabVIEW coding techniques for the recording device.

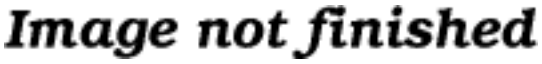


Figure 4.3: [video] LabVIEW coding techniques for recording device

4.1.8.3 Power spectrum display

Create a graphical indicator to display the power spectrum of the recorded microphone signal:

1. Use the "Express | Signal Analysis | Spectral Measurements" Express VI configured for power spectrum measurement.
2. Create a "Waveform Graph" indicator to display the power spectrum. Calibrate the Y-axis in decibels and the X-axis in hertz. Use logarithmic mapping for the frequency axis.
3. Enclose the power spectrum measurement and recording-related subVIs in a for-loop structure (N=80). This arrangement takes 80 short recordings, calculates the power spectrum for each, and averages them together to improve the estimated frequency response of the channel. The value of N is chosen to allow the recording/measurement process to complete before the white noise source stops.

NOTE: The white noise source and recording/measurement sections of the overall VI must be kept separate from each other to allow them to operate in parallel.

Refer to the video of Figure 4.4 for LabVIEW coding techniques for the power spectrum display.

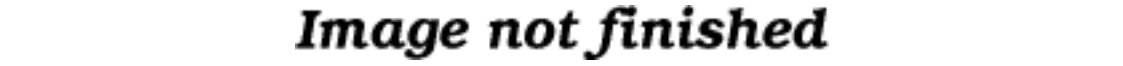


Figure 4.4: [video] LabVIEW coding techniques for power spectrum display

4.1.8.4 SAM channel physical setup

Set up the hardware for the SAM channel. Place the microphone within a few inches of the speaker. Adjust the soundcard setup and volume controls to generate and record sound to satisfy two goals:

1. Signal applied to sound card lies within the range ±1 with no clipping (saturation)
2. Signal recorded from sound card fills as much as possible the range ±1 without clipping

The video of Figure 4.5 for LabVIEW coding techniques for the power spectrum display.

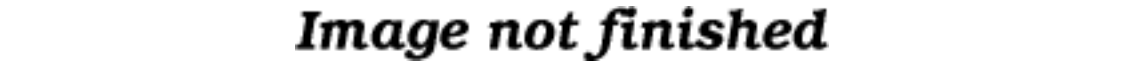


Figure 4.5: [video] Adjust the audio settings in Windows XP

4.1.8.5 SAM channel measurements

Apply the white noise source to the SAM channel and record its response for 5 to 10 seconds. Also record the **noise floor** of the channel; use the identical speaker/microphone arrangement but do not generate any sound (set the "Noise Amplitude" parameter of the noise source to zero).

Subtract the noise floor measurement from the white noise measurement. The result should be approximately zero dB outside the passband region of the SAM channel.

Make hardcopy of your two measurements on the SAM channel as well as the measurement with the noise floor subtracted (right-click on the "Waveform Graph" indicator and choose "Export Simplified Image"). Annotate the plots with hand-written labels to identify the lower passband limit, the upper passband limit, and the overall bandwidth of the channel.

State your numerical criterion for choosing the passband limits. **Half-power frequency**, the frequency at which the response drops by 3 dB, is a standard method.

4.1.9 Optional: To Go Further

Your instructor may ask you to complete one or more of the following activities:

1. If supported by your sound card, change the soundcard recording device to "Stereo Mix" and repeat the two measurements above. This soundcard loopback test helps you distinguish which parts of the frequency response to attribute to the soundcard itself as opposed to the speaker/microphone combination.
2. Repeat the SAM channel measurement for other combinations of speakers and microphones
3. Enhance the overall VI to record the white noise source on a first run of the VI and the noise floor on the next run of the VI. Subtract the noise floor from the white noise recording and display the result.

The video of Figure 4.6 shows how to retain a measurement from one run of the VI to the next.

Image not finished

Figure 4.6: [video] Retain a measurement from one run of a VI to the next

4.1.10 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1
2. Couch, Leon W. II, "Digital and Analog Communication Systems," 7th ed., Pearson Prentice Hall, 2007. ISBN-10: 0-13-142492-0
3. Haykin, Simon. "Communication Systems," 4th ed., Wiley, 2001. ISBN-10: 0-471-17869-1
4. Haykin, Simon, and Michael Moher, "Introduction to Analog and Digital Communication Systems," 2nd ed., Wiley, 2007. ISBN-13: 978-0-471-43222-7
5. Lathi, Bhagwandas P., "Modern Digital and Analog Communication Systems," 3rd ed., Oxford University Press, 1998. ISBN-10: 0-19-511009-9
6. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
7. Proakis, John G., and Masoud Salehi, "Communication Systems Engineering," 2nd ed., Pearson Prentice Hall, 2002. ISBN-10: 0-13-061793-8
8. Stern, Harold P.E., and Samy A. Mahmoud, "Communication Systems," Pearson Prentice Hall, 2004. ISBN-10: 0-13-040268-0

4.2 Binary ASK Transmitter[5]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[6] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 4.2

NOTE: Visit LabVIEW Setup[7] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[5] This content is available online at <http://cnx.org/content/m18668/1.1/>.
[6] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[7] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

4.2.1 Summary

Three parameters specify a sinusoidal carrier wave: amplitude, frequency, and phase. An individual parameter or combination of parameters may be **modulated** by a message to communicate information. The most basic modulation schemes switch a single parameter between two values to signal a binary 0 or binary 1.

In this project, construct and study a transmitter that switches the carrier wave's amplitude between zero and a non-zero value. The term **switching** is also called **keying** (as in a telegraph key), and so the transmitter in this project can be said to use **binary amplitude shift keying** (**binary ASK**).

4.2.2 Objectives

1. Study the spectral characteristics of binary ASK signals using both rectangular and raised cosine pulse shapes
2. Translate the ASK transmitter block diagram into a LabVIEW block diagram
3. Develop an ASK transmitter for the speaker-air-microphone (SAM) channel

4.2.3 Deliverables

1. Summary write-up of your results
2. Hardcopy of all LabVIEW code that you develop (block diagrams and front panels)
3. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

4.2.4 Setup

1. LabVIEW 8.5 or later version
2. Computer soundcard
3. Speaker

4.2.5 Textbook Linkages

Refer to the following textbooks for additional background on the project activities of this module; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 14
- Couch – Ch 5
- Haykin and Moher – Ch 7
- Lathi – Ch 13
- Proakis and Salehi (FCS) – Ch 10
- Stern and Mahmoud – Ch 5

4.2.6 Prerequisite Modules

Complete the lab project Speaker-Air-Microphone (SAM) Channel Characterization (Section 4.1) before you begin this project.

If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[8] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

[8] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

4.2.7 Introduction

Bandpass channels possess a non-zero lower cutoff frequency, and therefore cannot transmit a **baseband** signal. For example, the channel established between two voice-grade telephones begins at 300 Hz and ends at 3,000 Hz. A digital signal (baseband type) must be shifted in frequency so that its significant frequency components all exist within the 300 to 3,000 Hz range. Frequency shifting may be accomplished by impressing the baseband signal onto a sinusoidal **carrier wave**.

A sinusoidal carrier wave $c(t) = A_c \cos(2\pi f_c t + \varphi_c)$ possesses three parameters that can be switched (or keyed) by a binary message signal: **amplitude**, **frequency**, and **phase**; the resulting **digital continuous wave modulation** schemes are called ASK (amplitude shift keying), FSK (frequency shift keying), and PSK (phase shift keying), respectively.

The Figure 4.7 screencast video introduces the mathematical notation used in this module to discuss ASK modulation, and includes a visualization of the ASK waveform.

Image not finished

Figure 4.7: [video] Mathematical notation for ASK modulation and visualization of the ASK waveform

Figure 4.8 illustrates the block diagram of a binary ASK transmitter.

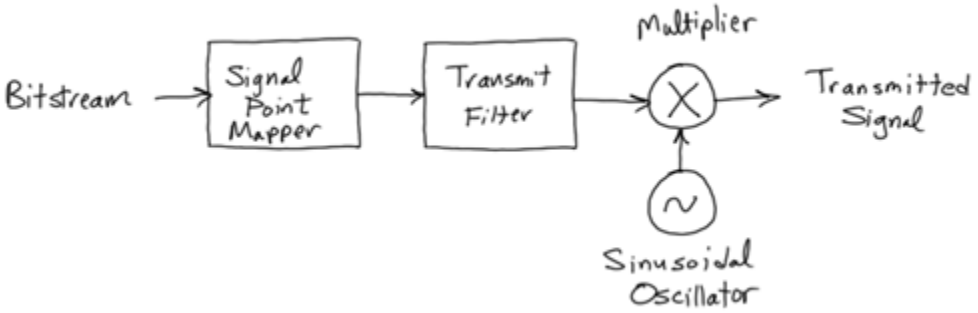


Figure 4.8: ASK transmitter block diagram

The transmitter's **signal point mapper** selects a value for each bit of the binary message (bitstream), and the **transmit filter** generates an analog signal waveform to be transmitted through the channel. The transmit filter is also known as the **pulse shaping filter**. Binary ASK maps a binary 1 to $\sqrt{E_b}$ and a binary 0 to zero; E_b denotes the energy per bit. The transmit filter scales a standard pulse shape by these values to produce the baseband signal, which in turn is shifted in frequency to match the channel's center frequency by multiplying by a sinusoidal carrier waveform to produce the transmitted signal. The Figure 4.9 screencast video discusses the spectrum of the transmitted signal, especially the impact of a rectangular pulse shape on the required bandwidth of the ASK signal.

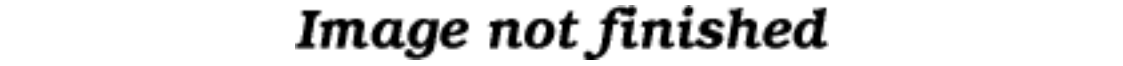


Figure 4.9: [video] ASK spectrum with rectangular pulses

As discussed in the previous video, the ASK signal created with rectangular pulses is spectrally inefficient. From an intuitive point of view, signals with sharp corners always possess a wideband spectrum. Rounding the corners should therefore produce a transmitted signal that does not require as much bandwidth.

The **raised cosine pulse** is a standard pulse shape widely used in communication systems that offers much better spectral efficiency; see the video in pam_RaisedCosinePulse.vi (Section 5.2.1.1) for more background on this important pulse shape, including an explanation of its **excess bandwidth** pulse shape parameter. The Figure 4.10 screencast video discusses the spectrum of the transmitted ASK signal with raised cosine pulse shaping.

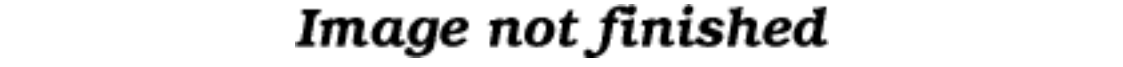


Figure 4.10: [video] ASK spectrum with raised cosine pulse shaping

Consider once again the transmitter block diagram of Figure 4.8. In a fully digital implementation, the pulse shaping filter output must be a sampled-value waveform. Rectangular pulse shapes are easy to implement: a given binary symbol simply maps to an array of constant values. Nonrectangular pulses take a bit more effort, however, especially when the pulse shape must extend over more than one bit interval.

The Figure 4.11 screencast video describes a pulse shaping filter implementation that can be used with any pulse shape. The basic idea involves driving an FIR filter with an impulse train.

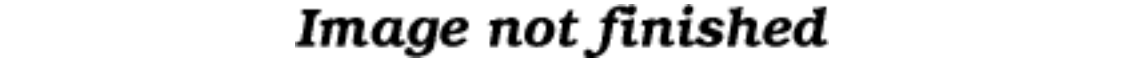


Figure 4.11: [video] Signal point mapper and pulse shaping filter implementation using an FIR filter driven by an impulse train

4.2.8 Procedure

4.2.8.1 SubVI construction

Build the subVIs listed below. You may already have some of these available from previous projects.

Demonstrate that each of these subVIs works properly before continuing to the next part.

1. util_BitstreamFromRandom.vi (Section 5.1.1.1)
2. pam_SignalPointMapper.vi (Section 5.2.2.1)
3. pam_TransmitFilter.vi (Section 5.2.2.2)

4. pam_RectanglePulse.vi (Section 5.2.1.2)
5. pam_RaisedCosinePulse.vi (Section 5.2.1.1)
6. bpm_ProductModulator.vi (Section 5.3.2)

4.2.8.2 ASK transmitter

Assemble an ASK transmitter using the subVIs you created in the previous step; refer to the ASK transmitter diagram of Figure 4.8. Drive the transmitter with a random bitstream containing equiprobable binary values. Plot the power spectrum of the ASK signal using the "Express | Signal Analysis | Spectral Measurements" Express subVI. Connect the transmitter output to the speaker using the technique you learned in Speaker-Air-Microphone Channel Characterization (Section 4.1).

Include the following controls on the front panel:

- `fc, carrier frequency [Hz]`
- `fs, sampling frequency [Hz]`
- `Eb, energy per bit`
- `Tb, bit interval [s]`
- `bitstream length`
- `seed`

Include the following indicators on the front panel:

- `ASK power spectrum` – waveform graph
- `time domain` – transmit filter and product modulator signals overlaid on the same waveform graph
- `Rb, bit rate [Hz]`
- `samples per bit interval`
- `total signal duration [s]`

Figure 4.12 illustrates a suggested layout for the VI front panel and shows the expected results of the initial parameter choices for the next section.

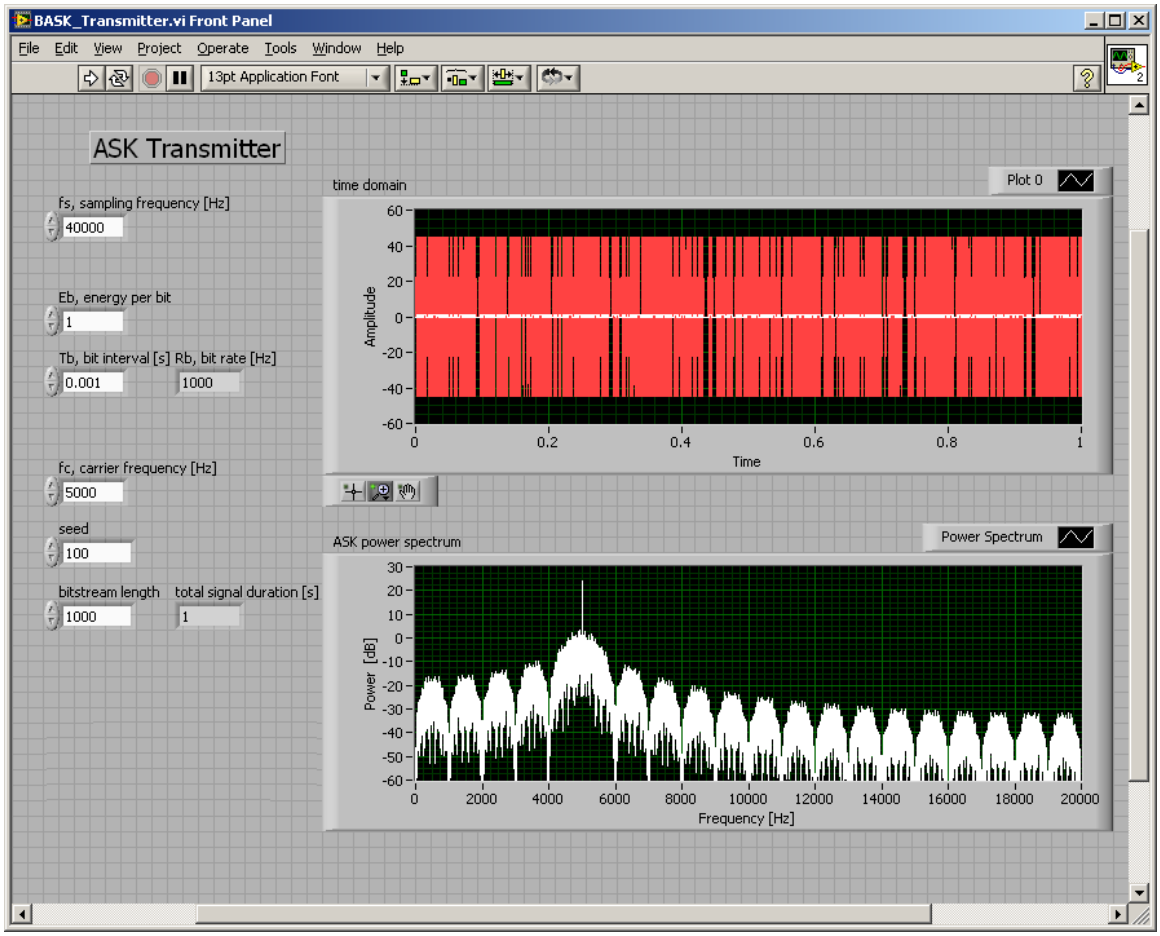


Figure 4.12: ASK transmitter VI front panel (click image to see fullsize version)

4.2.8.3 ASK transmitter parameter experiments

Begin with the following front panel control values:

- `fc, carrier frequency [Hz]` = 5,000 Hz
- `fs, sampling frequency [Hz]` = 40,000 Hz
- `Eb, energy per bit` = 1
- `Tb, bit interval [s]` = 0.001 s
- `bitstream length` = 1000
- `seed` = -1

These values should produce a bit rate of 1,000 Hz and total signal duration of 1 second.

