
Reed-Solomon Lab

Background

Reed-Solomon is an forward error correction (FEC) algorithm that enables the correction of errors injected into a data stream from a noisy channel. A data stream is broken up into blocks (called data packets) of the same number of bytes (each byte is called a data symbol). These data packets are then passed to the Reed-Solomon encoder, which adds several bytes (called check symbols) to the end of each data packet. Together, the data and check symbols make up a codeword.

The Reed-Solomon decoder uses the check symbols at the end of each codeword to correct any data symbols that were corrupted during transmission. The number of errors the decoder can correct is dependent on the number of check symbols that are in the codeword. For every two check symbols in a codeword, the decoder can correct one data symbol. Therefore, if there are 16 check symbols in a codeword, the decoder can correct 8 errors.

The designer can specify the number of data symbols and check symbols in each codeword. Depending on how you configure your system, there are various trade-offs to using different numbers of data and check symbols. For example, if you want the maximum bandwidth possible, you should add the fewest number of check symbols to each data packet. In contrast, if flawless data transmission is more important than overall throughput, you should add as many check symbols to each data packet as possible.

In this lab you will build an FEC system using the Reed-Solomon compiler MegaCore® function and the APEX™ DSP development board.

- *Exercise 1*—This exercise shows you how to build a Reed-Solomon encoder that conforms to the digital video broadcast (DVB) standard.
- *Exercise 2*—This exercise shows you how to build a decoder that receives the output from the encoder you built in exercise 1. You will add errors before simulation to see how the decoder handles them.
- *Exercise 3*—In this exercise, you will use an Altera-provided Windows application and the APEX DSP development board to experiment with sending data through a Reed-Solomon encoder/decoder.

Before You Begin

These instructions assume that you have already installed the software provided with the APEX DSP development kit onto your PC. If you have not done so, refer to the *APEX DSP Development Kit Getting Started User Guide* for installation instructions. You must also have the following software installed on your PC:

- Quartus® II (limited edition or a full version)



This white paper assume that you have installed the Quartus II software into the default location.

Exercise 1: Build a Reed-Solomon Encoder

This exercise shows you how to build a Reed-Solomon encoder that conforms to the DVB standard. You will simulate your encoder and view the results to see that the appropriate number of check symbols were added to the data packets. This exercise includes the following steps:

1. Create a Quartus II project.
2. Generate a Reed-Solomon encoder using the wizard interface.

3. Add input and output pins to your design.
4. Compile the design.
5. Simulate the design.
6. View the simulation results.

1. Create a New Quartus II Project

In the Quartus II software, a “project” consists of the complete set of design files, assignment files, simulation files, system settings (including user libraries), and hierarchy information for a design. To build a design, you must first create a project. Perform the following steps to create a Block Design File (.bdf) and a Quartus II project.


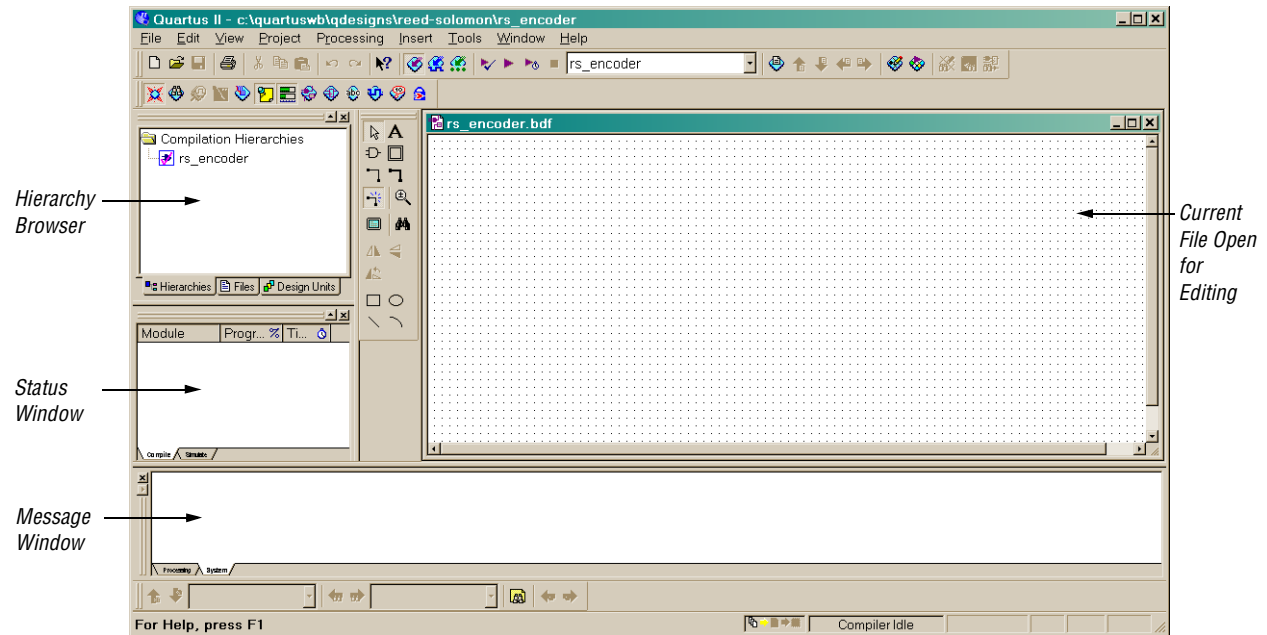
1. Choose **Programs > Altera > Quartus II <version> Limited Edition** (Windows Start menu).
2. Choose **New** (File menu).
3. Select **Block Diagram/Schematic File** in the Device Design Files tab.
4. Click **OK**. A new Block Diagram/Schematic File (.bdf) displays.
5. Choose **Save As** (File menu).
6. Browse to the `c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\lab\Part1_encoding` directory.
7. Enter `rs_encoder.bdf` in the **File name** box.
8. Make sure the **Create a new project based on this file** option is turned on (it is on by default).
9. Click **Save As**.
10. Click **Yes** in the Quartus II message box that asks if you want to create a new project.
11. Click **Next** in the **New Project Wizard: Introduction**.
 The **New Project Wizard: Introduction** page has the **Don't show me this introduction again** option. You will only see this page if you have not (or someone else has not) previously turned this option on.
12. The **New Project Wizard: Directory, Name, and Top-Level Entity** page should be filled in based on the name and location in which you saved your BDF. Click **Next**.
13. In the **New Project Wizard: Add Files** page, click the **User Library Pathnames** button.
14. Click the **Browse (...)** button.
15. Browse to the directory in which the Reed-Solomon compiler was installed. The default is `c:\MegaCore\reed-solomon_compiler-v3.2.0`.
16. Browse to the **lib** directory.
17. Click **Open**.
18. Click **OK**.
19. Click **Next** to view a summary of your project.
20. Click **Finish**.


