SubVI Specifications for "Communication Systems Projects with LabVIEW"

By:

Ed Doering

SubVI Specifications for "Communication Systems Projects with LabVIEW"

By:
Ed Doering

Online:
< http://cnx.org/content/col10608/1.2/ >

C O N N E X I O N S

Rice University, Houston, Texas

Table of Contents

Chapter 1

General-Purpose Utilities

1.1 Bitstream sources

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 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.

1.1.1.1 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)

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/>

1.1.1.3 Outputs (Indicators)

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

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.

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 1.1 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Figure 1.1: [video] LabVIEW coding tips and techniques for util_BitstreamFromRandom.vi

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 1.2

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/>

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)

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.

1.1.2.3 Outputs (Indicators)

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

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.

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 1.2 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 1.2: [video] LabVIEW coding tips and techniques for util_BitstreamFromText.vi

1.2 Bitstream conversion

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 1.3

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.

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)

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.

1.2.1.3 Outputs (Indicators)

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

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.

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 1.3 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 1.3: [video] LabVIEW coding tips and techniques for util_BitsToWords.vi

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 1.4

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.

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)

1.2.2.2 Inputs (Controls)

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

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

1.2.2.3 Outputs (Indicators)

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

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/>

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 1.4 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 1.4: [video] LabVIEW coding tips and techniques for util_WordsToBits.vi

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 1.5

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.

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)

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/>

1.2.3.3 Outputs (Indicators)

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

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.

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 1.5 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Figure 1.5: [video] LabVIEW coding tips and techniques for util_BitstreamToText.vi

1.3 Channel noise

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 1.6

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/>

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)

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.

1.3.1.3 Outputs (Indicators)

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

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."

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 1.6 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 1.6: [video] LabVIEW coding tips and techniques for util_BinarySymmetricChannel.vi

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 1.7

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.

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)

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.

1.3.2.3 Outputs (Indicators)

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

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 1.7.

[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/>

Image not finished

Figure 1.7: [video] Convert Eb/No to standard deviation

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 1.8 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 1.8: [video] LabVIEW coding tips and techniques for util_AWGNchannel_PtByPt.vi

1.4 Performance metrics

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 1.8

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.

[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/>

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)

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.

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

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.

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 1.9 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 1.9: [video] LabVIEW coding tips and techniques for util_MeasureBER.vi

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

1.5 Miscellaneous

1.5.1 util_EdgeDetector.vi[33]

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

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

Table 1.9

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.

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)

1.5.1.2 Inputs (Controls)

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

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

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

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.

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 1.10 screencast video for LabVIEW coding tips and techniques specific to this subVI.

[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/>

Image not finished

Figure 1.10: [video] LabVIEW coding tips and techniques for util_EdgeDetector.vi

1.5.2 util_GetAudio.vi[37]

	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 1.10

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.

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)

1.5.2.2 Inputs (Controls)

1. `path` – file path

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

1.5.2.3 Outputs (Indicators)

1. `audio` – waveform

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.

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

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 1.11 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 1.11: [video] LabVIEW coding tips and techniques for util_GetAudio.vi

1.5.3 util_Qfunction.vi[41]

	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 1.11

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.

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)

1.5.3.2 Inputs (Controls)

1. x – DBL

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

[40]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[41]This content is available online at <http://cnx.org/content/m18545/1.1/>.
[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/>

1.5.3.3 Outputs (Indicators)

1. `Q(x)` – DBL

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.

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 1.12 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Figure 1.12: [video] LabVIEW coding tips and techniques for util_Qfunction.vi

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

Chapter 2

Baseband Modulation and Pulse Amplitude Modulation (PAM)

2.1 Pulse shapes

2.1.1 pam_RaisedCosinePulse.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 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.

2.1.1.1 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)

2.1.1.2 Inputs (Controls)

1. `Tb, bit interval (0.01) [s]` – DBL

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

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.

2.1.1.3 Outputs (Indicators)

1. `pulse shape` – 1-D DBL array

2.1.1.4 Required Behavior

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

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

- 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), $0 \le \alpha \le 1$ (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.

The raised cosine pulse shape is fundamental to digital communication systems. Its name derives from its frequency-domain shape. Refer to the Figure 2.1 screencast video to learn more about the raised cosine pulse.