1. Run the VI several times. You should observe a different sequence for the bitstream for each run. Next, change the `seed` to an integer larger than -1 and run the VI several times again. You should now observe the bitstream sequence to be the same for each run. Use a seed value other that -1 to generate a constant bitstream sequence, when needed.

2. Vary the bit energy E_b and study the time-domain signals and power spectrum. Summarize the behavior of the signals as a function of E_b.
3. Vary the bit interval T_b and study the time-domain signals and power spectrum. Summarize the behavior of the signals as a function of T_b. Be sure to comment on the following points: (a) What does ASK sound like for a long bit interval such as 0.01 seconds compared to a short bit interval such as 0.0005 seconds? (b) How does the bit rate value manifest itself in the power spectrum display? Hint: Recall what you know about the first-null bandwidth of a sinc function. (c) How does the choice of bit interval affect the total time to transmit the message?
4. Vary the carrier frequency f_c and study the time-domain signals and power spectrum. Summarize the behavior of the signals as a function of f_c.
5. Reset the front panel controls to the initial value listed at the beginning of this section. Change the carrier frequency to 5,001 Hz, then to 5,002 Hz, and so on in small steps. What change do you observe in the sound of the ASK signal and the time-domain plot? Hint: zoom in on the ASK signal waveform so that you can see the signal between bit transition. Draw a conclusion: what relationship must exist between the carrier frequency f_c and the bit interval T_b to ensure that chopped edges do not occur?

Plot time-domain waveforms and power spectra for several representative choices of parameter values.

4.2.8.4 ASK transmitter parameter experiments: raised cosine pulse shaping

1. Modify the VI front panel to include a Boolean control to conveniently switch between rectangular and raised cosine pulse shapes. Use a case structure on the block diagram to generate the two types of pulse shapes. Include front panel controls for the `alpha` (excess bandwidth) parameter and `bit intervals for support` controls of the pam_RaisedCosinePulse.vi (Section 5.2.1.1) subVI.
2. Adjust the front panel control values to match those specified at the beginning of the previous section.
3. Select a rectangular pulse shape, and run the VI one time with autoscaling enabled on the power spectrum display. Right-click on the Y-axis of the power spectrum display and uncheck the "AutoScale Y" option. Next, select the raised cosine pulse shape and run the VI again. Zoom in on the time-domain signal plot, and disable autoscaling on the X-axis to ensure that the zoom level is retained from one run of the VI to the next.
4. Experiment with different values of the raised cosine pulse controls, then compare your results to those of the rectangular pulse shape. Be sure to comment on the following points: (a) signal shape in the time domain, (b) signal shape in the frequency domain (power spectrum), and (c) sound of the transmitted signal – try a larger bit interval (and short bitstream length) so that you can clearly hear the difference between rectangular and raised cosine pulse shaping).
5. Recall the work that you did to characterize the SAM channel in Speaker-Air-Microphone (SAM) Channel Characterization (Section 4.1), especially the lower and upper passband limits. Select raised cosine pulse shaping, and then choose values for carrier frequency and bit interval to make the transmitted signal fully occupy the channel passband region. State your two values. Listen to this signal for a bit – what does a "fully utilized channel" sound like? Can you perhaps now explain some parts of a typical telephone dial-up modem initiation sequence[9] ? (Click link to listen).

4.2.9 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1
2. Couch, Leon W. II, "Digital and Analog Communication Systems," 7th ed., Pearson Prentice Hall, 2007. ISBN-10: 0-13-142492-0
3. Haykin, Simon, and Michael Moher, "Introduction to Analog and Digital Communication Systems," 2nd ed., Wiley, 2007. ISBN-13: 978-0-471-43222-7

[9] http://cnx.org/content/m18668/latest/dialup-modem.mp3

4. Lathi, Bhagwandas P., "Modern Digital and Analog Communication Systems," 3rd ed., Oxford University Press, 1998. ISBN-10: 0-19-511009-9
5. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
6. Stern, Harold P.E., and Samy A. Mahmoud, "Communication Systems," Pearson Prentice Hall, 2004. ISBN-10: 0-13-040268-0

4.3 Texting Over the Speaker-Air-Microphone (SAM) Channel[10]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[11] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 4.3

NOTE: Visit LabVIEW Setup[12] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

4.3.1 Summary

Texting and instant messaging are familiar and popular communication techniques. Text messages composed of ASCII characters, with each character composed of eight bits, ultimately form a stream of bits transmitted from the source to the receiver. Normally the text message moves through a complex data network, and the user only sees the endpoints of the system. In this project, however, you will be able to follow the text message on its complete journey through a binary ASK transmitter, a speaker-air-microphone (SAM) channel, and a binary ASK receiver.

This project depends on successful completion of the two prerequisite projects, Speaker-Air-Microphone (SAM) Channel Characterization (Section 4.1) and Binary ASK Transmitter (Section 4.2).

4.3.2 Objectives

1. Implement an ASK demodulator based on an envelope detector, a noncoherent detector
2. Appreciate the importance of a message preamble for successful receiver operation
3. Translate the ASK receiver block diagram into a LabVIEW block diagram
4. Develop an ASK receiver for the speaker-air-microphone (SAM) channel

[10] This content is available online at <http://cnx.org/content/m18670/1.1/>.
[11] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[12] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

4.3.3 Deliverables

1. Summary write-up of your results
2. Hardcopy of all LabVIEW code that you develop (block diagrams and front panels)
3. Any plots or diagrams requested

NOTE: You can easily export LabVIEW front-panel waveform plots directly to your report. Right-click on the waveform indicator and choose "Export Simplified Image."

4.3.4 Setup

1. LabVIEW 8.5 or later version
2. Modulation Toolkit 4.0 or later version
3. Computer soundcard that supports full-duplex operation
4. Speaker
5. Microphone
6. Two computers (optional)

4.3.5 Textbook Linkages

Refer to the following textbooks for additional background on the project activities of this module; see the "References" section below for publication details:

- Carlson, Crilly, and Rutledge – Ch 14
- Couch – Ch 3 and 5
- Haykin and Moher – Ch 7
- Lathi – Ch 13
- Proakis and Salehi (FCS) – Ch 10
- Stern and Mahmoud – Ch 5

4.3.6 Prerequisite Modules

Complete the lab projects Speaker-Air-Microphone (SAM) Channel Characterization (Section 4.1) and Binary ASK Transmitter (Section 4.2) before you begin this project.

If you are relatively new to LabVIEW, consider taking the course LabVIEW Techniques for Audio Signal Processing[13] which provides the foundation you need to complete this project activity, including: block diagram editing techniques, essential programming structures, subVIs, arrays, and audio.

4.3.7 Introduction

Figure 4.13 shows the hardware setup for this project. One computer running a "text sender" LabVIEW VI translates a text message into a bitstream, modulates the bitstream as an ASK signal, and plays the signal to a speaker. A second computer running a "text receiver" VI listens to the microphone signal, records and demodulates the ASK signal, regenerates the transmitted bitstream, and translates the bitstream back to text. The text sender and receiver VIs can run in parallel on a single computer to simplify development.

[13] *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing* <http://cnx.org/content/col10440/latest/>

Figure 4.13: High-level block diagram of the project

The frequency response and bandwidth of the speaker-air-microphone (SAM) channel were measured in an earlier project, Speaker-Air-Microphone (SAM) Channel Characterization (Section 4.1). As discussed in the Binary ASK Transmitter (Section 4.2) project, the bit rate and pulse shape can be adjusted to place the transmitted signal in various sub-bands or to fully occupy the available channel bandwidth. The channel bandwidth can be allocated in a number of different ways for this project. For example, use the full channel bandwidth to minimize the amount of time to transmit a message. However, transmitting messages simultaneously between two computers requires two different transmitter carrier frequencies and careful choice of bit rates and pulse shapes to ensure the signals remain contained in a sub-band of the channel.

The text message to be transmitted is a collection of ASCII characters. **ASCII**, the **American Standard Code for Information Interchange**, defines a mapping between familiar alpha-numeric characters (as well as other types of characters) and 7-bit binary patterns. For example, the five character sequence "hello" maps to the bit sequence `1101000 1100101 1101100 1101100 1101111`. See the Wikipedia article on ASCII[14] for ASCII mapping tables and further information.

LabVIEW offers a complete set of tools to manipulate and process text, including methods to translate text to bit sequences and vice versa. Swapping the random number generator currently serving as the message source in the transmitter from the Binary ASK Transmitter (Section 4.2) project with a text-to-bitstream converter quickly creates the necessary text sender for this project.

Figure 4.14 shows a high-level block diagram of the ASK receiver to be constructed in this project.

[14] http://wikipedia.org/wiki/ASCII

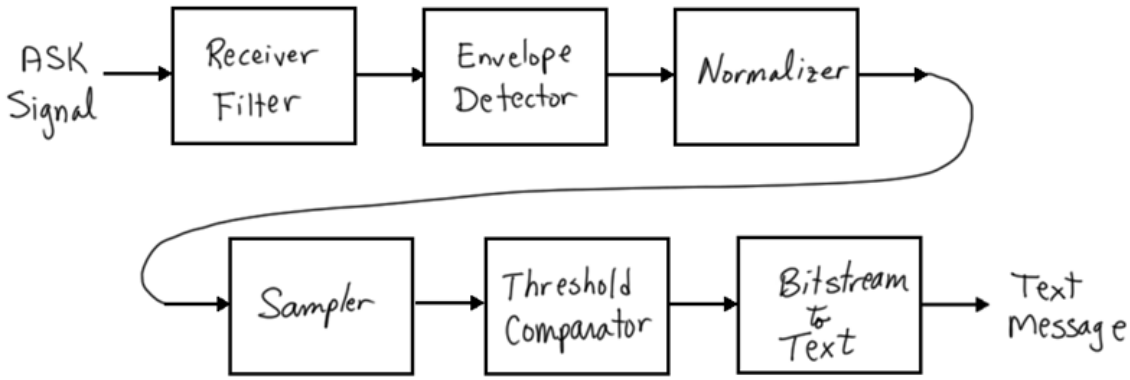


Figure 4.14: ASK receiver block diagram

The recorded microphone signal serves as the receiver input. The **receiver filter** is a bandpass filter centered at the transmitter's carrier frequency with a bandwidth selected to match the transmission bandwidth. The receiver filter therefore admits the signal of interest but removes out-of-band noise and signals in adjacent bands.

The ASK signal spectrum contains a strong carrier component, making noncoherent detection feasible. Coherent detection requires carrier synchronization, a more complex approach. Noncoherent detection, on the other hand, can be simply implemented using an **envelope detector**. The envelope detector recovers the baseband signal. The Figure 4.15 screencast video discusses the envelope detector principles of operation, implementation by a lowpass filter followed by a rectifier, and choice of lowpass filter corner frequency based on bit rate.

Image not finished

Figure 4.15: [video] Envelope detector principles of operation, implementation, and parameter selection

The **normalizer**, the third block in the sequence, shifts the baseband signal to eliminate the DC bias and scales the signal to the range ± 1. The normalizer can be considered a type of **automatic gain control** (**AGC**) to compensate for variations in speaker signal strength and microphone gain settings.

Up to this point the baseband signal can be considered an "analog" signal, even though it is in reality a discrete-time sampled signal. The **sampler** selects a single sample point from the baseband signal once each bit interval. The sampling process itself is very simple, but how does the sampler know when a bit interval has elapsed? Fortunately a hidden yet vital component in the sampler block called the **bit synchronizer** extracts timing pulses directly from the baseband signal, **provided** the signal meets certain requirements.

The Figure 4.16 screencast video discusses the bit synchronizer principles of operation and implementation considerations.

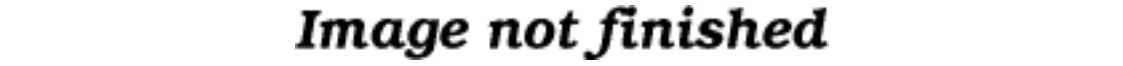


Figure 4.16: [video] Bit synchronizer principles of operation and implementation

As discussed in the previous video, the bit synchronizer's bandpass filter can be considered a high-Q resonator tuned to the bit rate, where **Q** stands for **quality factor**, the ratio of the bandpass filter's center frequency to its bandwidth; high Q indicates narrow bandwidth. The resonator produces sustained oscillations when the input bitstream contains approximately equiprobable ones and zeros. The high Q value assures that the resonator will continue to oscillate for a short time even when the bitstream has occasional runs of constant values. A high-Q resonator possesses **inertia** – once oscillations have started, the resonator will continue to oscillate for a time, even without excitation. However, just as in any system with inertia, an inactive resonator requires excitation for a significant period of time before oscillation reaches a useful amplitude.

Now consider a receiver waiting for a signal, and suppose the transmitter only generates nonzero signals when it is actually sending a message. If the bitstream can only be sampled once the bit synchronizer "revs up," so to speak, how can the first few bits of the message be correctly sampled? In fact, the first few bits of the message **could not** be recovered this way. However, the transmitter is free to transmit additional bits **before** the message bits – a preliminary bit sequence called a **preamble**.

The Figure 4.17 screencast video discusses the design of the preamble to accomplish two goals: (1) to "wake up" the bit synchronizer (more formally called **channel seizure**), and (2) to facilitate **frame synchronization**, the process of recovering meaningful symbols (ASCII characters, in this case) from the stream of ones and zeros.

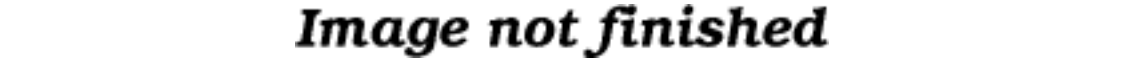


Figure 4.17: [video] Preamble design for bit synchronizer "wake up" and frame synchronization

Consider once again the receiver block diagram of Figure 4.14. The **threshold comparator** (also called the **decision device**) follows the sampler and compares each sampled signal value to zero. A sampled signal value higher than the threshold is declared a binary `1`, and a value lower than the threshold is declared a binary `0`. The threshold comparator completes the bitstream **regeneration** process; in the absence of noise, the bitstream produced by the comparator is identical to the transmitted bitstream.

The last block in the receiver retrieves ASCII characters from the bitstream. The earlier Figure 4.17 screencast video briefly discussed the need for frame synchronization, which is now considered in more detail. Suppose the message preamble consisted only of the alternating one-zero pattern necessary to start the bit synchronizer. Certainly the total length of the preamble is known, so in principle it would seem that the first message bit could be determined after some well-defined number of cycles after the bit synchronizer starts up. The problem is that this "well-defined" number of cycles can vary slightly due to noise levels, signal amplitude, and many other variables; the first actual message bit cannot be determined reliably this way.