Figure 1 shows a screen shot of the Quartus II software after you have created the project.

Figure 1. Quartus II Screen Shot




2. Generate a Reed-Solomon Encoder Using the Wizard Interface

You will now build a Reed-Solomon encoder design in the blank BDF you just created. First, you will create a Reed-Solomon encoder function using a wizard interface and then you will add pins to transfer the relevant data off chip. Perform the following steps to build the design.

 If the **rs_encoder.bdf** file that you created earlier is not open, then open it by choosing the **Open** command (File menu).

1. Choose the **Symbol** command (Insert menu).
2. Click the **MegaWizard Plug-In Manager** button.
3. Turn on the **Create a new custom megafunction variation** option and click **Next**.
4. You can select the IP block that you would like to use and the language in which you want it to be instantiated. Expand the **Signal Processing** folder and the **Forward Error Correction** folder in the **Available Megafunctions** box.
5. Select **RS Compiler v3.2.0**.
6. Choose the language of the output files you want to create. The wizard creates a file in the language of your choice that instantiates the Reed-Solomon encoder.

 For the purposes of this lab, it does not matter which language you choose. However, specifying a language is a convenient way to ensure compatibility within a design. For example, if you write your design in Verilog HDL, you can choose Verilog HDL for the instantiation file and, therefore, not have a file in another language instantiated in your top-level design file.

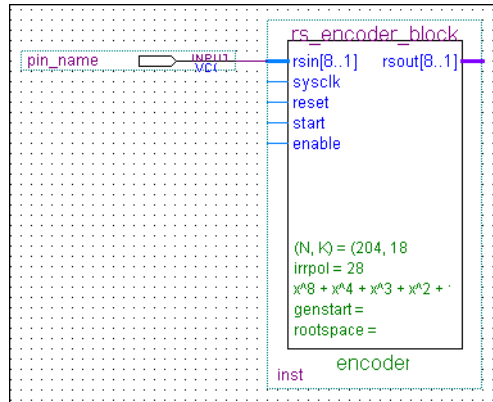
7. The **What name do you want for the output file?** box should have the directory in which you created your project in it. If it does, type `rs_encoder_block` after the directory. If it does not, click **Browse** and browse to the directory in which you created your project. Type `rs_encoder_block` in the **File name** box and click **Save**.
8. Click **Next**.
9. On this page you can choose the architecture of your Reed-Solomon core and whether you want an encoder or decoder. Select the **Encoder** option and click **Next**.
10. On this page, you specify the parameters based on the type of FEC system you want to build. Click the **DVB Standard** button, which automatically sets the number of codewords and check symbols to match the DVB standard.
11. You are done making wizard settings. Click **Finish** to exit the wizard or click **Next** to view the remaining wizard pages. If you click **Finish**, skip to step 15.
12. Page 5 includes an option to create a vector file, and (if a decoder is being built) it shows the name of the memory file and throughput calculator. Click **Next**.
13. Page 6 shows the ordering codes you would use to purchase a license for the core you just configured. Click **Next**.
14. Page 7 shows the wizard output files and where they will be saved. Click **Finish**.
15. Click **OK** and then click the insertion point to insert your symbol into the Block Editor window.