Image not finished

Figure 2.1: [video] Explanation of raised cosine pulse

2.1.1.5 LabVIEW Coding Tips

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

Refer to the Figure 2.2 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 2.2: [video] LabVIEW coding tips and techniques for pam_RaisedCosinePulse.vi

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

2.1.2 pam_RectanglePulse.vi[5]

[5]This content is available online at <http://cnx.org/content/m18454/1.1/>.

	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 2.2

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.

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)

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.

2.1.2.3 Outputs (Indicators)

1. `pulse shape` – 1-D DBL array

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`.

2.1.2.5 LabVIEW Coding Tips

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

2.1.3 pam_ManchesterPulse.vi[8]

[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/>
[8]This content is available online at <http://cnx.org/content/m18466/1.1/>.

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[9] 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 2.3

> NOTE: Visit LabVIEW Setup[10] 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.

2.1.3.1 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)

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.

2.1.3.3 Outputs (Indicators)

1. `pulse shape` – 1-D array of DBL

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.

2.1.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[11] to learn the mechanics of subVIs.

Refer to the Figure 2.3 screencast video for LabVIEW coding tips and techniques specific to this subVI.

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

Image not finished

Figure 2.3: [video] LabVIEW coding tips and techniques for pam_ManchesterPulse.vi

2.2 Transmitter components

2.2.1 pam_SignalPointMapper.vi[12]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[13] 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 2.4

NOTE: Visit LabVIEW Setup[14] 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.

2.2.1.1 LabVIEW SubVI: pam_SignalPointMapper.vi

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

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.

2.2.1.3 Outputs (Indicators)

1. `signal level` – 1-D DBL array

[12]This content is available online at <http://cnx.org/content/m18570/1.1/>.
[13]"NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[14]"LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

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`.

2.2.1.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[15] to learn the mechanics of subVIs.

Refer to the Figure 2.4 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 2.4: [video] LabVIEW coding tips and techniques for pam_SignalPointMapper.vi

2.2.2 pam_TransmitFilter.vi[16]

	This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[17] 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 2.5

NOTE: Visit LabVIEW Setup[18] 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.

2.2.2.1 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)

[15]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[16]This content is available online at <http://cnx.org/content/m18472/1.1/>.
[17]"NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>
[18]"LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>

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.

2.2.2.3 Outputs (Indicators)

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

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.

2.2.2.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[19] to learn the mechanics of subVIs.

Refer to the Figure 2.5 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 2.5: [video] LabVIEW coding tips and techniques for pam_TransmitFilter.vi

2.2.3 pam_TransmitSync.vi[20]

This module refers to LabVIEW, a software development environment that features a graphical programming language. Please see the LabVIEW QuickStart Guide[21] module for tutorials and documentation that will help you:
continued on next page

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

[20]This content is available online at <http://cnx.org/content/m18478/1.1/>.

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

Table 2.6

NOTE: Visit LabVIEW Setup[22] 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.

2.2.3.1 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)

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.

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

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.

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

2.2.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[23] to learn the mechanics of subVIs.

Refer to the Figure 2.6 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 2.6: [video] LabVIEW coding tips and techniques for pam_TransmitSync.vi

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

Chapter 3

Bandpass Modulation

3.1 bpm_EnvelopeDetector.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 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 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)

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.

[1]This content is available online at <http://cnx.org/content/m18420/1.1/>.
[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 Outputs (Indicators)

1. `baseband signal out` – waveform

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.

3.1.5 LabVIEW Coding Tips

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

Refer to the Figure 3.1 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 3.1: [video] LabVIEW coding tips and techniques for bpm_EnvelopeDetector.vi

3.2 bpm_ProductModulator.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 3.2

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/m18556/1.1/>.
[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/>

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)

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.

3.2.3 Outputs (Indicators)

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

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 3.2 screencast video explains the origin of this equation.
- `carrier` is the output of the sinusoidal oscillator used to modulate the inbound signal.

Figure 3.2: [video] Explanation of the "unit energy per bit" amplitude equation

3.2.5 LabVIEW Coding Tips

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

Refer to the Figure 3.3 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 3.3: [video] LabVIEW coding tips and techniques for bpm_ProductModulator.vi

3.3 bpm_ReceiverFilter.vi[9]

	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 3.3

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.

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)

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.

[8]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[9]This content is available online at <http://cnx.org/content/m18436/1.1/>.
[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/>

3.3.3 Outputs (Indicators)

1. `signal out` – waveform

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.