Also, suppose the individual ASCII bit patterns were all transmitted as one contiguous bitstream by packing all the 7-bit patterns one after another. If the first message could not be determined correctly, then **all** remaining bits would likewise be interpreted incorrectly. Good system design should be **fault tolerant**, meaning that an interpretation error on one character should not affect the entire message.

These problems are addressed by inserting **framing bits** around each ASCII character. The Figure 4.18 screencast video describes the concepts of framing bits and frame synchronization, and why a sequence of binary ones (also called the **steady mark** sequence) is an important second part of the preamble.

Figure 4.18: [video] Frame synchronization and steady mark sequence in preamble

4.3.8 Procedure

4.3.8.1 SubVI construction

Build the subVIs listed below. You may already have some of these available from previous projects.

Demonstrate that each of these subVIs works properly before continuing to the next part.

1. util_BitstreamFromText.vi (Section 5.1.1.2)
2. util_BitstreamToText.vi (Section 5.1.2.3)
3. util_BitsToWords.vi (Section 5.1.2.1)
4. sam_GrabAudioDynamic.vi (Section 5.6.2)
5. bpm_ReceiverFilter.vi (Section 5.3.3)
6. bpm_EnvelopeDetector.vi (Section 5.3.1)
7. regen_BitSync.vi (Section 5.4.1.2)
8. regen_FrameSync.vi (Section 5.4.1.3)
9. regen_ExtractPreamble.vi (Section 5.4.2.1)
10. regen_NormalizeToPreamble.vi (Section 5.4.2.2)
11. regen_Sampler.vi (Section 5.4.4.2)

4.3.8.2 Text sender

Copy the application VI you completed at the end of the Binary ASK Transmitter (Section 4.2) project to a new VI named `TextSender.vi`. Be sure to use the version that includes raised cosine pulse shaping as an option.

Modify the message generation section to use util_BitstreamFromText.vi (Section 5.1.1.2) to convert a front-panel text control value into a bitstream.

Design a preamble sequence that satisfies the requirements for bit sync and frame sync described in the introduction. Concatenate the preamble to the beginning of the message bitstream.

Clearly state your preamble sequence in your report.

Use util_BitstreamToText.vi (Section 5.1.2.3) to display the complete transmitted message in hexadecimal format.

Listen to your transmitted ASK signal and view the power spectrum for several choices of bit rate, carrier frequency, and pulse shape.

The Figure 4.19 screencast video describes two different ways to form the preamble sequence, and describes how to choose the hexadecimal display modes for text controls and indicators.

Image not finished

Figure 4.19: [video] LabVIEW coding tips for preamble sequence and hexadecimal displays

4.3.8.3 Audio test signals

Record several receiver input signals as audio .wav files as test signals to save development time in the next section. Use sam_GrabAudioDynamic.vi (Section 5.6.2) to capture the audio and save to a .wav file using "Graphics and Sound | Sound | Files | Sound File Write Simple.vi" as shown in Figure 4.20.

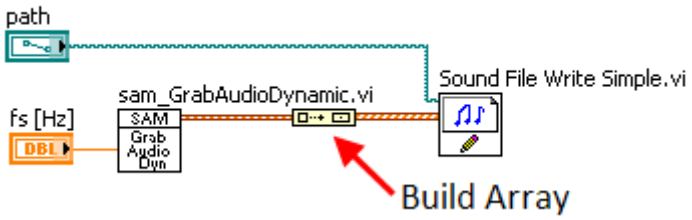


Figure 4.20: Record receiver input audio test signals

4.3.8.4 Text receiver construction

Review the ASK receiver block diagram of Figure 4.14. Assemble your subVIs to create `TextReceiver.vi` according to the block diagram. Note that several of the blocks on the diagram are composed of two subVIs: (1) The "normalizer" includes regen_ExtractPreamble.vi (Section 5.4.2.1) and regen_NormalizeToPreamble.vi (Section 5.4.2.2), (2) the "sampler" includes regen_BitSync.vi (Section 5.4.1.2) and regen_Sampler.vi (Section 5.4.4.2), and (3) the "bitstream to text" block includes regen_FrameSync.vi (Section 5.4.1.3) and util_BitstreamToText.vi (Section 5.1.2.3).

Remember to test as you build. Start with the input signal, add a processing block, and then add relevant waveform indicators to confirm correct operation before you proceed to the next processing block.

Include the following front panel control values:

- `recorded .wav file` – path
- `use recorded signal` – Boolean
- `fs, sampling frequency [Hz]` – DBL
- `fc, carrier frequency [Hz]` – DBL
- `Tb, bit interval [s]` – DBL
- `adjust sampling instant [% of bit interval] (0%)` – DBL
- `receiver filter bandwidth [Hz]` – DBL

The following front-panel indicators, especially the waveform graph indicators stacked vertically in the order listed below, visualize the complete "inner workings" of the ASK demodulation, bit sync, and frame sync

operations; these visualizations will greatly improve your understanding of the behavior of each major component of the receiver as well as their interrelationships:

- `received signal` – the receiver input signal from the microphone or test file
- `received spectrum` – with amplitude calibrated in dB
- `receiver filter spectrum` – received filter output signal, with amplitude calibrated in dB and Y-scale to match the received spectrum; output of the receiver filter
- `receiver filter` – receiver filter output signal
- `baseband` – envelope filter output signal
- `preamble` – regen_ExtractPreamble.vi (Section 5.4.2.1) output signal
- `normalized baseband` – regen_NormalizeToPreamble.vi (Section 5.4.2.2) output signal
- `baseband absolute value` – intermediate signal from regen_BitSync.vi (Section 5.4.1.2)
- `bandpass filter` – intermediate signal from regen_BitSync.vi (Section 5.4.1.2)
- `thresholded bandpass filter` – intermediate signal from regen_BitSync.vi (Section 5.4.1.2)
- `baseband + sampling instants` – envelope detector and "actual sampling instants" from regen_Sampler.vi (Section 5.4.4.2) signals overlaid on the same waveform graph

Also include the following indicators on the front panel, clustered together below the vertical stack of waveform graphs:

- `eye diagram` – waveform graph; output of the LabVIEW Modulation Toolkit "MT Format Eye Diagram.vi" to show the eye diagram of the baseband signal produced by the envelope detector
- `preamble detected?` – Boolean from regen_ExtractPreamble.vi (Section 5.4.2.1)
- `message detected?` – Boolean from regen_FrameSync.vi (Section 5.4.1.3)
- `bitstream length` – length of bitstream produced by regen_FrameSync.vi (Section 5.4.1.3)
- `bitstream` – Boolean 1-D array produced by threshold comparator
- `bitstream (hex)` – text with hex display mode to show output of threshold comparator
- `received message` – text produced by util_BitstreamToText.vi (Section 5.1.2.3)
- `framing error?` – Boolean 1-D array produced by util_BitstreamToText.vi (Section 5.1.2.3)
- `received frames` – Boolean 2-D array produced by util_BitsToWords.vi (Section 5.1.2.1)
- `bit rate [Hz]` – reciprocal of bit interval, Tb

The Figure 4.21 screencast video shows a suggested front-panel layout for the text receiver.

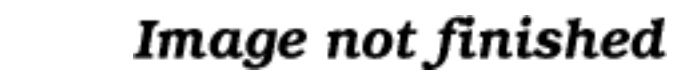


Figure 4.21: [video] Recommended panel layout for the text receiver

4.3.8.5 Testing and demonstration

Debug and test your receiver with your pre-recorded audio .wav files. Once you are satisfied that your receiver works properly, run your text sender as a separate application. Your soundcard is likely full duplex, and so both the text sender and text receiver should be able to access the soundcard simultaneously. You may also like to work with a friend with his or her own computer to make the texting activity more interesting.

Demonstrate your completed system to your instructor when you are ready.

You may also want to experiment with full-duplex texting between two different machines. Choose two different carrier frequencies, one for your friend and another for yourself. Be sure to choose a bit rate and pulse shaping so that each of you only uses up to half of the available SAM channel bandwidth. Try relatively

long messages and confirm that you can simultaneously send and receive messages. This is the essence of **frequency division multiplexing**.

4.3.9 References

1. Carlson, A. Bruce, Paul B. Crilly, and Janet C. Rutledge, "Communication Systems," 4th ed., McGraw-Hill, 2002. ISBN-13: 978-0-07-011127-1
2. Couch, Leon W. II, "Digital and Analog Communication Systems," 7th ed., Pearson Prentice Hall, 2007. ISBN-10: 0-13-142492-0
3. Haykin, Simon, and Michael Moher, "Introduction to Analog and Digital Communication Systems," 2nd ed., Wiley, 2007. ISBN-13: 978-0-471-43222-7
4. Lathi, Bhagwandas P., "Modern Digital and Analog Communication Systems," 3rd ed., Oxford University Press, 1998. ISBN-10: 0-19-511009-9
5. Proakis, John G., and Masoud Salehi, "Fundamentals of Communication Systems," Pearson Prentice Hall, 2005. ISBN-10: 0-13-147135-X
6. Stern, Harold P.E., and Samy A. Mahmoud, "Communication Systems," Pearson Prentice Hall, 2004. ISBN-10: 0-13-040268-0

4.4 Introduction to the LabVIEW Modulation Toolkit[15]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[16] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 4.4

NOTE: Visit LabVIEW Setup[17] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

4.4.1 Introduction

The LabVIEW Modulation Toolkit is an optional add-on to LabVIEW that offers a wide variety of subVIs to quickly and efficiently implement digital and analog communication systems. The toolkit subVIs combine to create many modulation schemes including ASK (amplitude shift keying), PAM (pulse amplitude modulation), QAM (quadrature amplitude modulation), FSK (frequency shift keying), MSK (minimum shift keying, a variant of FSK), PSK (phase shift keying), and CPM (continuous phase modulation). Channel impairments simulate various real-world troubles, including additive white Gaussian noise (AWGN), phase noise (also called jitter), fading, multi-tone interference, and quadrature inaccuracies. Standard visualization tools such as constellation plots, eye diagrams, and trellis diagrams are available, as are standard measurement

[15] This content is available online at <http://cnx.org/content/m18715/1.1/>.
[16] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[17] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

tools for bit error rate (BER), quadrature impairments, burst timing, and modulation quality. Channel coding with linear block codes and convolutional codes is supported, as well as direct sequence spread spectrum (DSSS). Channel equalization is available to correct inter-symbol interference (ISI).

The Figure 4.22 screencast video continues the introduction to the LabVIEW Modulation Toolkit with a quick walk-through of the various subVI palettes.

Figure 4.22: [video] Tour of the LabVIEW Modulation Toolkit subVI palettes

4.4.2 Complex Baseband Concept

The Modulation Toolkit uses complex baseband to represent signaling waveforms. All modulation schemes can be represented in this common mathematical notation. The real part of the complex signal is called the in-phase component and denoted I, while the imaginary part of the signal is called the quadrature component and denoted Q. The Figure 4.23 screencast video explains the mathematical foundation of the complex baseband concept, and describes how several different modulation schemes can all be conveniently represented in this notation.

Figure 4.23: [video] Explanation of the complex baseband concept

4.4.3 Demonstrations

The LabVIEW Modulation Toolkit offers a powerful way to quickly implement and explore a wide variety of digital communication systems. A good working knowledge of digital modulation schemes is prerequisite to effective use of the toolkit, however. Working through the detailed implementations of the projects Texting Over the Speaker-Air-Microphone (SAM) Channel (Section 4.3), Hamming Block Code Channel Encoder (Section 2.1), Hamming Block Code Channel Decoder (Section 2.2), and Caller ID Decoder (Section 3.1) will give you the experience necessary to make intelligent use of the toolkit. This section shows how an existing project can be re-implemented using the Modulation Toolkit, and then be quickly modified to try another modulation scheme. In addition, this section illustrates how a modulation scheme can be studied within LabVIEW to improve insight and understanding of the scheme.

4.4.3.1 BASK Transmitter

The Binary ASK Transmitter (Section 4.2) project implemented two-level amplitude shift keying (ASK) for transmission over a speaker-air-microphone (SAM) channel. The time-domain signaling waveform and associated power spectral density were also visualized in this project. The Figure 4.24 screencast video shows how to re-implement this project with components from the LabVIEW Modulation Toolkit.

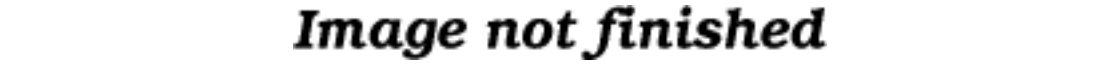


Figure 4.24: [video] Re-implement the BASK transmitter with components from the LabVIEW Modulation Toolkit

4.4.3.2 Alternative Transmitter Modulation Schemes

Once a particular modulation scheme has been implemented with Modulation Toolkit components, other modulation schemes can be explored with minimal effort. For example, the following screencast videos show how to convert the binary ASK scheme to multilevel ASK (Figure 4.25), and then to binary PSK (Figure 4.26), and finally to quadrature PSK (QPSK) (Figure 4.27).

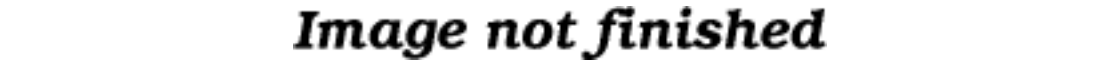


Figure 4.25: [video] Convert the binary ASK transmitter to multi-level ASK

Figure 4.26: [video] Convert the binary ASK transmitter to binary PSK

Figure 4.27: [video] Convert the binary PSK transmitter to quadrature PSK (QPSK)

4.4.3.3 QAM Exploration

Quadrature amplitude modulation (QAM) is the modulation scheme used by modern dial-up modems. Your textbook includes a section on QAM and explains the mathematical foundation for this scheme. The Figure 4.28 screencast video shows how you can quickly develop deeper insight into a modulation scheme such as QAM by visualizing how a bitstream maps to a signaling waveform, and by visualizing the complex baseband signal as a constellation plot.

Image not finished

Figure 4.28: [video] Exploring the QAM modulation scheme

4.4.4 Project Ideas

The LabVIEW Modulation Toolkit greatly simplifies the implementation effort for many types of modulation schemes. Try one or more of the following project activities to make interesting and practical communication systems using the toolkit:

1. Implement the project Texting Over the Speaker-Air-Microphone (SAM) Channel (Section 4.3) with Modulation Toolkit subVIs, specifically those for binary ASK. Confirm that you can obtain the same results as on the original version.
2. Implement the project Texting Over the Speaker-Air-Microphone (SAM) Channel (Section 4.3) with Modulation Toolkit subVIs, and try other modulation schemes such as M-ary ASK, PSK, FSK, and QAM. Compare the spectra of the various schemes, and study the impact of channel noise for schemes with a high number of bits per symbol.
3. Implement the projects Hamming Block Code Channel Encoder (Section 2.1) and Hamming Block Code Channel Decoder (Section 2.2) using the block coding subVIs from the Modulation Toolkit. Pick a modulation scheme and introduce channel impairments, and then measure the bit error rate with and without block coding.
4. Implement the Caller ID Decoder (Section 3.1) project using the FSK-related subVIs from the Modulation Toolkit.

4.4.5 Additional Project Resources

The National Instruments Developer Zone offers twenty-two software simulation and examples that explore a wide variety of communication systems concepts. Some of these examples require the LabVIEW Modulation Toolkit, while others do not. Visit http://zone.ni.com/devzone/cda/tut/p/id/6037#software[18] to access these projects, which include:

1. Amplitude Modulation – This example includes background information and step-by-step instructions that examine Amplitude Modulation (AM). Construct a LabVIEW VI that transmits and receives a signal in software using AM.
2. Frequency Modulation – This example includes background information and step-by-step instructions that examine Frequency Modulation (FM). Construct a LabVIEW VI that transmits and receives a signal in software using FM.
3. Single Sideband Modulation (SSB) – This example examines Single Sideband Modulation (SSB) with a LabVIEW VI that produces a modulated single-sideband signal.
4. Amplitude Shift Keying (ASK) – This example includes background information and step-by-step instructions that examine the Amplitude Shift Keying (ASK) digital modulation scheme. Construct a LabVIEW VI that transmits and receives a bit stream in software using ASK.
5. Frequency Key Shifting (FSK) – Frequency Shift Keying (FSK) is a digital modulation scheme that modulates a carrier sinusoid's frequency to transfer digital information. In this step-by-step exercise, construct a LabVIEW VI that transmits and receives a digital bit stream in software using FSK.