3. Add Input & Output Pins to Your Design

Now that you have built the encoder, you must assign pins so that the signals go off chip. However, in the lab you will only simulate the signals. The Quartus II software compiles a design such that the smallest amount of logic is used to implement the functionality of the output pin signals based on the input pin signals. Therefore, if you do not add pins, your design will have no logic after compilation. Add input and output pins to the design by performing the following steps:

1. Choose **Symbol** (Insert menu).
2. Enter `input` in the **Name** box. The Input symbol appears.
3. Click **OK** and click the insertion point to insert the symbol into the Block Editor window.
4. You can make connections in a BDF by positioning symbols next to each other so that the lines are touching (move one of the blocks away from the other: a connection line appears) or by drawing a line between the two ports you want to connect. When you point to the line on an input/output port it turns into a drawing tool and allows you to draw a line. Connect the input port `rsin[8..1]` to the Input symbol at the top left of the Reed-Solomon block. See [Figure 2](#).

Figure 2. Connect the Symbols



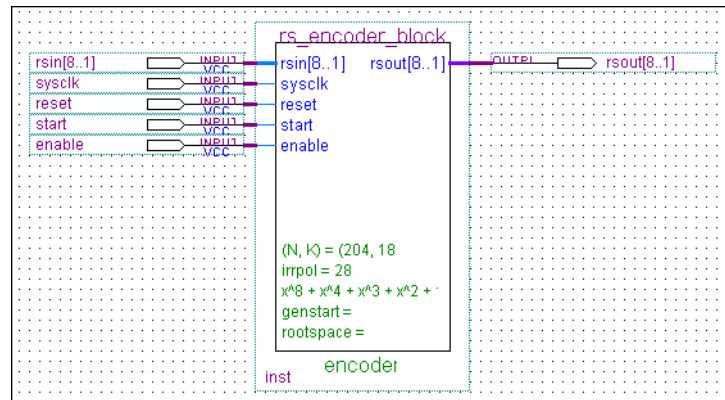
In a BDF, thick lines represent a bus of two or more bits; a thin lines represents a single bit. Search for “Choosing a Line Style” in Quartus II Help for information on how to change the line weight.

- Double-click the Input symbol to open the **Pin Properties** dialog box.
- Enter the input name into the **Pin name(s)** box.

The input pin name **MUST** be the same name as the corresponding port on the Reed-Solomon symbol name to work with the Altera-provided vector files (in a later step). For example, if the Reed-Solomon symbol signal name is `rsin[8..1]`, you must name the input pin `rsin[8..1]`.

- Click **OK**. The default voltage is the pin voltage if no external device drives it to another value. Leave this value at the default, which is **VCC**.
- Add input pins for all of the Reed-Solomon block inputs.
- Repeat steps 1 through 7 for the output pin (except enter `output` in the **Symbol** dialog box **Name** box). **Figure 3** shows the completed design.

Figure 3. Reed-Solomon Encoder BDF



4. Compile the Design

The next step is to analyze and elaborate the design so that you can perform functional simulation. During this step, the Quartus II software synthesizes your design but does not perform place-and-route. Perform the following steps to compile.

1. Choose **Compile Mode** (processing menu).
2. If you receive a message that modified documents will be saved automatically, click **Yes**.
3. Choose the **Start Analysis & Elaboration** command (Processing menu).

Compilation stops if the Compiler issues any errors. The Status window shows the compilation progress and the Message window displays messages and errors. You can find help on any error message by right clicking on the message and choosing **Help**. To locate an error in your design, double-click on the error message.



 For a greater explanation on how to compile a design, refer to “Session 8: Compile the Design” in the Quartus II Tutorial.

Figure 4 shows the messages you receive while compiling.

Figure 4. Reed-Solomon Encoder Compilation Messages

```
Info: Found 1 design units and 1 entities in source file c:\quartuswb\qdesigns\reed-solomon\rs_encoder.bdf
Info: Found 1 design units and 1 entities in source file c:\quartuswb\qdesigns\reed-solomon\rs_encoder_block.tdf
Info: Found 1 design units and 1 entities in source file c:\megacore\reed-solomon_compiler-v3.2.0\lib\RS_enc.tdf
Info: Assertion information: Compiling ALTERA's Reed-Solomon Encoder MegaCore Function
Info: Found 1 design units and 1 entities in source file C:\quartusWB\libraries\megafunctions\lpm_counter.tdf
Info: Found 1 design units and 1 entities in source file C:\quartusWB\libraries\megafunctions\alt_synch_counter.tdf
Info: Found 1 design units and 1 entities in source file c:\megacore\reed-solomon_compiler-v3.2.0\lib\gfmul_cnt.tdf
Info: Implemented 232 device resources
Info: Implemented 8 output pins
Info: Automatically selected device EP20K30ETC144-1 for design rs_encoder
Info: Started 1 fitting attempt on Thu Aug 30 2001 at 14:18:22
Warning: Found pins functioning as undefined clocks and/or memory enables
Info: Clock sysclk has Internal fmax of 108.8 MHz between source register
rs_encoder_block:inst|RS_enc:RS_enc_component|lpm_counter:count|alt_synch_counter:wysi_counter[q[5] and destination register
rs_encoder_block:inst|RS_enc:RS_enc_component|reg[12][1] (period= 9.191 ns)
Info: Design rs_encoder: Full compilation was successful. 0 errors, 3 warnings
```

 If you are evaluating the Reed-Solomon core using the OpenCore feature, you will also receive warning messages that the Compiler cannot generate programming files. These warnings do not affect the lab.