3.3.5 LabVIEW Coding Tips

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

Refer to the Figure 3.4 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 3.4: [video] LabVIEW coding tips and techniques for bpm_ReceiverFilter.vi

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

Chapter 4

Demodulation and Bitstream Regeneration

4.1 Synchronization

4.1.1 regen_BitClock.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 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.

4.1.1.1 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)

[1]This content is available online at <http://cnx.org/content/m18612/1.1/>.
[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/>

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.

4.1.1.3 Outputs (Indicators)

1. `bit clock` – 1-D Boolean array

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.

4.1.1.5 LabVIEW Coding Tips

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

Refer to the Figure 4.1 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.1: [video] LabVIEW coding tips and techniques for regen_BitClock.vi

4.1.2 regen_BitSync.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
continued on next page

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

[5]This content is available online at <http://cnx.org/content/m18627/1.1/>.

	• 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.

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)

4.1.2.2 Inputs (Controls)

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

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

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

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`.

4.1.2.5 LabVIEW Coding Tips

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

Refer to the Figure 4.2 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.2: [video] LabVIEW coding tips and techniques for regen_BitSync.vi

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

4.1.3 regen_FrameSync.vi[9]

[9]This content is available online at <http://cnx.org/content/m18576/1.1/>.

	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 4.3

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.

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)

4.1.3.2 Inputs (Controls)

1. `bitstream in` – 1-D Boolean array

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

4.1.3.3 Outputs (Indicators)

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

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.

[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/>

4.1.3.5 LabVIEW Coding Tips

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

Refer to the Figure 4.3 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.3: [video] LabVIEW coding tips and techniques for regen_FrameSync.vi

4.2 Preamble processing

4.2.1 regen_ExtractPreamble.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 4.4

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.

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)

[12]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[13]This content is available online at <http://cnx.org/content/m18585/1.1/>.
[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/>

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.

4.2.1.3 Outputs (Indicators)

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

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`.

4.2.1.5 LabVIEW Coding Tips

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

Refer to the Figure 4.4 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.4: [video] LabVIEW coding tips and techniques for regen_ExtractPreamble.vi

4.2.2 regen_NormalizeToPreamble.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:

continued on next page

[16]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[17]This content is available online at <http://cnx.org/content/m18483/1.1/>.

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

Table 4.5

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.

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)

4.2.2.2 Inputs (Controls)

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

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

4.2.2.3 Outputs (Indicators)

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

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 $\pm$.
- Return the measured DC and RMS values as indicators.

[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/>

4.2.2.5 LabVIEW Coding Tips

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

Refer to the Figure 4.5 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.5: [video] LabVIEW coding tips and techniques for regen_NormalizeToPreamble.vi

4.3 Coherent detection

4.3.1 regen_Correlator.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 4.6

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.

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)

[20]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[21]This content is available online at <http://cnx.org/content/m18579/1.1/>.
[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/>

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.

4.3.1.3 Outputs (Indicators)

1. `signal out` – DBL

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.

4.3.1.5 LabVIEW Coding Tips

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

Refer to the Figure 4.6 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.6: [video] LabVIEW coding tips and techniques for regen_Correlator.vi

4.4 Sampling

4.4.1 regen_SampleHold.vi[25]

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:

continued on next page

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

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

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

Table 4.7

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.

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)

4.4.1.2 Inputs (Controls)

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

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

4.4.1.3 Outputs (Indicators)

1. `signal out` – DBL

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.

4.4.1.5 LabVIEW Coding Tips

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

Refer to the Figure 4.7 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.7: [video] LabVIEW coding tips and techniques for regen_SampleHold.vi

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

4.4.2 regen_Sampler.vi[29]

[29]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[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 4.8

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.

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)

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.

4.4.2.3 Outputs (Indicators)

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

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.

[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/>

4.4.2.5 LabVIEW Coding Tips

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

Refer to the Figure 4.8 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.8: [video] LabVIEW coding tips and techniques for regen_Sampler.vi

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

Table 4.9

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.

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)

4.4.3.2 Inputs (Controls)

1. `bit in` – Boolean
2. `save bit` – Boolean

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

[32]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[33]This content is available online at <http://cnx.org/content/m18494/1.1/>.
[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/>

4.4.3.3 Outputs (Indicators)

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

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.