[18] http://zone.ni.com/devzone/cda/tut/p/id/6037#software

6. Phase Shift Keying (PSK) – This example includes background information and step-by-step instructions that examine the Phase Shift Keying (PSK) digital modulation scheme. Construct a LabVIEW VI that transmits and receives a digital bit stream in software using PSK.
7. Differential Phase Shift Keying (DPSK) – This example includes background information and step-by-step instructions that examine the Differential Phase Shift Keying (DPSK) digital modulation scheme. Construct a LabVIEW VI that transmits and receives a digital bit stream in software using DPSK.
8. OQPSK – Offset Quadrature Phase Shift Keying (OQPSK) is a variant of Phase Shift Keying modulation that uses four different values of the phase to transmit. This example LabVIEW VI transmits and receives a digital bit stream in software using OQPSK.
9. Minimum Shift Keying (MSK) – This example examines the Minimum Shift Keying (MSK) digital modulation scheme.
10. QAM Symbol Mapping – This example includes background information and step-by-step instructions that examine the Quadrature Amplitude Modulation (QAM) digital modulation scheme.
11. QAM M-ary vs. Channel Noise – This step-by-step demo illustrates the effect of channel noise on an M-ary QAM signal with a LabVIEW-based simulation that shows how noise can effect the transmission of a textual message.
12. Phase-Locked Loops – This demo examines the theory behind phase-locked loops with a LabVIEW-based simulation that synchronizes the phase of a generated signal with a reference signal.
13. LPF and HPF Filter – This example includes background information and step-by-step instructions that explore high- and low-pass filters. Construct a LabVIEW VI that blocks or attenuates signals of frequencies outside the specified band.
14. Time Division Multiplexing – This example introduces Time Division Multiplexing (TDM) with a LabVIEW-based simulation that appends one signal to the end of another, and displays each in both analog and digital formats.
15. OFDM – This example examines orthogonal frequency-division multiplexing (OFDM) with a LabVIEW-based simulation of a multi-carrier OFDM digital communication system.
16. Pulse Width Modulation (PWM) – This example includes background information and step-by-step instructions that explore Pulse Width Modulation (PWM), a digital modulation scheme that transmits analog information by altering pulse width.
17. Pulse Position Modulation (PPM) – This example includes theory and step-by-step instructions that explore Pulse Position Modulation (PPM).
18. Pulse Amplitude Modulation (PAM) – This example includes background information and step-by-step instructions that explore Pulse Amplitude Modulation (PAM). In this exercise, construct a LabVIEW VI that transmits analog information by changing pulse amplitude.
19. IQ Data – This demo introduces IQ data and explores why it is useful in communications. Analyze three LabVIEW VI's that show how IQ data represents changes in the magnitude and phase of a sine wave.
20. Sampling Theorem – This step-by-step example examines the sampling theorem and how it is used to determine minimum sampling speeds. In this exercise, construct a LabVIEW VI that illustrates the concept behind the sampling theorem.
21. Channel Coding – This example examines the processing technique of channel coding with a LabVIEW-based simulation that illustrates how channel coding allows original data to recover from noise in the channel.
22. Carrier Recovery – Channel noise can have a significant effect on carrier recovery. In this demo, analyze a LabVIEW VI that shows what behavior can occur when channel noise is significant enough to prevent carrier locking.

Chapter 5

SubVI Specifications

5.1 General-Purpose Utilities

5.1.1 Bitstream Sources

5.1.1.1 util_BitstreamFromRandom.vi[1]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[2] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.1

NOTE: Visit LabVIEW Setup[3] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.1.1.1 LabVIEW SubVI: util_BitstreamFromRandom.vi

- **Description:** Generate a bitstream from a random number generator. The probability of generating a 1 can be controlled, as can the value of the random number seed.
- **Category:** General-purpose utility ("util" prefix)

5.1.1.1.2 Inputs (Controls)

1. `length (128)` – I32
2. `ones probability (0.5)` – DBL
3. `seed (-1)` – I32

Parentheses ( ) indicate default value; square brackets [ ] designate units.

[1] This content is available online at <http://cnx.org/content/m18528/1.1/>.
[2] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[3] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.1.1.1.3 Outputs (Indicators)

1. `bitstream out` – 1-D Boolean array

5.1.1.1.4 Required Behavior

- The bitstream length defaults to 128 bits.
- A "seed" value of -1 indicates that a new set of random bits should be generated each time the subVI is called. Positive seed values will cause the same pattern to be generated each time, with the particular seed value selecting a different pattern.

5.1.1.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[4] to learn the mechanics of subVIs.

Refer to the Figure 5.1 screencast video for LabVIEW coding tips and techniques specific to this subVI.

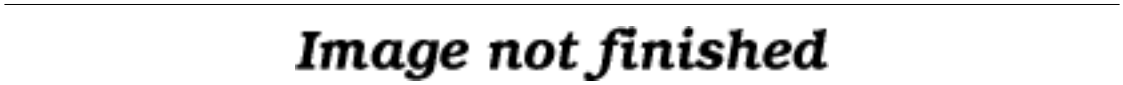


Figure 5.1: [video] LabVIEW coding tips and techniques for util_BitstreamFromRandom.vi

5.1.1.2 util_BitstreamFromText.vi[5]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[6] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.2

NOTE: Visit LabVIEW Setup[7] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[4] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[5] This content is available online at <http://cnx.org/content/m18631/1.1/>.
[6] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[7] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.1.1.2.1 LabVIEW SubVI: util_BitstreamFromText.vi

- **Description:** Generate a bitstream from a sequence of text characters. Framing bits (start bit and stop bit) may optionally be added to the bitstream. The bitstream is also available in the form of a wordstream.
- **Category:** General-purpose utility ("util" prefix)

5.1.1.2.2 Inputs (Controls)

1. `text` – string
2. `insert framing bits (F)` – Boolean
3. `start bit value (T)` – Boolean

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.1.2.3 Outputs (Indicators)

1. `bitstream out` – 1-D Boolean array
2. `wordstream out` – 2-D Boolean array

5.1.1.2.4 Required Behavior

- Converted text follows the indexing schemes imposed by the LabVIEW built-in nodes "String to Byte Array" and "Number to Boolean Array."
- When requested, the "start bit value" will be prepended to the 8-bit Boolean value, and the complement of the "start bit value" will be appended to the 8-bit Boolean value.
- The wordstream is an Nx8 2-D version of the 1-D bitstream (Nx10 if framing bits have been inserted), where "N" is the number of characters in the text control.

5.1.1.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[8] to learn the mechanics of subVIs.

Refer to the Figure 5.2 screencast video for LabVIEW coding tips and techniques specific to this subVI.

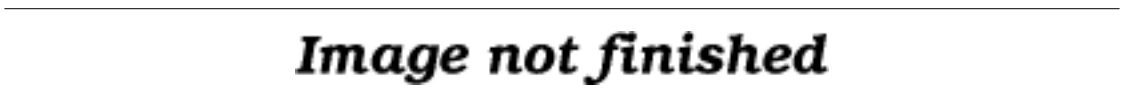


Figure 5.2: [video] LabVIEW coding tips and techniques for util_BitstreamFromText.vi

5.1.2 Bitstream Conversion

5.1.2.1 util_BitsToWords.vi[9]

[8] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

[9] This content is available online at <http://cnx.org/content/m18596/1.1/>.

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[10] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.3

NOTE: Visit LabVIEW Setup[11] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.2.1.1 LabVIEW SubVI: util_BitsToWords.vi

- **Description:** Convert a bitstream into a wordstream (sequence of k-bit words) by reshaping a 1-D Boolean array into a 2-D Boolean array.
- **Category:** General-purpose utility ("util" prefix)

5.1.2.1.2 Inputs (Controls)

1. `bitstream in` – 1-D Boolean array
2. `k, word size` – I32

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.2.1.3 Outputs (Indicators)

1. `wordstream out` – 2-D Boolean array

5.1.2.1.4 Required Behavior

- The inbound bitstream of length N produces an outbound wordstream (2-D array) of dimension (N/k) by k, where k is the wordsize.
- When N is not an integer multiple of k, the wordstream will be padded with Boolean "False" value.

5.1.2.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[12] to learn the mechanics of subVIs.

Refer to the Figure 5.3 screencast video for LabVIEW coding tips and techniques specific to this subVI.

[10] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[11] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[12] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

Image not finished

Figure 5.3: [video] LabVIEW coding tips and techniques for util_BitsToWords.vi

5.1.2.2 util_WordsToBits.vi[13]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[14] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.4

NOTE: Visit LabVIEW Setup[15] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.2.2.1 LabVIEW SubVI: util_WordsToBits.vi

- **Description:** Convert a wordstream (sequence of k-bit words) into a bitstream by reshaping a 2-D Boolean array into a 1-D Boolean array.
- **Category:** General-purpose utility ("util" prefix)

5.1.2.2.2 Inputs (Controls)

1. `wordstream in` – 2-D Boolean array

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.2.2.3 Outputs (Indicators)

1. `bitstream out` – 1-D Boolean array

5.1.2.2.4 Required Behavior

- The inbound wordstream (2-D array) of dimension (N/k) by k, where k is the wordsize, produces an outbound bitstream of length N.

[13] This content is available online at <http://cnx.org/content/m18551/1.1/>.
[14] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[15] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.1.2.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[16] to learn the mechanics of subVIs.

Refer to the Figure 5.4 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.4: [video] LabVIEW coding tips and techniques for util_WordsToBits.vi

---

5.1.2.3 util_BitstreamToText.vi[17]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[18] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.5

NOTE: Visit LabVIEW Setup[19] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.2.3.1 LabVIEW SubVI: util_BitstreamToText.vi

- **Description:** Interpret a bitstream as a sequence of text characters. Framing bits (start bit and stop bit) may optionally have been added to the bitstream, and are removed. Framing errors (mismatch between expected and actual values of framing bits) are indicated.
- **Category:** General-purpose utility ("util" prefix)

5.1.2.3.2 Inputs (Controls)

1. `bitstream in` – Boolean 1-D array
2. `includes framing bits (F)` – Boolean
3. `start bit value (T)` – Boolean

Parentheses ( ) indicate default value; square brackets [ ] designate units.

[16] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[17] This content is available online at <http://cnx.org/content/m18629/1.1/>.
[18] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[19] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.1.2.3.3 Outputs (Indicators)

1. `text out` – string
2. `framing error?` – 1-D Boolean array

5.1.2.3.4 Required Behavior

- The bitstream must follow the indexing schemes imposed by the LabVIEW built-in nodes "Boolean Array to Number" and "Byte Array to String."
- When `includes framing bits` is true, the start bit leading the 8-element Boolean subarray (a single text character) and the trailing stop bit will be removed from the bitstream before converting to text. In addition, the start bit will be compared to the expected value `start bit value`; the same holds true for the stop bit, which is assumed to be the complement of `start bit value`. Any mismatch is to be flagged as a framing error by setting `framing error?` true.

5.1.2.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[20] to learn the mechanics of subVIs.

Refer to the Figure 5.5 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.5: [video] LabVIEW coding tips and techniques for util_BitstreamToText.vi

---

5.1.3 Channel Noise

5.1.3.1 util_BinarySymmetricChannel.vi[21]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[22] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.6

NOTE: Visit LabVIEW Setup[23] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[20] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[21] This content is available online at <http://cnx.org/content/m18537/1.1/>.
[22] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[23] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.1.3.1.1 LabVIEW SubVI: util_BinarySymmetricChannel.vi

- **Description:** A **binary symmetric channel** (BSC) simulates a digital communication channel with a simple probabilistic model. The simple model makes two assumptions: (1) bit errors occur independently for each bit transmitted through the channel, and (2) a bit error transforming a 0 to a 1 is equally likely as an error transforming a 1 to a 0, i.e., the channel is symmetric. Create a subVI that accepts an input bitstream to produce an output bitstream in which errors are inserted according to a specified bit error rate (or probability of error).
- **Category:** General-purpose utility ("util" prefix)

5.1.3.1.2 Inputs (Controls)

1. `bitstream in` – 1-D Boolean array
2. `bit error rate` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.3.1.3 Outputs (Indicators)

1. `bitstream out` – 1-D Boolean array

5.1.3.1.4 Required Behavior

- Introduce bit errors into the bitstream according to the specified bit error rate.
- Bit errors from one bit to the next are independent of one another.
- Transforming a "True" to a "False" is equally likely to transforming a "False" to a "True."

5.1.3.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[24] to learn the mechanics of subVIs.

Refer to the Figure 5.6 screencast video for LabVIEW coding tips and techniques specific to this subVI. Two distinct approaches are demonstrated, one based on the "Random Number (0-1)" built-in LabVIEW node and another based on the "Bernoulli Noise" built-in subVI.

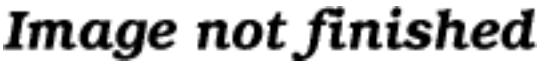


Figure 5.6: [video] LabVIEW coding tips and techniques for util_BinarySymmetricChannel.vi

5.1.3.2 util_AWGNchannel_PtByPt.vi[25]

[24] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[25] This content is available online at <http://cnx.org/content/m18515/1.1/>.

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[26] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.7

NOTE: Visit LabVIEW Setup[27] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.3.2.1 LabVIEW SubVI: util_AWGNchannel_PtByPt.vi

- **Description:** Simulate an additive white Gaussian noise-impaired channel. This subVI works on a point-by-point basis and is intended to operated within a for-loop or while-loop structure.
- **Category:** General-purpose utility ("util" prefix)

5.1.3.2.2 Inputs (Controls)

1. `signal in` – DBL
2. `Eb, energy per bit [J/bit] (1)` – DBL
3. `Eb/No, SNR per bit [dB] (10)` – DBL
4. `fs [Hz] (1000)` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.3.2.3 Outputs (Indicators)

1. `signal out` – DBL
2. `sigma` – DBL
3. `Eb/No` – DBL

5.1.3.2.4 Required Behavior

- A new sample of a Gaussian white noise process is determined and added to `signal in` each time the subVI runs to produce `signal out`.
- The standard deviation parameter of the Gaussian white noise generator is calculated as $\sigma = \sqrt{\frac{E_b f_s}{2 \cdot 10^{E_b/N_0[\mathrm{dB}]/10}}}$ and is reported by the `sigma` indicator.
- `Eb/No` indicates the SNR per bit converted from the decibel form of the corresponding control.

The equation used to convert Eb/No to standard deviation is derived in the screencast video of Figure 5.7.

[26] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[27] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

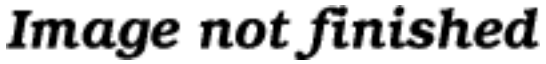


Figure 5.7: [video] Convert Eb/No to standard deviation

5.1.3.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[28] to learn the mechanics of subVIs.

Refer to the Figure 5.8 screencast video for LabVIEW coding tips and techniques specific to this subVI.

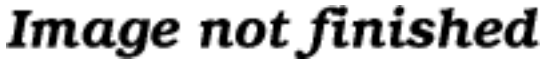


Figure 5.8: [video] LabVIEW coding tips and techniques for util_AWGNchannel_PtByPt.vi

5.1.4 Performance Metrics

5.1.4.1 util_MeasureBER.vi[29]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[30] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.8

NOTE: Visit LabVIEW Setup[31] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.4.1.1 LabVIEW SubVI: util_MeasureBER.vi

- **Description:** Measure the **bit error rate** (**BER**) between two bitstreams. This subVI is commonly used to compare a transmitted bitstream to a received bitstream after passing through a noisy channel.
- **Category:** General-purpose utility ("util" prefix)

[28] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[29] This content is available online at <http://cnx.org/content/m18547/1.1/>.
[30] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[31] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.1.4.1.2 Inputs (Controls)

1. `bitstream A` – 1-D Boolean array
2. `bitstream B` – 1-D Boolean array

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.4.1.3 Outputs (Indicators)

1. `error bitstream` – 1-D Boolean array
2. `BER, bit error rate` – DBL
3. `error count` – I32
4. `array size mismatch` – Boolean

5.1.4.1.4 Required Behavior

- A **bit error** is defined as any discrepancy between `bitstream A` and `bitstream B` at each array index. The output `error count` indicates the total number of bit errors.
- The `error bitstream` output indicates `T` (true) at each index value where a bit error occurred. Absence of bit errors is indicated by `F` (false).
- The bit error rate (BER) is calculated as the total number of bit errors divided by the bitstream length. The bit error rate is reported as `NaN` ("Not a Number") if the two inbound bitstreams do not have the same length.
- The output `array size mismatch` will be active (`T`) when the two bitstreams do not have the same length.