5. Simulate the Design

In this step, you will use an Altera-provided vector file to simulate the design.

1. Choose **Simulate Mode** (Processing menu).
2. Choose **Simulator Settings** (Processor menu).
3. Click the **Mode** tab.
4. Select **Functional** from the **Simulation mode** drop-down list box.
5. Click the **Time/Vectors** tab.
6. Click the **Browse (...)** button in the **Vectors** box.
7. Browse to the **c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\lab\Part1_encoding** directory.

- Select the **RS_Encoder.vwf** file and click **Open**. The signal names in the vector file correspond to the input/output pin names that you specified in “3. Add Input & Output Pins to Your Design” on page 4. The stimuli in this file will exercise the Reed Solomon encoder.

The data bus (`rsin[8..1]`) consists of 188 bytes (1 through 188), which is the data to which the encoder adds check symbols.

The `start` bit is only held high for the first valid byte of each block of data.

The file has clock, reset and enable signals. The core resets once at the beginning of the file and is enabled for the rest of the simulation. The simulation output appears on the `rsout[8..1]` bus.

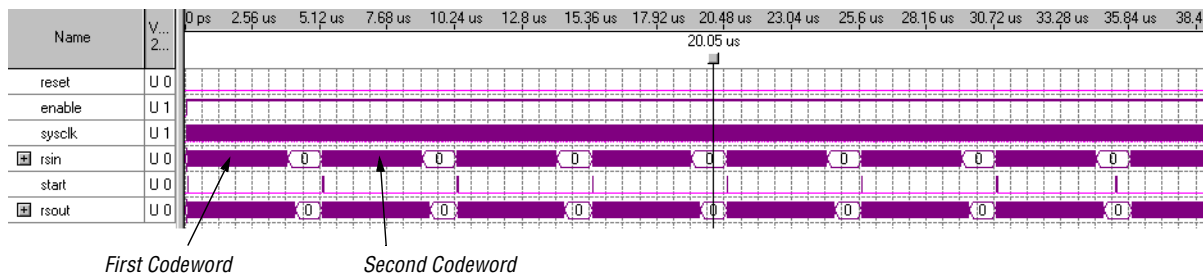
- Click **OK**.
- Choose **Run Simulation** (Processing menu). The Quartus II software analyzes and elaborates the design, and then performs simulation.

6. View the Simulation Results

View the simulation results to see the 16 check symbols that were added to the end of each codeword.

- Choose **Full Screen** (View menu) to expand the simulation view. You can zoom in by pressing the Ctrl + Space Bar keys; zoom out by pressing the Ctrl, Shift + Space Bar keys.
- Zoom all the way out. See Figure 5.

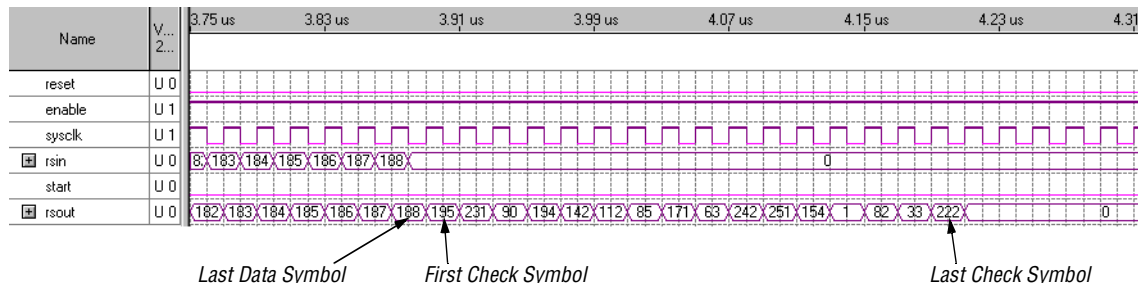
Figure 5. Simulation Zoomed Out



- Zoom in on the first codeword to see the check symbols. The data symbols in this project are the numbers 1 through 188. The first check symbol immediately follows the 188 value. See Figure 6.

To locate a particular codeword, zoom out so that the codewords appear as shown in Figure 5. Highlight the codeword you want to view and zoom in using the Control + Space Bar keys.

Figure 6. Zoom in to See Check Symbols



Exercise 2: Build a Reed-Solomon Decoder

In this exercise you will create a Reed-Solomon decoder and verify the functionality of the encoder and decoder. The decoder corrects up to 8 different errors in the encoded codewords. This exercise involves the following steps:

1. Create a Quartus II project.
2. Generate a Reed-Solomon decoder using the wizard interface.
3. Add input and output pins to your design.
4. Compile the design.
5. Simulate the design.
6. View the simulation results.
7. Analyze the throughput.
8. Analyze the effective silicon cost.