4.4.3.5 LabVIEW Coding Tips

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

Refer to the Figure 4.9 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 4.9: [video] LabVIEW coding tips and techniques for regen_BitstreamBuffer.vi

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

Chapter 5

Hamming Block Coding

5.1 hamming_DetectorCorrector.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 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 codewords is used as an index into the syndrome table to retrieve the most-likely error pattern, which 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)

5.1.2 Inputs (Controls)

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

[1]This content is available online at <http://cnx.org/content/m18427/1.1/>.
[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/>

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.

5.1.3 Outputs (Indicators)

1. `corrected wordstream` – Boolean 2-D array
2. `error detected` – Boolean 1-D array

5.1.4 Required Behavior

Refer to the description above.

5.1.5 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 hamming_DetectorCorrector.vi

5.2 hamming_GeneratorMatrix.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/m18563/1.1/>.
[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/>

5.2.1 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)

5.2.2 Inputs (Controls)

1. `q, checkbits (3)` – I32
2. `k, message length (4)` – I32

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

5.2.3 Outputs (Indicators)

1. `G matrix, k by n` – Real Matrix
2. `P matrix, k by q` – Real Matrix

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.

5.2.5 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.

Image not finished

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

5.3 hamming_HammingCodeParameters.vi[9]

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:

continued on next page

[8]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[9]This content is available online at <http://cnx.org/content/m18441/1.1/>.

	• 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.

5.3.1 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)

5.3.2 Inputs (Controls)

1. `q, checkbits (3)` – I32

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

5.3.3 Outputs (Indicators)

1. `n, codeword length` – I32
2. `k, message length` – I32
3. `Rc, code rate` – DBL

5.3.4 Required Behavior

1. $n = 2^q - 1$
2. $k = n - q$
3. $R_c = \frac{k}{n}$

5.3.5 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.

Image not finished

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

[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/>

5.4 hamming_Mod2MatrixMultiply.vi[13]

[13]This content is available online at <http://cnx.org/content/m18562/1.1/>.

	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.

5.4.1 LabVIEW SubVI: hamming_Mod2MatrixMultiply.vi

- **Description:** Multiply two matrices A and B under modulo-2 arithmetic.
- **Category:** Hamming (n,k) block code ("hamming" prefix)

5.4.2 Inputs (Controls)

1. `A` – Real Matrix
2. `B` – Real Matrix

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

5.4.3 Outputs (Indicators)

1. `A*B` – Real Matrix

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.

5.4.5 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 hamming_Mod2MatrixMultiply.vi

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

5.5 hamming_ParityCheckMatrix.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.

5.5.1 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)

5.5.2 Inputs (Controls)

1. `P matrix, k by q` – Real Matrix

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

5.5.3 Outputs (Indicators)

1. `HT matrix, q by n` – Real Matrix

5.5.4 Required Behavior

1. The transpose of the parity check matrix is

$$H^T \triangleq \begin{bmatrix} P \\ \hline I_q \end{bmatrix}$$

, where

$$I_q$$

is the q by q identity matrix, and P is the parity matrix associated with Hamming code generator (G) matrix.

[17]This content is available online at <http://cnx.org/content/m18460/1.1/>.
[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/>

5.5.5 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 hamming_ParityCheckMatrix.vi

5.6 hamming_SyndromeTable.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.

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)

5.6.2 Inputs (Controls)

1. `HT matrix, n by q` – Real Matrix

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

[20]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[21]This content is available online at <http://cnx.org/content/m18618/1.1/>.
[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/>

5.6.3 Outputs (Indicators)

1. `syndrome table` – Boolean 2-D array

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.

5.6.5 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.

Image not finished

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

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

Chapter 6

Speaker - Air - Microphone (SAM) Channel

6.1 sam_GrabAudio.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 6.1

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.

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)

6.1.2 Inputs (Controls)

1. `duration [s] (1)` – DBL
2. `threshold level (0.1)` – DBL
3. `fs [Hz] (22050)` – DBL

[1]This content is available online at <http://cnx.org/content/m18499/1.1/>.
[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/>

4. `error in (no error)` – error cluster

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

6.1.3 Outputs (Indicators)

1. `waveform out` – waveform
2. `error out` – error cluster

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.

6.1.5 LabVIEW Coding Tips

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

Refer to the Figure 6.1 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 6.1: [video] LabVIEW coding tips and techniques for sam_GrabAudio.vi

6.2 sam_GrabAudioDynamic.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 6.2

[4]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>
[5]This content is available online at <http://cnx.org/content/m18641/1.1/>.
[6]"NI LabVIEW Getting Started FAQ" <http://cnx.org/content/m15428/latest/>

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.