5.1.4.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[32] to learn the mechanics of subVIs.

Refer to the Figure 5.9 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.9: [video] LabVIEW coding tips and techniques for util_MeasureBER.vi

---

5.1.5 Miscellaneous

5.1.5.1 util_EdgeDetector.vi[33]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[34] module for tutorials and documentation that will help you:

continued on next page

[32] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

[33] This content is available online at <http://cnx.org/content/m18606/1.1/>.

• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.9

NOTE: Visit LabVIEW Setup[35] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.5.1.1 LabVIEW SubVI: util_EdgeDetector.vi

- **Description:** Detect edges (transitions) in a bitstream, and indicate rising edge, falling edge, or either edge as three distinct outputs.
- **Category:** General-purpose utility ("util" prefix)

5.1.5.1.2 Inputs (Controls)

1. `bitstream in` – 1-D array of Boolean

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.5.1.3 Outputs (Indicators)

1. `rising edge` – 1-D array of Boolean
2. `falling edge` – 1-D array of Boolean
3. `either edge` – 1-D array of Boolean

5.1.5.1.4 Required Behavior

- Each of the three Boolean output indicators is an array of the same size as the input bitstream.
- `rising edge` is `T` whenever the the input bitstream sequence changes from `F` to `T`.
- `falling edge` is `T` whenever the the input bitstream sequence changes from `T` to `F`.
- `either edge` is the logical "OR" of the previous two indicator outputs.

5.1.5.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[36] to learn the mechanics of subVIs.

Refer to the Figure 5.10 screencast video for LabVIEW coding tips and techniques specific to this subVI.

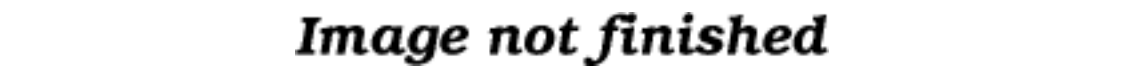


Figure 5.10: [video] LabVIEW coding tips and techniques for util_EdgeDetector.vi

[34] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[35] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[36] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

5.1.5.2 util_GetAudio.vi[37]

[37] This content is available online at <http://cnx.org/content/m18532/1.1/>.

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[38] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.10

NOTE: Visit LabVIEW Setup[39] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.5.2.1 LabVIEW SubVI: util_GetAudio.vi

- **Description:** Retrieve audio from a .wav file, specifically the left channel, and return as a monaural waveform.
- **Category:** General-purpose utility ("util" prefix)

5.1.5.2.2 Inputs (Controls)

1. `path` – file path

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.5.2.3 Outputs (Indicators)

1. `audio` – waveform

5.1.5.2.4 Required Behavior

- Retrieve a .wav audio file which can be either monaural (single channel) or stereo (two-channel), extract the left channel, and return as a waveform data type.

5.1.5.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[40] to learn the mechanics of subVIs.

Refer to the Figure 5.11 screencast video for LabVIEW coding tips and techniques specific to this subVI.

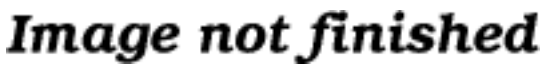


Figure 5.11: [video] LabVIEW coding tips and techniques for util_GetAudio.vi

[38] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[39] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[40] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

5.1.5.3 util_Qfunction.vi[41]

[41] This content is available online at <http://cnx.org/content/m18545/1.1/>.

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[42] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.11

NOTE: Visit LabVIEW Setup[43] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.1.5.3.1 LabVIEW SubVI: util_Qfunction.vi

- **Description:** Evaluate the Q-function Q(x), the area under the right-side tail of a zero-mean unit-variance Gaussian probability density function from x to positive infinity. The Q-function is widely used in communication systems for probability-of-error calculations.
- **Category:** General-purpose utility ("util" prefix)

5.1.5.3.2 Inputs (Controls)

1. `x` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.1.5.3.3 Outputs (Indicators)

1. `Q(x)` – DBL

5.1.5.3.4 Required Behavior

- Given the parameter value `x`, return the area under the right-side tail of a zero-mean unit-variance Gaussian probability density function from x to positive infinity.

5.1.5.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[44] to learn the mechanics of subVIs.

Refer to the Figure 5.12 screencast video for LabVIEW coding tips and techniques specific to this subVI.

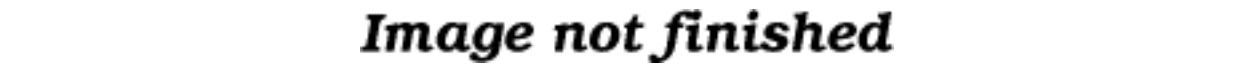


Figure 5.12: [video] LabVIEW coding tips and techniques for util_Qfunction.vi

[42] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[43] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[44] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

5.2 Baseband Modulation and Pulse Amplitude Modulation (PAM)

5.2.1 Pulse Shapes

5.2.1.1 pam_RaisedCosinePulse.vi[45]

[45] This content is available online at <http://cnx.org/content/m18566/1.1/>.

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[46] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.12

NOTE: Visit LabVIEW Setup[47] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.2.1.1.1 LabVIEW SubVI: pam_RaisedCosinePulse.vi

- **Description:** Create a raised cosine pulse shape suitable for a pulse amplitude modulation (PAM) transmitter.
- **Category:** Pulse amplitude modulation (PAM) ("pam" prefix)

5.2.1.1.2 Inputs (Controls)

1. `Tb, bit interval (0.01) [s]` – DBL
2. `alpha, excess bandwidth factor (0.5)` – DBL
3. `N, bit intervals for support (4)` – DBL
4. `fs, sampling frequency (1000) [Hz]` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.2.1.1.3 Outputs (Indicators)

1. `pulse shape` – 1-D DBL array

5.2.1.1.4 Required Behavior

- "pulse shape" is an array containing the raised cosine pulse shape defined by the equation

$$p(t) = \text{sinc}(2B_0 t)\left(\frac{\cos(2\pi\alpha B_0 t)}{1 - 16(\alpha B_0 t)^2}\right)$$

- B_0 = Nyquist bandwidth, the minimum possible transmit bandwidth achieved by a sinc pulse
- $B_0 = \frac{1}{2T_b}$, where Tb is the bit interval
- α = roll-off factor (also called excess bandwidth factor), $0 \leq \alpha \leq 1$ (alpha = 0 creates an unmodified sinc pulse, and alpha = 1 creates a fully damped sinc pulse with twice the Nyquist bandwidth).
- The "alpha" control value must be limited to the range 0 to 1 and be incrementable by steps of 0.1.

[46] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[47] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

The raised cosine pulse shape is fundamental to digital communication systems. Its name derives from its frequency-domain shape. Refer to the Figure 5.13 screencast video to learn more about the raised cosine pulse.

Image not finished

Figure 5.13: [video] Explanation of raised cosine pulse

5.2.1.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[48] to learn the mechanics of subVIs.

Refer to the Figure 5.14 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.14: [video] LabVIEW coding tips and techniques for pam_RaisedCosinePulse.vi

---

5.2.1.2 pam_RectanglePulse.vi[49]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[50] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.13

NOTE: Visit LabVIEW Setup[51] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

---

[48] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[49] This content is available online at <http://cnx.org/content/m18454/1.1/>.
[50] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[51] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.2.1.2.1 LabVIEW SubVI: pam_RectanglePulse.vi

- **Description:** Create a rectangle pulse shape suitable for pulse amplitude modulation (PAM) transmitters.
- **Category:** Pulse amplitude modulation (PAM) ("pam" prefix)

5.2.1.2.2 Inputs (Controls)

1. `Tb, bit interval [s]` – DBL
2. `fs, sampling frequency [Hz]` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.2.1.2.3 Outputs (Indicators)

1. `pulse shape` – 1-D DBL array

5.2.1.2.4 Required Behavior

- `pulse shape` is an array of constant unit value.
- The array length is one bit interval `Tb` times the sampling frequency `fs`.

5.2.1.2.5 LabVIEW Coding Tips

Review the LabVIEW help page for "Programming | Array | Initialize Array."

5.2.1.3 pam_ManchesterPulse.vi[52]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[53] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.14

NOTE: Visit LabVIEW Setup[54] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[52] This content is available online at <http://cnx.org/content/m18466/1.1/>.
[53] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[54] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.2.1.3.1 LabVIEW SubVI: pam_ManchesterPulse.vi

- **Description:** Create a prototype Manchester pulse shape. The Manchester pulse is a polar NRZ (non return to zero) that is 1 during the first half of the bit interval and -1 during the second half of the bit interval.
- **Category:** Pulse amplitude modulation ("pam" prefix)

5.2.1.3.2 Inputs (Controls)

1. `Tb, bit interval [s] (1)` – DBL
2. `fs, sampling frequency [Hz] (10)` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.2.1.3.3 Outputs (Indicators)

1. `pulse shape` – 1-D array of DBL

5.2.1.3.4 Required Behavior

- `pulse shape` contains `Tb` times `fs` sample points.
- The first half of the output array contains +1, while the second half of the output array contains -1.

5.2.1.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[55] to learn the mechanics of subVIs.

Refer to the Figure 5.15 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.15: [video] LabVIEW coding tips and techniques for pam_ManchesterPulse.vi

---

5.2.2 Transmitter Components

5.2.2.1 pam_SignalPointMapper.vi[56]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[57] module for tutorials and documentation that will help you:

continued on next page

[55] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

[56] This content is available online at <http://cnx.org/content/m18570/1.1/>.

• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.15

NOTE: Visit LabVIEW Setup[58] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.2.2.1.1 LabVIEW SubVI: pam_SignalPointMapper.vi

- **Description:** Map a bitstream onto two different levels.
- **Category:** Pulse amplitude modulation (PAM) ("pam" prefix)

5.2.2.1.2 Inputs (Controls)

1. `bitstream in` – 1-D Boolean array
2. `T level (1)` – DBL
3. `F level (0)` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.2.2.1.3 Outputs (Indicators)

1. `signal level` – 1-D DBL array

5.2.2.1.4 Required Behavior

- Each element of the bitstream maps to one of two possible signal levels: `T` values convert to the value specified by `T level` and `F` values convert to the value specified by `F level`.

5.2.2.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[59] to learn the mechanics of subVIs.

Refer to the Figure 5.16 screencast video for LabVIEW coding tips and techniques specific to this subVI.

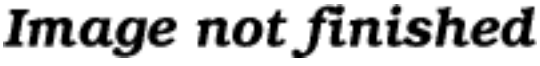


Figure 5.16: [video] LabVIEW coding tips and techniques for pam_SignalPointMapper.vi

[57] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[58] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[59] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

5.2.2.2 pam_TransmitFilter.vi[60]

[60] This content is available online at <http://cnx.org/content/m18472/1.1/>.

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[61] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.16

NOTE: Visit LabVIEW Setup[62] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.2.2.2.1 LabVIEW SubVI: pam_TransmitFilter.vi

- **Description:** Convert a sequence (array) of signal levels to a signal waveform with a user-defined pulse shape. Each element of the signal levels array generates one "analog" pulse (a sampled-value discrete-time waveform). This device is commonly called a "transmit filter" since it is implemented by an impulse train driving an FIR filter.
- **Category:** Pulse amplitude modulation (PAM) ("pam" prefix)

5.2.2.2.2 Inputs (Controls)

1. `signal levels in` – 1-D DBL array
2. `pulse shape` – 1-D DBL array
3. `Tb, bit interval [s]` – DBL
4. `fs, sampling frequency [Hz]` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.2.2.2.3 Outputs (Indicators)

1. `waveform out` – waveform
2. `samples per bit` – 1-D DBL array

5.2.2.2.4 Required Behavior

- Each element of `signal levels in` indicates the amplitude of a user-defined `pulse shape`, which is assumed to have a unit amplitude.
- Pulses are generated once each bit interval defined by `Tb`. The finished waveform `waveform out` is the superposition (sum) of all individual time-shifted pulse waveforms.
- The prototype waveform `waveform out` may extend beyond a single bit interval.

[61] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[62] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.2.2.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[63] to learn the mechanics of subVIs.

Refer to the Figure 5.17 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.17: [video] LabVIEW coding tips and techniques for pam_TransmitFilter.vi

---

5.2.2.3 pam_TransmitSync.vi[64]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[65] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.17

NOTE: Visit LabVIEW Setup[66] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.2.2.3.1 LabVIEW SubVI: pam_TransmitSync.vi

- **Description:** Create transmitter sync pulses to indicate the start and end of a bit interval. Also report the samples per bit interval.
- **Category:** Pulse amplitude modulation ("pam" prefix)

5.2.2.3.2 Inputs (Controls)

1. `message length (10)` – I32
2. `Tb, bit interval [s] (1)` – DBL
3. `fs, sampling frequency [Hz] (10)` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

[63] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[64] This content is available online at <http://cnx.org/content/m18478/1.1/>.
[65] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[66] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.2.2.3.3 Outputs (Indicators)

1. `start bit interval` – 1-D array of Boolean
2. `end bit interval` – 1-D array of Boolean
3. `samples per bit interval` – I32

5.2.2.3.4 Required Behavior

- `samples per bit interval` indicates `Tb` times `fs` sample points.
- `start bit interval` and `end bit interval` each contain `message length` times `samples per bit interval` elements in which `T` indicates the boundary of a bit interval.
- The first element of `start bit interval` is `T`. The remaining elements for the bit interval are `F`.
- `end bit interval` is similar to `start bit interval`, except the `T` element occurs at the end of a bit interval.

5.2.2.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[67] to learn the mechanics of subVIs.

Refer to the Figure 5.18 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.18: [video] LabVIEW coding tips and techniques for pam_TransmitSync.vi

---

5.3 Bandpass Modulation

5.3.1 bpm_EnvelopeDetector.vi[68]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[69] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.18

NOTE: Visit LabVIEW Setup[70] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[67] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[68] This content is available online at <http://cnx.org/content/m18420/1.1/>.
[69] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[70] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.3.1.1 LabVIEW SubVI: bpm_EnvelopeDetector.vi

- **Description:** Demodulate an amplitude shift keyed (ASK) signal using envelope detection, a type of noncoherent detection. The envelope detector is a "square-law" device (actually an absolute value operator) followed by a lowpass filter.
- **Category:** Bandpass modulation ("bpm" prefix)

5.3.1.2 Inputs (Controls)

1. `modulated signal in` – waveform
2. `LPF corner frequency [Hz] (100)` – DBL
3. `LPF order (2)` – I32

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.3.1.3 Outputs (Indicators)

1. `baseband signal out` – waveform

5.3.1.4 Required Behavior

- The absolute value of `modulated signal in` is filtered by a Butterworth lowpass filter to produce `baseband signal out`
- The Butterworth filter corner frequency and order may be adjusted.

5.3.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[71] to learn the mechanics of subVIs.

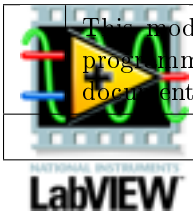
Refer to the Figure 5.19 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.19: [video] LabVIEW coding tips and techniques for bpm_EnvelopeDetector.vi

---

5.3.2 bpm_ProductModulator.vi[72]



This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[73] module for tutorials and documentation that will help you:

continued on next page

[71] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

[72] This content is available online at <http://cnx.org/content/m18556/1.1/>.

• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.19

NOTE: Visit LabVIEW Setup[74] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.3.2.1 LabVIEW SubVI: bpm_ProductModulator.vi

- **Description:** Modulate a baseband signal by a sinusoidal carrier wave that has unit energy as measured over a single bit interval.
- **Category:** Bandpass modulation ("bpm" prefix)

5.3.2.2 Inputs (Controls)

1. `waveform in` – waveform
2. `Tb, bit interval [s]` – DBL
3. `fc, carrier frequency [Hz]` – DBL
4. `fs, sampling frequency [Hz]` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.3.2.3 Outputs (Indicators)

1. `waveform out` – waveform
2. `carrier` – waveform

5.3.2.4 Required Behavior

- `waveform out` is the product of `waveform in` and the sinusoidal carrier signal

  $$A_c \cos(2\pi f_c t)$$

  , where

  $$A_c$$

  is the carrier amplitude and

  $$f_c$$

  is the carrier frequency in Hz.
- The carrier sinusoid amplitude must be

  $$A_c = \sqrt{\frac{2}{T_b}}$$

  in order to achieve the "unit energy per bit interval" criterion. The Figure 5.20 screencast video explains the origin of this equation.

[73] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[74] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

- `carrier` is the output of the sinusoidal oscillator used to modulate the inbound signal.

Image not finished

Figure 5.20: [video] Explanation of the "unit energy per bit" amplitude equation

---

5.3.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[75] to learn the mechanics of subVIs.

Refer to the Figure 5.21 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.21: [video] LabVIEW coding tips and techniques for bpm_ProductModulator.vi

---

5.3.3 bpm_ReceiverFilter.vi[76]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[77] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.20

NOTE: Visit LabVIEW Setup[78] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[75] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[76] This content is available online at <http://cnx.org/content/m18436/1.1/>.
[77] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[78] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.3.3.1 LabVIEW SubVI: bpm_ReceiverFilter.vi

- **Description:** Remove out-of-band signals at the front end of a receiver using a bandpass filter tuned to the carrier frequency and with a bandwidth that matches the bandwidth of the transmitted signal.
- **Category:** Bandpass modulation ("bpm" prefix)

5.3.3.2 Inputs (Controls)

1. `signal in` – waveform
2. `center frequency [Hz] (1000)` – DBL
3. `bandwidth [Hz] (100)` – DBL
4. `order (10)` – I32

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.3.3.3 Outputs (Indicators)

1. `signal out` – waveform

5.3.3.4 Required Behavior

- `signal in` is filtered by an elliptic bandpass filter to produce `signal out`
- The elliptic filter characteristics (center frequency, bandwidth, and order) may be adjusted.

5.3.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[79] to learn the mechanics of subVIs.

Refer to the Figure 5.22 screencast video for LabVIEW coding tips and techniques specific to this subVI.

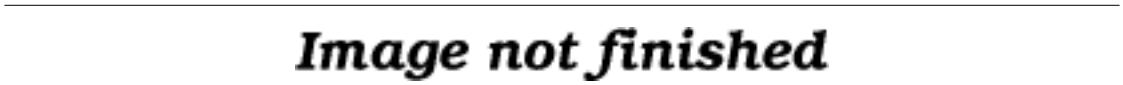


Figure 5.22: [video] LabVIEW coding tips and techniques for bpm_ReceiverFilter.vi

[79] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

5.4 Demodulation and Bitstream Regeneration

5.4.1 Synchronization

5.4.1.1 regen_BitClock.vi[80]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[81] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.21

NOTE: Visit LabVIEW Setup[82] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.4.1.1.1 LabVIEW SubVI: regen_BitClock.vi

- **Description:** Create a bit clock signal based on a free-running oscillator with a sync input. The bit clock signal is a square wave oscillating at a nominal frequency. The oscillator phase resets when the synchronizing input pulse is active.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.1.1.2 Inputs (Controls)

1. `restart bit interval` – 1-D Boolean array
2. `nominal frequency [Hz]` – DBL
3. `fs [Hz]` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.4.1.1.3 Outputs (Indicators)

1. `bit clock` – 1-D Boolean array

5.4.1.1.4 Required Behavior

- `bit clock` is the output of a square wave oscillator represented as a Boolean array. The nominal oscillation frequency is determined by the inputs `nominal frequency` in Hz and the system sampling frequency `fs`, also in Hz.
- The `bit clock` output array is the same length as the input array `restart bit interval`.
- The oscillator phase resets anytime that `restart bit interval` is `T`, thereby synchronizing the bit clock to the beginning of a bit interval as detected by another system.

[80] This content is available online at <http://cnx.org/content/m18612/1.1/>.
[81] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[82] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.4.1.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[83] to learn the mechanics of subVIs.

Refer to the Figure 5.23 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.23: [video] LabVIEW coding tips and techniques for regen_BitClock.vi

---

5.4.1.2 regen_BitSync.vi[84]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[85] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.22

NOTE: Visit LabVIEW Setup[86] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.4.1.2.1 LabVIEW SubVI: regen_BitSync.vi

- **Description:** Recover a bitstream synchronization signal from a polar NRZ baseband signal as an array of sampling instants.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.1.2.2 Inputs (Controls)

1. `signal in` – waveform
2. `bit rate [Hz]` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

[83] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[84] This content is available online at <http://cnx.org/content/m18627/1.1/>.
[85] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[86] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.4.1.2.3 Outputs (Indicators)

1. `sampling instants` – Boolean 1-D array
2. `intermediate signals` – cluster of three waveforms: (1) absolute value, (2) bandpass filter, and (3) thresholded BPF

5.4.1.2.4 Required Behavior

- Pass `signal in` through a "square-law" device (square the waveform), and then through a narrowband bandpass filter tuned to the bit rate.
- Detect locations of negative-going zero crossings of the bandpass filter output and return as the Boolean array `sampling instants`.
- Bundle the intermediate signals (square-law device output, bandpass filter output, and thresholded bandpass filter output) as the cluster `intermediate signals`.

5.4.1.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[87] to learn the mechanics of subVIs.

Refer to the Figure 5.24 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.24: [video] LabVIEW coding tips and techniques for regen_BitSync.vi

---

5.4.1.3 regen_FrameSync.vi[88]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[89] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.23

NOTE: Visit LabVIEW Setup[90] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[87] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[88] This content is available online at <http://cnx.org/content/m18576/1.1/>.
[89] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[90] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.4.1.3.1 LabVIEW SubVI: regen_FrameSync.vi

- **Description:** Establish frame sync on a message containing a standard preamble, and then return the message portion of the bitstream. The bitstream must satisfy the following requirement: (1) Message frame size = 10 bits (start bit, 8-bit character, and stop bit), (2) start bit = `F`, stop bit = `T` , and (3) message is preceded by one frame containing the 8-bit value 0xFF. Requirement (3) can equivalently be restated as the preamble must end with 9 consecutive `T` values.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.1.3.2 Inputs (Controls)

1. `bitstream in` – 1-D Boolean array

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.4.1.3.3 Outputs (Indicators)

1. `bitstream out` – 1-D Boolean array
2. `message detected?` – Boolean

5.4.1.3.4 Required Behavior

- Search `bitstream in` for 9 consecutive `T` values, and return the remaining array as `bitstream out`. Return an empty array if the required pattern is not found.
- Set `message detected?` to `T` if the required pattern is found.

5.4.1.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[91] to learn the mechanics of subVIs.

Refer to the Figure 5.25 screencast video for LabVIEW coding tips and techniques specific to this subVI.

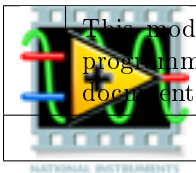
Image not finished

Figure 5.25: [video] LabVIEW coding tips and techniques for regen_FrameSync.vi

---

5.4.2 Preamble Processing

5.4.2.1 regen_ExtractPreamble.vi[92]



This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[93] module for tutorials and documentation that will help you:

continued on next page

[91] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

[92] This content is available online at <http://cnx.org/content/m18585/1.1/>.

• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.24

NOTE: Visit LabVIEW Setup[94] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.4.2.1.1 LabVIEW SubVI: regen_ExtractPreamble.vi

- **Description:** Detect and extract the preamble from a baseband signal. A preset number of alternating 1s and 0s (also designated as **marks** and **spaces**) typically starts the preamble to "wake up" the receiver's carrier sync and bit sync subsystems. This subVI assumes the received signal is quiet (nominally zero) prior to the preamble.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.2.1.2 Inputs (Controls)

1. `signal in` – waveform
2. `Tb [s]` – DBL
3. `bit intervals to skip (4)` – I32
4. `bit intervals to keep (32)` – I32
5. `threshold (0.1)` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.4.2.1.3 Outputs (Indicators)

1. `preamble out` – waveform
2. `preamble detected?` – Boolean

5.4.2.1.4 Required Behavior

- `signal in` is scanned from the beginning to detect when the signal amplitude exceeds `threshold`. If the input signal never exceeds the threshold, `preamble out` returns an empty waveform and `preamble detected?` returns `F`.
- Once a valid threshold crossing is detected, `preamble out` extracts a portion of `signal in` of duration `bit intervals to keep` times the bit interval `Tb`; the extracted signal begins at the location of the first threshold crossing plus `bit intervals to skip` times the bit interval. The Boolean indicator `preamble detected?` is set to `T`.

[93] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[94] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.4.2.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[95] to learn the mechanics of subVIs.

Refer to the Figure 5.26 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.26: [video] LabVIEW coding tips and techniques for regen_ExtractPreamble.vi

---

5.4.2.2 regen_NormalizeToPreamble.vi[96]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[97] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.25

NOTE: Visit LabVIEW Setup[98] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.4.2.2.1 LabVIEW SubVI: regen_NormalizeToPreamble.vi

- **Description:** Normalize a received baseband signal according to the DC and RMS values of the preamble portion of the signal. The preamble is assumed to be a region of alternating 1s and 0s (marks and spaces) that approximates a sinusoid. The DC offset and RMS values of the preamble are measured, and then used to normalize the entire signal.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.2.2.2 Inputs (Controls)

1. `signal in` – waveform
2. `preamble` – waveform

Parentheses ( ) indicate default value; square brackets [ ] designate units.

[95] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[96] This content is available online at <http://cnx.org/content/m18483/1.1/>.
[97] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[98] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.4.2.2.3 Outputs (Indicators)

1. `normalized signal out` – waveform
2. `preamble DC value` – DBL
3. `preamble RMS value` – DBL

5.4.2.2.4 Required Behavior

- Measure the DC (average) value of `preamble`.
- Measure the RMS (root mean square) value of `preamble`.
- Produce `normalized signal out` by (1) subtracting the DC value from `signal in`, (2) dividing by the RMS value, and (3) multiplying by the square root of 2. The resulting signal has approximately zero average value and lies approximately in the range ±.
- Return the measured DC and RMS values as indicators.

5.4.2.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[99] to learn the mechanics of subVIs.

Refer to the Figure 5.27 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.27: [video] LabVIEW coding tips and techniques for regen_NormalizeToPreamble.vi

---

5.4.3 Coherent Detection

5.4.3.1 regen_Correlator.vi[100]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[101] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.26

NOTE: Visit LabVIEW Setup[102] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[99] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[100] This content is available online at <http://cnx.org/content/m18579/1.1/>.
[101] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[102] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.4.3.1.1 LabVIEW SubVI: regen_Correlator.vi

- **Description:** Demodulate a pulse-amplitude modulated (PAM) signal using a correlator. The correlator multiples the received signal by the same pulse shape used by the transmitter and integrates the product over the bit interval. A Boolean control indicates when to clear the integrator and restart the pulse. This subVI is intended for point-by-point operation within a repeating structure such as a for-loop or while-loop.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.3.1.2 Inputs (Controls)

1. `signal in` – DBL
2. `fs [Hz]` – DBL
3. `start integrating` – Boolean
4. `pulse` – 1-D array of DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.4.3.1.3 Outputs (Indicators)

1. `signal out` – DBL

5.4.3.1.4 Required Behavior

- `signal out` is the time integral of the product of `signal in` and the pulse shape `pulse`. The integration is computed on a point-by-point basis, so each call to the subVI calculates only a single output value. The integrator output and position (index) within the pulse signal is preserved from one subVI call to the next.
- When `start integrating` is `T` the integrator is reset and the pulse signal index is reset to zero, i.e., the beginning of the pulse shape array.

5.4.3.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[103] to learn the mechanics of subVIs.

Refer to the Figure 5.28 screencast video for LabVIEW coding tips and techniques specific to this subVI.

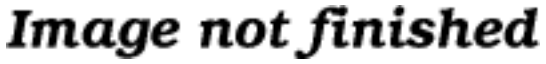


Figure 5.28: [video] LabVIEW coding tips and techniques for regen_Correlator.vi

5.4.4 Sampling

5.4.4.1 regen_SampleHold.vi[104]

[103] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

[104] This content is available online at <http://cnx.org/content/m18621/1.1/>.

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[105] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.27

NOTE: Visit LabVIEW Setup[106] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.4.4.1.1 LabVIEW SubVI: regen_SampleHold.vi

- **Description:** Sample a signal on demand and hold the signal value across multiple calls to the subVI. This subVI is intended for point-by-point processing within a for-loop or while-loop structure.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.4.1.2 Inputs (Controls)

1. `signal in` – DBL
2. `sample now` – Boolean

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.4.4.1.3 Outputs (Indicators)

1. `signal out` – DBL

5.4.4.1.4 Required Behavior

- `signal out` takes on one of two possible values: if `sample now` is `T` the output value is `signal in`, otherwise it is the value of `signal out` from the previous call to the subVI.

5.4.4.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[107] to learn the mechanics of subVIs.

Refer to the Figure 5.29 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.29: [video] LabVIEW coding tips and techniques for regen_SampleHold.vi

---

[105] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[106] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[107] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

5.4.4.2 regen_Sampler.vi[108]

[108] This content is available online at <http://cnx.org/content/m18593/1.1/>.

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[109] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.28

NOTE: Visit LabVIEW Setup[110] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.4.4.2.1 LabVIEW SubVI: regen_Sampler.vi

- **Description:** Sample a signal at selected instants in time. The signal input is a discrete-time sampled signal that represents an "analog" signaling waveform. The sampling instants are indicated by a Boolean array of the same length as the signal input. A user-defined delay can be applied to shift the sampling instants by a fixed amount.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.4.2.2 Inputs (Controls)

1. `signal in` – waveform
2. `sampling instants` – 1-D Boolean array
3. `delay [samples] (0)` – I32

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.4.4.2.3 Outputs (Indicators)

1. `sampling signal out` – 1-D array of DBL
2. `actual sampling instants` – 1-D Boolean array

5.4.4.2.4 Required Behavior

- `sampled signal out` contains the subset of values from `signal in` that match the index values of the `T`-valued elements of `sampling instants`. `sampling instants` is assumed to be of the same length as `signal in`. Expressed another way, the Boolean array `sampling instants` contains `T` (true) values at each time that `signal in` is to be sampled. The output `sampled signal out` therefore contains the resulting samples.
- The `delay` value adds a constant shift to the position of the sampling instants. The delay amount defaults to zero; a positive value delays the sampling instants and a negative value advances the sampling instants.
- The `actual sampling instants` is a copy of the input `sampling instants` with the delay value applied.

[109] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[110] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.4.4.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[111] to learn the mechanics of subVIs.

Refer to the Figure 5.30 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.30: [video] LabVIEW coding tips and techniques for regen_Sampler.vi

---

5.4.4.3 regen_BitstreamBuffer.vi[112]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[113] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.29

NOTE: Visit LabVIEW Setup[114] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.4.4.3.1 LabVIEW SubVI: regen_BitstreamBuffer.vi

- **Description:** Build a bitstream by accumulating bits on demand. This subVI is intended for point-by-point processing within a for-loop or while-loop structure.
- **Category:** Bitstream regeneration ("regen" prefix)

5.4.4.3.2 Inputs (Controls)

1. `bit in` – Boolean
2. `save bit` – Boolean

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.4.4.3.3 Outputs (Indicators)

1. `bitstream out` – 1-D array of Boolean

---

[111] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[112] This content is available online at <http://cnx.org/content/m18494/1.1/>.
[113] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[114] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.4.4.3.4 Required Behavior

- The `bitstream out` array is empty on the first call to the subVI.
- The `bitstream out` array values are retained from one subVI call to the next.
- When `save bit` is `T` the `bit in` value is appended to the `bitstream out` array, otherwise the array returned unchanged.