1. Create a Quartus II Project

Follow these steps to create a new project. This process is similar to the one in “[1. Create a New Quartus II Project](#)” on page 2.

1. If you have a project open, choose **Close Project** (File menu).
2. Create a new BDF named **rs_decoder.bdf** and save it into the **c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\lab\Part2_decoding** directory.
3. Create a new project based on this file and add the Reed-Solomon user library.

2. Generate a Reed-Solomon Decoder Using the Wizard Interface

Create a Reed-Solomon decoder. This process is similar to the one in “[2. Generate a Reed-Solomon Encoder Using the Wizard Interface](#)” on page 3.

1. Launch the MegaWizard Plug-In Manager and create a new megafunction variation.
2. Select the Reed-Solomon core, specify a language, and name the output file **rs_decoder_block** (save it into the **c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\lab\Part2_decoding** directory).
3. On page 3, make the following selections:
 - Choose the **Decoder** option.
 - Choose **Streaming** under **Architecture**.
 - Choose **Half** under **Keysize/Throughput Tradeoff**.
4. On page 4, click the **DVB Standard** button.
5. You are done making wizard settings. Click **Finish** to exit the wizard or click **Next** to view the remainder of the wizard.



Wizard page 5 contains a throughput calculator, which you will use in “[7. Analyze Throughput](#)” on page 13 to calculate the decoder’s throughput.

6. Insert the symbol into your BDF.

3. Add Input & Output Pins to Your Design

Add input and output pins to the design by performing the following steps:

1. Insert a VCC symbol into your design using the same method described in “3. Add Input & Output Pins to Your Design” on page 4 for Input symbols except type VCC instead of Input in the **Name** box. Connect the VCC symbol to the Reed-Solomon `dsout` input.
2. Insert a GND symbol (search for GND instead of VCC) and connect it to the Reed-Solomon `bypass` input.
3. Add input and output pins.

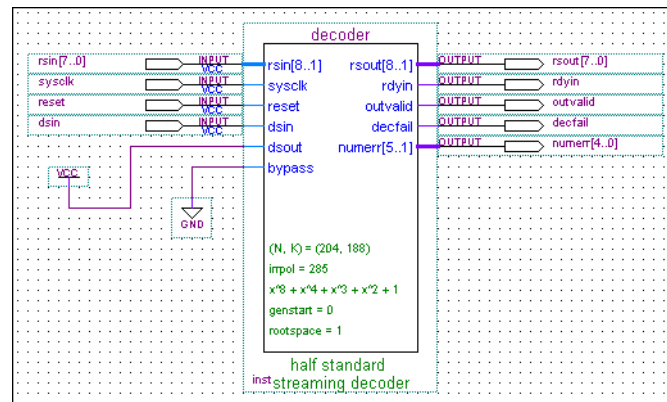


For the encoder all inputs and outputs must have the same name as the ports in the symbol. However, for the decoder, only the single-bit signals must have the same name. The buses must have the same number of bits but the bit numbers can be different.

Name the buses so that the least significant bit is a zero, which makes it simpler to view the simulation results. For example, the port `rsin[8..1]` on the decoder should connect to an input pin named `rsin[7..0]`. In a typical FEC system, the least significant bit is used to determine when valid data is on the data bus. It could be used, for example, to synchronize other blocks such as an interleaver.

Figure 7 shows the completed decoder BDF.

Figure 7. Decoder BDF



4. Compile the Design

The next step is to analyze and elaborate the design, which makes it possible to run a functional simulation on the Reed-Solomon decoder design.

1. Choose **Compile Mode** (Processing menu).
2. Choose the **Start Analysis & Elaboration** command (Processing menu).

5. Simulate the Design

In this section you will add errors into the Altera-provided Vector Waveform File (.vwf) and then simulate the decoder.

1. Choose **Open** (File menu).
2. Select **Waveform/Vector Files (*.vwf, *.vec, *.tbl)** from the **Files of type** drop-down list box.
3. Browse to the `c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\lab\Part2_decoding` directory.
4. Select the **RS_Decoder.vwf** file and click **Open**. The encoder you built in exercise 1 works together with the decoder you just built. The decoder expects to receive the encoder data and check symbols on the input port `rsin[8..1]`. The **RS_Decoder.vwf** `rsin[7..0]` node contains the simulation output from the encoder. The codewords, given in the `rsin` node, all consist of the same data symbols (1 through 188) plus the corresponding check symbols.