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 6.3). In addition, sam_GrabAudio.vi (Section 6.1) should be constructed before attempting the dynamic-stop version of this module.
- **Category:** Speaker-air-microphone (SAM) channel ("sam" prefix)

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.

6.2.3 Outputs (Indicators)

1. `signal out` – waveform
2. `error out` – error cluster

6.2.4 Required Behavior

- Use sam_ListenForAudio.vi (Section 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 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.

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[8] .

You may find it helpful to test the subVI with the demo sam_GrabAudioDynamic_demo.vi[9] .

[7]"LabVIEW Setup for "Communication Systems Projects with LabVIEW"" <http://cnx.org/content/m17319/latest/>
[8]http://cnx.org/content/m18641/latest/sam_GrabAudioDynamic.vi
[9]http://cnx.org/content/m18641/latest/sam_GrabAudioDynamic_demo.vi

6.3 sam_ListenForAudio.vi[10]

[10]This content is available online at <http://cnx.org/content/m18598/1.1/>.

	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 6.3

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.

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)

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.

6.3.3 Outputs (Indicators)

1. `task ID out` – U32
2. `first block` – waveform
3. `error out` – error cluster

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.

[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/>

6.3.5 LabVIEW Coding Tips

View the screencast video in Create a SubVI in LabVIEW[13] to learn the mechanics of subVIs.

Refer to the Figure 6.2 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Image not finished

Figure 6.2: [video] LabVIEW coding tips and techniques for sam_ListenForAudio.vi

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

Chapter 7

Caller ID Decoder

7.1 cid_Demodulator.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 7.1

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.

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)

7.1.2 Inputs (Controls)

1. `FSK signal` – waveform
2. `VCO carrier frequency [Hz]` – DBL

[1]This content is available online at <http://cnx.org/content/m18638/1.1/>.
[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. `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.

7.1.3 Outputs (Indicators)

1. `baseband signal` – waveform
2. `phase error magnitude` – waveform
3. `PLL locked` – 1-D Boolean array

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.
- 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 7.1.

Figure 7.1: [video] Demonstration of the LabVIEW Modulation Toolkit PLL

7.1.5 LabVIEW Coding Tips

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

Refer to the Figure 7.2 screencast video for LabVIEW coding tips and techniques specific to this subVI.

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

Image not finished

Figure 7.2: [video] LabVIEW coding tips and techniques for cid_Demodulator.vi

7.2 cid_DetectStartBit.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 7.2

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.

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)

7.2.2 Inputs (Controls)

1. `bitstream in` – 1-D Boolean array

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

7.2.3 Outputs (Indicators)

1. `datablock bitstream` – 1-D Boolean array
2. `start bit index` – I32

[5]This content is available online at <http://cnx.org/content/m18432/1.1/>.
[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/>

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.

7.2.5 LabVIEW Coding Tips

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

Refer to the Figure 7.3 screencast video for LabVIEW coding tips and techniques specific to this subVI.

Figure 7.3: [video] LabVIEW coding tips and techniques for cid_DetectStartBit.vi

[8]"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

Attributions

Collection: *SubVI Specifications for "Communication Systems Projects with LabVIEW"*
Edited by: Ed Doering
URL: http://cnx.org/content/col10608/1.2/
License: http://creativecommons.org/licenses/by/2.0/

Module: "util_BitstreamFromRandom.vi"
By: Ed Doering
URL: http://cnx.org/content/m18528/1.1/
Pages: 1-2
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: 2-3
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: 3-5
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: 5-6
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: 6-7
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: 7-8
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: 35-38
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: 38-39
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: 39-41
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: 41-42
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: 42-43
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: 43-46
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: 46-47
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: 49-50
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

SubVI Specifications for "Communication Systems Projects with LabVIEW"
"Communication Systems Projects with LabVIEW" features ten projects that draw upon a library of over 40 subVIs constructed as part of the project activities. Each module in this manual completely describes the input/output requirements as well as the required behavior of a subVI. In addition, each module features a narrated screencast video of the LabVIEW program in action to demonstrate relevant coding techniques to successfully build and test the subVI. Use the Connexions "Table of Contents" viewer to quickly browse for subVIs of interest. Download the manual as a single PDF document (see "Content Actions") as a convenient way to access all modules in a single file; the PDF includes an index.