5.4.4.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[115] to learn the mechanics of subVIs.

Refer to the Figure 5.31 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.31: [video] LabVIEW coding tips and techniques for regen_BitstreamBuffer.vi

---

5.5 Hamming Block Coding

5.5.1 hamming_DetectorCorrector.vi[116]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[117] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.30

NOTE: Visit LabVIEW Setup[118] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.5.1.1 LabVIEW SubVI: hamming_DetectorCorrector.vi

- **Description:** Implement (n,k) Hamming linear block code error detection and correction using the "table lookup syndrome decoder" method. The syndrome calculated from a received stream of code-words is used as an index into the syndrome table to retrieve the most-likely error pattern, which

[115] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[116] This content is available online at <http://cnx.org/content/m18427/1.1/>.
[117] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[118] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

subsequently is added (modulo-2 addition) to the received codeword to generate the corrected codeword output. Checkbits may optionally be removed from the output wordstream. Detected errors (single and double bit errors) are indicated separately.

- **Category:** Hamming (n,k) block code ("hamming" prefix)

5.5.1.2 Inputs (Controls)

1. `uncorrected wordstream` – Boolean 2-D array
2. `syndrome` – Boolean 2-D array
3. `syndrome table` – Boolean 2-D array
4. `remove checkbits (F)` – Boolean

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.5.1.3 Outputs (Indicators)

1. `corrected wordstream` – Boolean 2-D array
2. `error detected` – Boolean 1-D array

5.5.1.4 Required Behavior

Refer to the description above.

5.5.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[119] to learn the mechanics of subVIs.

Refer to the Figure 5.32 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.32: [video] LabVIEW coding tips and techniques for hamming_DetectorCorrector.vi

---

5.5.2 hamming_GeneratorMatrix.vi[120]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[121] module for tutorials and documentation that will help you:

continued on next page

[119] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[120] This content is available online at <http://cnx.org/content/m18563/1.1/>.

• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.31

NOTE: Visit LabVIEW Setup[122] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.5.2.1 LabVIEW SubVI: hamming_HammingCodeParamters.vi

- **Description:** Create the generator matrix (G matrix) for the (n,k) Hamming linear block code, as well as the parity matrix (P matrix), given the number of checkbits "q" and the message length "k".
- **Category:** Hamming (n,k) block code ("hamming" prefix)

5.5.2.2 Inputs (Controls)

1. `q, checkbits (3)` – I32
2. `k, message length (4)` – I32

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.5.2.3 Outputs (Indicators)

1. `G matrix, k by n` – Real Matrix
2. `P matrix, k by q` – Real Matrix

5.5.2.4 Required Behavior

1. "P" is a k by q matrix of q-bit words containing two or more 1s arranged in any order (or, equivalently, the minimum Hamming weight of each row of the "P" matrix is 2).
2. "G" is defined as [I | P], where I is the k by k identity matrix.

5.5.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[123] to learn the mechanics of subVIs.

Refer to the Figure 5.33 screencast video for LabVIEW coding tips and techniques specific to this subVI.

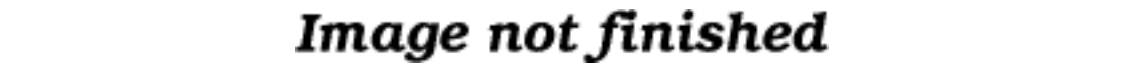


Figure 5.33: [video] LabVIEW coding tips and techniques for hamming_GeneratorMatrix.vi

[121] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[122] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[123] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

5.5.3 hamming_HammingCodeParameters.vi[124]

[124] This content is available online at <http://cnx.org/content/m18441/1.1/>.

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[125] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.32

NOTE: Visit LabVIEW Setup[126] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.5.3.1 LabVIEW SubVI: hamming_HammingCodeParameters.vi

- **Description:** Generate the (n,k) parameters for a Hamming linear block code given the exponent q. Also calculate the coding rate.
- **Category:** Hamming (n,k) block code ("hamming" prefix)

5.5.3.2 Inputs (Controls)

1. `q, checkbits (3)` – I32

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.5.3.3 Outputs (Indicators)

1. `n, codeword length` – I32
2. `k, message length` – I32
3. `Rc, code rate` – DBL

5.5.3.4 Required Behavior

1. $n = 2^q - 1$
2. $k = n - q$
3. $R_c = \frac{k}{n}$

5.5.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[127] to learn the mechanics of subVIs.

Refer to the Figure 5.34 screencast video for LabVIEW coding tips and techniques specific to this subVI.

[125] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[126] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[127] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

Image not finished

Figure 5.34: [video] LabVIEW coding tips and techniques for hamming_HammingCodeParameters.vi

5.5.4 hamming_Mod2MatrixMultiply.vi[128]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[129] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.33

NOTE: Visit LabVIEW Setup[130] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.5.4.1 LabVIEW SubVI: hamming_Mod2MatrixMultiply.vi

- **Description:** Multiply two matrices A and B under modulo-2 arithmetic.
- **Category:** Hamming (n,k) block code ("hamming" prefix)

5.5.4.2 Inputs (Controls)

1. `A` – Real Matrix
2. `B` – Real Matrix

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.5.4.3 Outputs (Indicators)

1. `A*B` – Real Matrix

5.5.4.4 Required Behavior

The subVI produces the matrix product of A and B subject to modulo-2 arithmetic. Since this subVI is intended for use on matrices populated only by the values 0 and 1, multiplication follows the standard arithmetic rules, while sums are computed using modulo-2 addition.

[128] This content is available online at <http://cnx.org/content/m18562/1.1/>.
[129] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[130] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.5.4.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[131] to learn the mechanics of subVIs.

Refer to the Figure 5.35 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.35: [video] LabVIEW coding tips and techniques for hamming_Mod2MatrixMultiply.vi

---

5.5.5 hamming_ParityCheckMatrix.vi[132]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[133] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.34

NOTE: Visit LabVIEW Setup[134] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.5.5.1 LabVIEW SubVI: hamming_ParityCheckMatrix.vi

- **Description:** Create the parity check matrix (H matrix) for the (n,k) Hamming linear block code, given the parity matrix (P matrix). Since the matrix is defined in terms of its transpose, the subVI actually produces HT (the transpose of H).
- **Category:** Hamming (n,k) block code ("hamming" prefix)

5.5.5.2 Inputs (Controls)

1. `P matrix, k by q` – Real Matrix

Parentheses ( ) indicate default value; square brackets [ ] designate units.

[131] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[132] This content is available online at <http://cnx.org/content/m18460/1.1/>.
[133] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[134] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.5.5.3 Outputs (Indicators)

1. `HT matrix, q by n` – Real Matrix

5.5.5.4 Required Behavior

1. The transpose of the parity check matrix is

$$H^T \triangleq \left[\frac{P}{I_q}\right]$$

, where

$$I_q$$

is the q by q identity matrix, and P is the parity matrix associated with Hamming code generator (G) matrix.

5.5.5.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[135] to learn the mechanics of subVIs.

Refer to the Figure 5.36 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.36: [video] LabVIEW coding tips and techniques for hamming_ParityCheckMatrix.vi

---

5.5.6 hamming_SyndromeTable.vi[136]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[137] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.35

NOTE: Visit LabVIEW Setup[138] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[135] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[136] This content is available online at <http://cnx.org/content/m18618/1.1/>.
[137] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[138] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.5.6.1 LabVIEW SubVI: hamming_SyndromeTable.vi

- **Description:** Create the **syndrome table** for the Hamming block code channel decoder. The table contains the most likely error patterns indexed by syndrome number.
- **Category:** Hamming (n,k) block code ("hamming" prefix)

5.5.6.2 Inputs (Controls)

1. `HT matrix, n by q` – Real Matrix

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.5.6.3 Outputs (Indicators)

1. `syndrome table` – Boolean 2-D array

5.5.6.4 Required Behavior

- Determine the number of checkbits "q" from the dimensions of matrix HT.
- "Most likely" error patterns are the no-error pattern and all possible single-bit error patterns.
- Syndrome table is an array of most likely error patterns indexed according to the associated syndrome number. For example, suppose the error pattern `FFTFFFF` was found to produce a syndrome value `TTF`. Retrieving the array value of `syndrome table` at index "3" will then produce the Boolean array `FFTFFFF`. Note that the syndrome pattern is converted to an integer using the built-in LabVIEW node "Boolean Array to Number" which assumes the first element in the Boolean array is the least significant bit.

5.5.6.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[139] to learn the mechanics of subVIs.

Refer to the Figure 5.37 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.37: [video] LabVIEW coding tips and techniques for hamming_SyndromeTable.vi

---

5.6 Speaker - Air - Microphone (SAM) Channel

5.6.1 sam_GrabAudio.vi[140]

[139] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

[140] This content is available online at <http://cnx.org/content/m18499/1.1/>.

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[141] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.36

NOTE: Visit LabVIEW Setup[142] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.6.1.1 LabVIEW SubVI: sam_GrabAudio.vi

- **Description:** Wait for audio to exceed a user-defined threshold, and then record audio for a specified time interval. This subVI depends on sam_ListenForAudio.vi.
- **Category:** Speaker-air-microphone (SAM) channel ("sam" prefix)

5.6.1.2 Inputs (Controls)

1. `duration [s] (1)` – DBL
2. `threshold level (0.1)` – DBL
3. `fs [Hz] (22050)` – DBL
4. `error in (no error)` – error cluster

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.6.1.3 Outputs (Indicators)

1. `waveform out` – waveform
2. `error out` – error cluster

5.6.1.4 Required Behavior

- Use sam_ListenForAudio.vi to continually acquire audio samples in 1024-sample blocks. Once "sam_ListenForAudio" completes execution (i.e., then the audio level exceeds `threshold level`), record audio for `duration` seconds at the sampling frequency `fs`.
- The audio output of sam_ListenForAudio.vi serves as the beginning of the audio signal `waveform out`.
- The sound-card must be cleaned up using "Graphics and Sound | Sound | Input | Sound Input Clear" once the audio has been recorded.

[141] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[142] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.6.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[143] to learn the mechanics of subVIs.

Refer to the Figure 5.38 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.38: [video] LabVIEW coding tips and techniques for sam_GrabAudio.vi

---

5.6.2 sam_GrabAudioDynamic.vi[144]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[145] module for tutorials and documentation that will help you:
	• Apply LabVIEW to Audio Signal Processing
	• Get started with LabVIEW
	• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.37

NOTE: Visit LabVIEW Setup[146] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.6.2.1 LabVIEW SubVI: sam_GrabAudioDynamic.vi

- **Description:** Wait for audio level to exceed a user-defined threshold, and then record audio until the audio level drops below the threshold again or recording duration reaches a maximum value. This subVI depends on sam_ListenForAudio.vi (Section 5.6.3). In addition, sam_GrabAudio.vi (Section 5.6.1) should be constructed before attempting the dynamic-stop version of this module.
- **Category:** Speaker-air-microphone (SAM) channel ("sam" prefix)

5.6.2.2 Inputs (Controls)

1. `frame length [s] (0.1)` – DBL
2. `max duration [s] (10)` – DBL
3. `threshold level (0.1)` – DBL
4. `fs [Hz] (22050)` – DBL
5. `error in (no error)` – error cluster

Parentheses ( ) indicate default value; square brackets [ ] designate units.

[143] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[144] This content is available online at <http://cnx.org/content/m18641/1.1/>.
[145] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[146] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.6.2.3 Outputs (Indicators)

1. `signal out` – waveform
2. `error out` – error cluster

5.6.2.4 Required Behavior

- Use sam_ListenForAudio.vi (Section 5.6.3) to continually acquire audio samples in blocks (frames) of size `frame length`. Once "sam_ListenForAudio" completes execution (i.e., when the audio level exceeds `threshold level`), record and store audio frames until either of two possible conditions occurs: (1) maximum audio level within a frame is lower than the `threshold level`, or (2) total number of stored audio frames would exceed `max duration`.
- The audio output of sam_ListenForAudio.vi (Section 5.6.3) serves as the beginning of the audio signal `signal out`. The last audio frame containing silence must be excluded from `signal out`.
- The sound-card must be cleaned up using "Graphics and Sound | Sound | Input | Sound Input Clear" once the audio has been recorded.

5.6.2.5 Free LabVIEW Code

This subVI is rather complex to build and debug, so feel free to download the finished subVI sam_GrabAudioDynamic.vi[147] .

You may find it helpful to test the subVI with the demo sam_GrabAudioDynamic_demo.vi[148] .

5.6.3 sam_ListenForAudio.vi[149]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[150] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.38

NOTE: Visit LabVIEW Setup[151] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

[147] http://cnx.org/content/m18641/latest/sam_GrabAudioDynamic.vi
[148] http://cnx.org/content/m18641/latest/sam_GrabAudioDynamic_demo.vi
[149] This content is available online at <http://cnx.org/content/m18598/1.1/>.
[150] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[151] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.6.3.1 LabVIEW SubVI: sam_ListenForAudio.vi

- **Description:** Monitor an audio input and detect when the audio level exceeds a user-defined threshold. Execution flow remains within the VI until the threshold is exceeded, at which time the subVI exits and returns the most recent block of audio.
- **Category:** Speaker-air-microphone (SAM) channel ("sam" prefix)

5.6.3.2 Inputs (Controls)

1. `task ID` – U32
2. `threshold (0.1)` – DBL
3. `number of samples/ch (1024)` – I32
4. `error in (no error)` – error cluster

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.6.3.3 Outputs (Indicators)

1. `task ID out` – U32
2. `first block` – waveform
3. `error out` – error cluster

5.6.3.4 Required Behavior

- Continually acquire audio samples (as in a while-loop structure) in blocks of size `number of samples/ch` for each of the two stereo channels. The subVI exits when the maximum value of an audio block exceeds the value of `threshold`.
- The output `first block` contains the most recent block of audio, i.e., the block containing the audio sample that exceeds the threshold. The output is provided to subsequent subVIs that would consider this waveform to be the first block of useful (non-silent) audio.
- The value of `task ID out` is identical to `task ID in` and facilitates clean block diagram layout for sound-related subVIs.

5.6.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[152] to learn the mechanics of subVIs.

Refer to the Figure 5.39 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 5.39: [video] LabVIEW coding tips and techniques for sam_ListenForAudio.vi

---

[152] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

5.7 Caller ID Decoder

5.7.1 cid_Demodulator.vi[153]

[153] This content is available online at <http://cnx.org/content/m18638/1.1/>.

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[154] module for tutorials and documentation that will help you:
• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.39

NOTE: Visit LabVIEW Setup[155] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.7.1.1 LabVIEW SubVI: cid_Demodulator.vi

- **Description:** Demodulate a Caller ID FSK (frequency shift keying) signal using a PLL (phase-locked loop) from the LabVIEW Modulation Toolkit. The subVI accepts an signal that can include ringer pulses (the FSK signal itself occurs between the first and second ringer pulses), and demodulates the signal to baseband. A "PLL locked" output signal indicates the portion of the baseband signal that should be considered useable for further decoding.
- **Category:** Caller ID decoding ("cid" prefix)

5.7.1.2 Inputs (Controls)

1. `FSK signal` – waveform
2. `VCO carrier frequency [Hz]` – DBL
3. `VCO gain [degrees/V]` – DBL
4. `phase error LPF cutoff frequency [Hz]` – DBL
5. `comparator threshold for PLL lock` – DBL

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.7.1.3 Outputs (Indicators)

1. `baseband signal` – waveform
2. `phase error magnitude` – waveform
3. `PLL locked` – 1-D Boolean array

5.7.1.4 Required Behavior

- `FSK signal` should contain an audio recording of the Caller ID FSK message sent by the telephone company central office (CO). The signal should lie in the range ±1; ringer pulses will be clipped, and the FSK signal amplitude should occupy as much of the ±1 range as possible without clipping.
- The `baseband signal` output contains the demodulated baseband signal produced by the LabVIEW Modulation Toolkit "MT Phase Locked Loop.vi" phase error output.