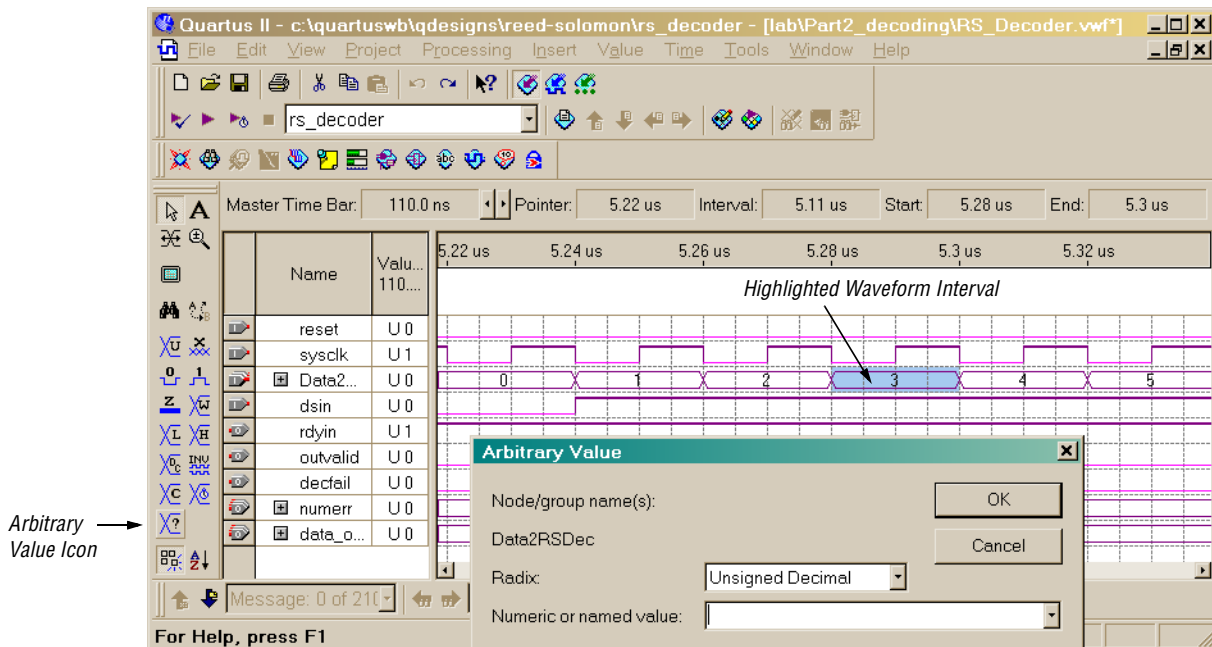
Leave the first codeword as is: it will be used to ensure that all of the signals have the appropriate check symbols appended to them. This decoder input (the `rsin` node) has exactly the same values as the encoder `rsout` node in [Figure 6 on page 7](#).



To locate a particular codeword, zoom out so that the codewords appear as shown in [Figure 5 on page 7](#). Highlight the codeword you want to view and zoom in using the Control + Space Bar keys.

5. In the second codeword, change any 8 of the values. You can make the values all zeros, all ones, or any value except the current one. After simulating, you will check the values to make sure the decoder is correcting up to 8 errors per codeword—because the decoder has 16 check symbols, they should all be corrected. Refer to [Figure 6 on page 7](#) to see where the codewords are located. To change a value, perform the following steps.
 - a. The Waveform Editor toolbar should be visible. If it is not, choose **Toolbars** (Tools) menu and ensure that the **Waveform Editor** option is turned on. Click **OK**. The tool bar buttons are grayed out until you highlight a signal in the waveform.
 - b. Select the portion of the waveform value you want to change by double-clicking on the waveform interval (you can also select it with the mouse). The interval is highlighted. Only select the node that you want to change—the simulation does not work properly if you select portions of waveform that are above or below the `rsin[7..0]` signal. See [Figure 8](#).

Figure 8. Editing the Waveform



- c. Click the **Arbitrary Value** icon in the Waveform Editor toolbar.
 - d. Enter a number between 0 and 255 in the **Numeric or named value** box.
 - e. Click **OK**. The new value displays in the waveform.
6. In the third codeword, change more than 8 values using the same process as described for the second codeword. After simulating you will view this codeword to see how the decoder fails when it receives too many errors per codeword.
 7. Choose **Save As** (File menu).
 8. Browse to the `c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\lab\Part2_decoding` directory.
 9. Type `my_RS_Decoder.vwf` in the **File name** box.
 10. Click **Save**.
 11. Choose **Simulate Mode** (Processing menu).
 12. Choose **Simulator Settings** (Processor menu).
 13. Click the **Mode** tab.
 14. Select **Functional** from the **Simulation mode** drop-down list box.
 15. Click the **Time/Vectors** tab.
 16. Click the **Browse (...)** button in the **Vectors** box.
 17. Browse to the `c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\lab\Part2_decoding` directory.

18. Select the **my_RS_Decoder.vwf** file and click **Open**. The signal names in the vector file correspond to the input/output pin names that you specified in “3. Add Input & Output Pins to Your Design” on page 9.
19. Click **OK**.
20. Choose **Run Simulation** (Processing menu). The simulation runs for 5 to 10 minutes, depending on the speed of your PC. You will receive a message when simulation is complete.

6. View the Simulation Results

The decoder uses the check symbols that the encoder appended to the data symbols to correct any errors that were introduced into the data stream; it can correct 1 error for every 2 check symbols. The decoder asserts a signal (`outvalid`) when valid data is output on the `rsout[8..1]` port. The decoder has a three codeword latency, i.e., it must receive three codewords before the first one is output. Refer to the *Reed-Solomon MegaCore Function User Guide* for complete information on the Reed-Solomon compiler core operation. Table 1 describes the functionality of the decoder signals used in this exercise.

Table 1. Reed-Solomon Decoder Signals

Signal	Direction	Description
<code>rsin[8..1]</code>	Input	Data input bus. Encoded codewords should be fed into this input port.
<code>sysclk</code>	Input	System clock. All data manipulation is done on the rising edge of this signal.
<code>reset</code>	Input	Decoder reset. This signal clears the codewords in the decoder and prepares it for operation.
<code>dsin</code>	Input	This signal is high when valid data is sent to the decoder. It is low when valid data is not present.
<code>dsout</code>	Input	Controls when the decoder will output the codewords. This signal is tied permanently high in this design, so the Decoder will output data as soon as it is done processing it.
<code>bypass</code>	Input	When asserted, the decoder outputs the uncorrected input data instead of the corrected data. All other operations are unaffected and the decoder's latency remains the same. In this exercise, the decoder receives the entire codeword at once; therefore, this signal is tied low.
<code>rsout[8..1]</code>	Output	Output data bus. The decoder outputs the corrected data symbols and the check symbols if it was able to correct all of the errors. If the decoder failed to correct all of the errors, the corrupted data and the check symbols appear on this signal.
<code>rdyin</code>	Output	Indicates the decoder is ready to accept a new codeword. In the provided RS_Decoder.vwf file, the <code>rsin[8..1]</code> signal only has valid data after this signal has gone high.
<code>decfail</code>	Output	Asserted at the beginning of a data output when the decoder has detected errors and cannot correct them.
<code>numerr[5..1]</code>	Output	Number of symbol errors found. Displays up to the maximum number of correctable errors. This signal only display the number of errors found if the <code>decfail</code> signal does not go high. If the <code>decfail</code> signal goes high, <code>numerr</code> signal value is irrelevant.

Perform the following steps to view the simulation results, verifying whether the decoder is working properly.

1. Choose **Open Simulation Report** (Processing menu).
2. Search through the Vector Waveform File (**.vwf**) to find where the `outvalid` signal goes high for the first time. Do the values 1 through 188 plus the check symbols all occur while the `outvalid` signal is high?
3. Check the second codeword. Are the values 1 through 8 that you changed in [step 5 on page 10](#) corrected? Does the `numerr` signal appropriately report the number of errors that were corrected?
4. Check the third codeword. Did the decoder fail? Is the `numerr` signal correct?

7. Analyze Throughput

In this portion of the lab you will determine the throughput of the FEC system using the decoder simulation results. The first step is to specify a specific device in which to implement the design and run a full compile. After compilation, the Quartus II software reports the design's f_{MAX} , which is the maximum clock frequency the decoder can achieve without setting design constraints. To determine the decoder throughput, perform the following steps.

1. Choose the **Compile Mode** command (Processing menu).
2. Choose **Compiler Settings** (Processing menu).
3. Click the **Chips & Devices** tab.
4. Select **APEX 20KE** in the **Family** drop-down list box.
5. Click **No** if you are asked if you want to allow the Compiler to select a device automatically.
6. Select the **EP20K60EBC356-1** device under **Available devices**.
7. Click **OK**.
8. Choose **Start Compilation** (Processing menu) to begin a full compilation. The Quartus II software analyzes and elaborates the design, and fits it into the target device. The Quartus II Timing Analyzer reports the fastest clock speed at which the logic runs. The design's f_{MAX} is displayed in the Message Window. You can also perform the following steps to view the f_{MAX} .
 - a. Choose **Open Compilation Report** (Processing menu).
 - b. Expand **Timing Analyses**.
 - c. Click **fmax (not incl. delays to/from pins)**.
 - d. Scroll all the way to the right to view the f_{MAX} for `sysclk`.

You will use this speed for the throughput analysis.

9. Page 3 of the decoder wizard has a throughput calculator. Using the clock frequency you found in step 6, calculate your system throughput.



The throughput calculator expects the f_{MAX} to be in MHz.

The following steps explain how the calculator calculates the throughput, assuming a 100-MHz f_{MAX} . The design has 204 bytes (188 data symbols and 16 check symbols) that are processed at the rate of one byte per clock cycle. Therefore, the design requires 204 clock cycles to process a full codeword.

- a. A clock frequency of 100 MHz yields a period of 10 ns.
- b. $10 \text{ ns} \times 204 \text{ clock cycles} = 2,040 \text{ ns}$. The decoder decodes one full codeword in 2.040 μs .
- c. Each codeword has 188 data bytes per codeword; compute the throughput of the system by dividing 188 by .00002040.
- d. The calculation yields a throughput of 92,156,863 bytes per second or 737,254,904 bits per second. This result should be similar to the one given by the throughput calculator.

8. Analyze the Effective Silicon Cost

In this portion of the lab you will determine the effective silicon cost of the FEC system. Altera APEX devices are made up of logic elements (LEs) that perform the necessary digital logic manipulation. The more LEs a design uses, the bigger the device needed. Therefore, the amount of logic in a design dictates the effective silicon cost. You can determine the number of LEs your design uses by looking at the Quartus II Report File when compilation completes.