[154] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[155] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

- The Boolean array `PLL locked` indicates the region in which the PLL is locked onto the FSK signal; this indicator serves to distinguish between the valid FSK signal and any other portion of the original recorded signal.
- The VCO carrier frequency and gain are two controls for the PLL.
- `phase error LPF cutoff frequency` sets the cutoff frequency of the lowpass filter applied to the magnitude of the PLL phase error. The phase error magnitude is a rapidly changing and relatively large amplitude signal value when the PLL is out of lock, and a relatively low amplitude signal in lock. The lowpass filter removes the rapid variation. The `phase error magnitude` output is the lowpass-filtered absolute value of the PLL phase error.
- `comparator threshold` sets the threshold level for the comparator that generates the `PLL locked` Boolean output.

The LabVIEW Modulation Toolkit PLL is introduced and demonstrated in the screencast video of Figure 5.40.

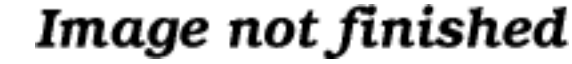


Figure 5.40: [video] Demonstration of the LabVIEW Modulation Toolkit PLL

5.7.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[156] to learn the mechanics of subVIs.

Refer to the Figure 5.41 screencast video for LabVIEW coding tips and techniques specific to this subVI.

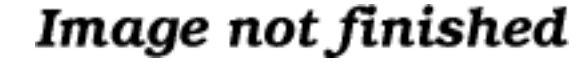
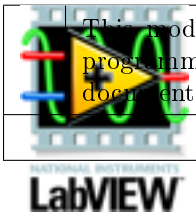


Figure 5.41: [video] LabVIEW coding tips and techniques for cid_Demodulator.vi

5.7.2 cid_DetectStartBit.vi[157]



This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[158] module for tutorials and documentation that will help you:

continued on next page

[156] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[157] This content is available online at <http://cnx.org/content/m18432/1.1/>.

• Apply LabVIEW to Audio Signal Processing
• Get started with LabVIEW
• Obtain a fully-functional evaluation edition of LabVIEW

Table 5.40

NOTE: Visit LabVIEW Setup[159] to learn how to adjust your own LabVIEW environment to match the settings used by the LabVIEW screencast video(s) in this module. Click the "Fullscreen" button at the lower right corner of the video player if the video does not fit properly within your browser window.

5.7.2.1 LabVIEW SubVI: cid_DetectStartBit.vi

- **Description:** Detect the first start bit in the Caller ID message bitstream, and return only the remaining bits in the bitstream. The Caller ID message consists of three distinct regions: (1) channel seizure (alternating pattern of `T` and `F` values), (2) steady mark (constant `T` values), and (3) data block containing the message payload. This subVI detects the steady mark region and then identifies the array index (time point) at which the input bitstream first changes to `F`.
- **Category:** Caller ID decoding ("cid" prefix)

5.7.2.2 Inputs (Controls)

1. `bitstream in` – 1-D Boolean array

Parentheses ( ) indicate default value; square brackets [ ] designate units.

5.7.2.3 Outputs (Indicators)

1. `datablock bitstream` – 1-D Boolean array
2. `start bit index` – I32

5.7.2.4 Required Behavior

- The `bitstream in` input should contain a complete Caller ID message bitstream as generated by other demodulating and bit synchronization and sampling subsystems.
- The `datablock bitstream` output contains only the data block portion of the input bitstream, beginning with the first start bit of the first character, i.e., the first frame. If no data block is detected then `datablock bitstream` will return empty.
- `start bit index` is the index (location) of the data block detected in the input message. If no data block is detected then `start bit index` will return -1.

[158] "NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[159] "LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

5.7.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[160] to learn the mechanics of subVIs.

Refer to the Figure 5.42 screencast video for LabVIEW coding tips and techniques specific to this subVI.

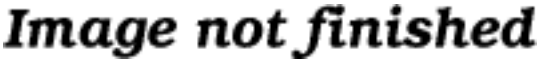


Figure 5.42: [video] LabVIEW coding tips and techniques for cid_DetectStartBit.vi

[160] "Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

((n,k) block code, § 2.2(30)
(n,k) block codes, 29
(n,k) Hamming block codes, 29
(n,k) Hamming code, 32

A Additive white Gaussian noise, 20
AGC, 58
American Standard Code for Information Interchange, 57
amplitude, 50
ASCII, 57
automatic gain control, 58
AWGN, 20

B Bandpass channels, 50
baseband, 50
BER, 78
binary amplitude shift keying, 49
binary antipodal signaling, 19
binary ASK, 49
binary FSK, 36
binary symmetric channel, 4, § 2.2(30), 76
bit error, 79
bit error rate, 78
bit slots, 10
bit sync generator, 19
bit synchronizer, 58
Block coding, 27, § 2.2(30), 31
block diagram, 1

C Caller ID, 36
carrier wave, 50
central office, 37
channel decoder, 27, § 2.2(30), 31, 32
channel encoder, § 2.2(30)
Channel encoding, 27, 31
channel impairment, 17
channel seizure, 38, 59
checkbits, 29, § 2.2(30)
CO, 37
code rate, 29
codeword, 27, 31
coherent detection, 17
Communication Systems Projects with LabVIEW, 1, 1, 2, 2
correction power, § 2.2(30)
correlation detector, 17
correlator, 20
CPE, 37
customer premises equipment, 37

D data block, 38
decision device, 59
delay distortion, 15
digital continuous wave modulation, 50
directory information, 36
discrete memoryless channel, 4

E envelope detector, 58
Error control coding, 28, § 2.2(30), 32
error correction, 27, § 2.2(30), 31
error detection, 27, § 2.2(30), 31
event structure, 15
excess bandwidth, 51
eye diagram, 10, 12

F fault tolerant, 59
frame synchronization, 59
frames, 8, 38
framing bits, 60
frequency, 50
frequency division multiplexing, 63
frequency shift keying, 36
front panel, 1
full duplex, 45

G generator matrix, 29, § 2.2(30)

H half duplex, 45
Half-power frequency, 47
Hamming code, § 2.2(30)
Hamming distance, 28, § 2.2(30)

I inertia, 59
integrate-and-dump, 20

interference, 10
intersymbol interference, 10
ISI, 10, 12

K keying, 49

L LabVIEW, § 2.2(30)
linear block codes, 29
lookup table, 32
LSB, 38

M Manchester, 19
marks, 103
minimum distance, 28
modulated, 49
modulo-2 matrix multiplication, § 2.2(30)
MSB, 38

N Noise, 17
noise floor, 47
noise margin, 10, 10, 12
non-coherent detection, 17
normalized frequency, 11
normalizer, 58

O off hook, 37
on hook, 37

P phase, 50
polar NRZ, 19
preamble, 59
property nodes, 15
pulse generator, 19
pulse shaping filter, 50

Q Q, 59
quality factor, 59

R raised cosine pulse, 51
receiver filter, 58
rectangular, 19
regeneration, 59
ringing pulse, 37

S SAM channel, 43
sample-and-hold, 20
sampler, 58
screencasts, 2
signal point mapper, 19, 50
SLIC, 37
spaces, 103
stacked chart, 22
start bit, 38
steady mark, 38, 60
stop bit, 38
subscriber line interface card, 37
subVIs, 2
switching, 49
syndrome, § 2.2(30), 32
syndrome table, § 2.2(30), 119

T table lookup decoder, § 2.2(30)
table lookup syndrome decoder, 32
threshold comparator, 59
timing jitter, 10
timing sensitivity, 10, 12
transmit filter, 50

V Virtual Instrument, 2

W while-loop structure, 7

Z zero-crossing jitter, 12

Attributions

Collection: *Communication Systems Projects with LabVIEW*
Edited by: Ed Doering
URL: http://cnx.org/content/col10610/1.2/
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introduction"
By: Ed Doering
URL: http://cnx.org/content/m18826/1.2/
Pages: 1-2
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Digital Communication System Simulation and Visualization"
By: Ed Doering
URL: http://cnx.org/content/m18660/1.2/
Pages: 3-8
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Intersymbol Interference (ISI) and the Eye Diagram"
By: Ed Doering
URL: http://cnx.org/content/m18662/1.1/
Pages: 8-15
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "PAM Transmitter and Receiver Implementing Coherent Detection"
By: Ed Doering
URL: http://cnx.org/content/m18652/1.2/
Pages: 15-25
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Hamming Block Code Channel Encoder"
By: Ed Doering
URL: http://cnx.org/content/m18663/1.1/
Pages: 27-30
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Hamming Block Code Channel Decoder"
By: Ed Doering
URL: http://cnx.org/content/m18665/1.2/
Pages: 30-34
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Caller ID Decoder"
By: Ed Doering
URL: http://cnx.org/content/m18708/1.1/
Pages: 35-42
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Speaker-Air-Microphone (SAM) Channel Characterization"
By: Ed Doering
URL: http://cnx.org/content/m18666/1.2/
Pages: 43-48
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Binary ASK Transmitter"
By: Ed Doering
URL: http://cnx.org/content/m18668/1.1/
Pages: 48-55
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Texting Over the Speaker-Air-Microphone (SAM) Channel"
By: Ed Doering
URL: http://cnx.org/content/m18670/1.1/
Pages: 55-63
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introduction to the LabVIEW Modulation Toolkit"
By: Ed Doering
URL: http://cnx.org/content/m18715/1.1/
Pages: 63-67
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_BitstreamFromRandom.vi"
By: Ed Doering
URL: http://cnx.org/content/m18528/1.1/
Pages: 69-70
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_BitstreamFromText.vi"
By: Ed Doering
URL: http://cnx.org/content/m18631/1.1/
Pages: 70-71
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_BitsToWords.vi"
By: Ed Doering
URL: http://cnx.org/content/m18596/1.1/
Pages: 71-73
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_WordsToBits.vi"
By: Ed Doering
URL: http://cnx.org/content/m18551/1.1/
Pages: 73-74
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_BitstreamToText.vi"
By: Ed Doering
URL: http://cnx.org/content/m18629/1.1/
Pages: 74-75
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_BinarySymmetricChannel.vi"
By: Ed Doering
URL: http://cnx.org/content/m18537/1.1/
Pages: 75-76
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_AWGNchannel_PtByPt.vi"
By: Ed Doering
URL: http://cnx.org/content/m18515/1.1/
Pages: 76-78
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_MeasureBER.vi"
By: Ed Doering
URL: http://cnx.org/content/m18547/1.1/
Pages: 78-79
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_EdgeDetector.vi"
By: Ed Doering
URL: http://cnx.org/content/m18606/1.1/
Pages: 79-80
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_GetAudio.vi"
By: Ed Doering
URL: http://cnx.org/content/m18532/1.1/
Pages: 80-82
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_Qfunction.vi"
By: Ed Doering
URL: http://cnx.org/content/m18545/1.1/
Pages: 82-84
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "pam_RaisedCosinePulse.vi"
By: Ed Doering
URL: http://cnx.org/content/m18566/1.1/
Pages: 85-87
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "pam_RectanglePulse.vi"
By: Ed Doering
URL: http://cnx.org/content/m18454/1.1/
Pages: 87-88
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "pam_ManchesterPulse.vi"
By: Ed Doering
URL: http://cnx.org/content/m18466/1.1/
Pages: 88-89
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "pam_SignalPointMapper.vi"
By: Ed Doering
URL: http://cnx.org/content/m18570/1.1/
Pages: 89-90
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "pam_TransmitFilter.vi"
By: Ed Doering
URL: http://cnx.org/content/m18472/1.1/
Pages: 90-93
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "pam_TransmitSync.vi"
By: Ed Doering
URL: http://cnx.org/content/m18478/1.1/
Pages: 93-94
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "bpm_EnvelopeDetector.vi"
By: Ed Doering
URL: http://cnx.org/content/m18420/1.1/
Pages: 94-95
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "bpm_ProductModulator.vi"
By: Ed Doering
URL: http://cnx.org/content/m18556/1.1/
Pages: 95-97
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "bpm_ReceiverFilter.vi"
By: Ed Doering
URL: http://cnx.org/content/m18436/1.1/
Pages: 97-98
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_BitClock.vi"
By: Ed Doering
URL: http://cnx.org/content/m18612/1.1/
Pages: 99-100
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_BitSync.vi"
By: Ed Doering
URL: http://cnx.org/content/m18627/1.1/
Pages: 100-101
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_FrameSync.vi"
By: Ed Doering
URL: http://cnx.org/content/m18576/1.1/
Pages: 101-102
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_ExtractPreamble.vi"
By: Ed Doering
URL: http://cnx.org/content/m18585/1.1/
Pages: 102-104
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_NormalizeToPreamble.vi"
By: Ed Doering
URL: http://cnx.org/content/m18483/1.1/
Pages: 104-105
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_Correlator.vi"
By: Ed Doering
URL: http://cnx.org/content/m18579/1.1/
Pages: 105-106
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_SampleHold.vi"
By: Ed Doering
URL: http://cnx.org/content/m18621/1.1/
Pages: 106-107
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_Sampler.vi"
By: Ed Doering
URL: http://cnx.org/content/m18593/1.1/
Pages: 107-110
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "regen_BitstreamBuffer.vi"
By: Ed Doering
URL: http://cnx.org/content/m18494/1.1/
Pages: 110-111
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "hamming_DetectorCorrector.vi"
By: Ed Doering
URL: http://cnx.org/content/m18427/1.1/
Pages: 111-112
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "hamming_GeneratorMatrix.vi"
By: Ed Doering
URL: http://cnx.org/content/m18563/1.1/
Pages: 112-113
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "hamming_HammingCodeParameters.vi"
By: Ed Doering
URL: http://cnx.org/content/m18441/1.1/
Pages: 113-116
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "hamming_Mod2MatrixMultiply.vi"
By: Ed Doering
URL: http://cnx.org/content/m18562/1.1/
Pages: 116-117
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "hamming_ParityCheckMatrix.vi"
By: Ed Doering
URL: http://cnx.org/content/m18460/1.1/
Pages: 117-118
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "hamming_SyndromeTable.vi"
By: Ed Doering
URL: http://cnx.org/content/m18618/1.1/
Pages: 118-119
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "sam_GrabAudio.vi"
By: Ed Doering
URL: http://cnx.org/content/m18499/1.1/
Pages: 119-121
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "sam_GrabAudioDynamic.vi"
By: Ed Doering
URL: http://cnx.org/content/m18641/1.1/
Pages: 121-122
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "sam_ListenForAudio.vi"
By: Ed Doering
URL: http://cnx.org/content/m18598/1.1/
Pages: 122-123
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "cid_Demodulator.vi"
By: Ed Doering
URL: http://cnx.org/content/m18638/1.1/
Pages: 124-126
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "cid_DetectStartBit.vi"
By: Ed Doering
URL: http://cnx.org/content/m18432/1.1/
Pages: 126-128
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Communication Systems Projects with LabVIEW

"Communication Systems Projects with LabVIEW" features ten project activities in digital communication systems based on the LabVIEW graphical dataflow programming platform. Each project includes introductory material in the form of text and narrated screencast videos, specifications for low-level subVI building blocks, and step-by-step instructions to assemble application VIs to implement a variety of simulations, visualizations, and working transmitters and receivers. This textbook organizes the projects into related topics, including: (1) Simulation and Visualization of Fundamental Concepts, (2) Channel Coding and Error Control, (3) FSK Demodulation (Caller ID), and (4) Bandpass Communications over the Speaker-Air-Microphone Channel.

About Connexions

Since 1999, Connexions has been pioneering a global system where anyone can create course materials and make them fully accessible and easily reusable free of charge. We are a Web-based authoring, teaching and learning environment open to anyone interested in education, including students, teachers, professors and lifelong learners. We connect ideas and facilitate educational communities.

Connexions's modular, interactive courses are in use worldwide by universities, community colleges, K-12 schools, distance learners, and lifelong learners. Connexions materials are in many languages, including English, Spanish, Chinese, Japanese, Italian, Vietnamese, French, Portuguese, and Thai. Connexions is part of an exciting new information distribution system that allows for **Print on Demand Books**. Connexions has partnered with innovative on-demand publisher QOOP to accelerate the delivery of printed course materials and textbooks into classrooms worldwide at lower prices than traditional academic publishers.