1. Determine the resource usage for the decoder.
 - a. Choose **Open Compilation Report** (Processing menu)
 - b. Expand **Resource Section**.
 - c. Click **Resource Usage Summary** to view the number of LEs used. (The design uses roughly 2,322 LEs.)
 - d. Choose **Close Project** (File menu) when you are done.
2. Determine the resource usage for the encoder.
 - a. Open the encoder project.
 - b. Perform a full compilation (you can specify the same device you used for the encoder or let the Quartus II Compiler choose one automatically).
 - c. Open the compilation report and view the LEs used. (The design uses roughly 206 LEs.)
3. Add the LEs used for the encoder and decoder. This number is the total number of LEs required for the entire FEC system.
4. You can calculate the effective silicon cost by multiplying the percentage of LEs used times the device price. For example, you can implement the 737-MBit/second, 2,600-LE decoder design on an Altera device for less than \$15.

Exercise 3: The Reed-Solomon Demonstration


The Altera Reed-Solomon demonstration shows the benefits of using Altera's forward error correction (FEC) solutions. The demonstration consists of a Windows application in which you specify a picture to transmit over a channel using the Reed-Solomon encoder/decoder. You can use this application to alter the type and intensity of errors inserted into the data as it passes through the channel. The APEX DSP development board implements the hardware portion of the design, which includes the encoder, decoder, and channel.

After you select the picture and error type, the bytes representing the colors of the pixels within the picture are transmitted to the board via an RS-232 port. The data is loaded into a Nios™ embedded processor, which stores it in a FIFO buffer. The data is read out of the FIFO buffer and processed in two ways:

- One data path goes through the selected channel only and the picture is displayed in the GUI with the errors visible.

and

- The other data path goes through the Reed-Solomon encoder, through the channel, through the Reed-Solomon decoder, and then the picture is displayed in the GUI.

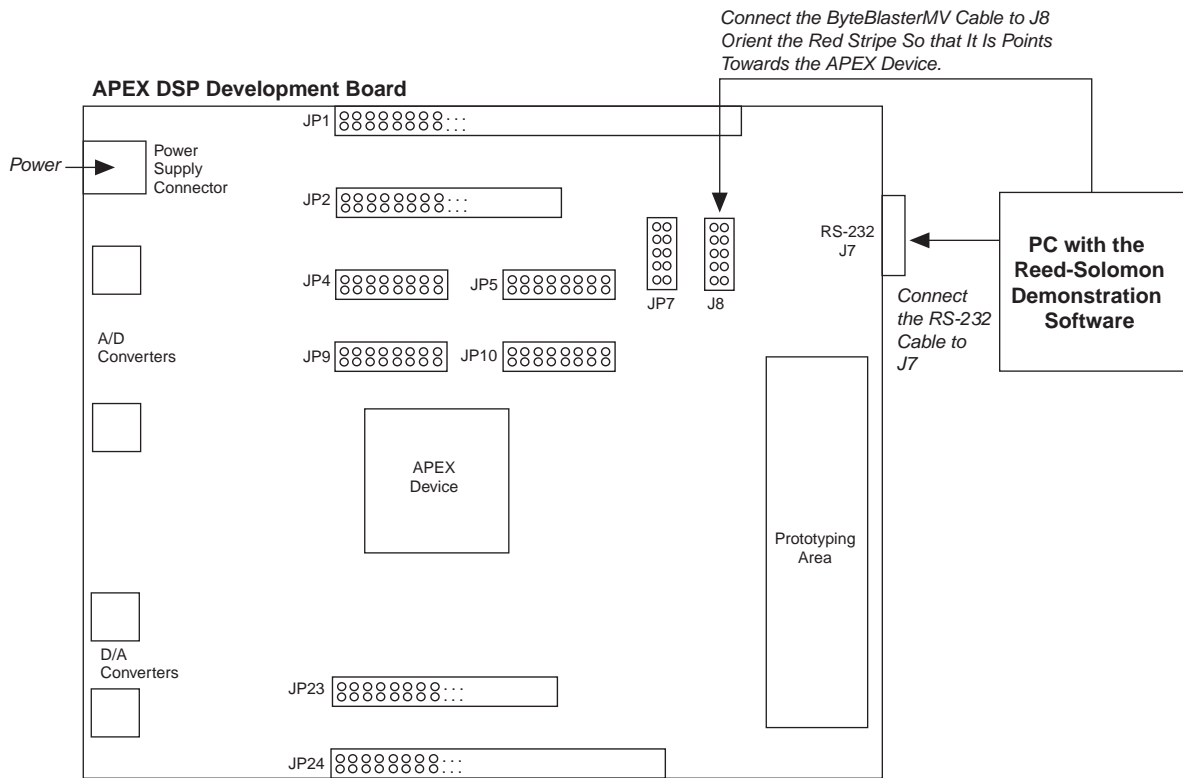
 The demonstration provides an option to include an interleaver/deinterleaver in the data path. If you turn on the **Add Interleaver/Deinterleaver** option (it is only available with the burst channel), the demonstration places an interleaver just after the Reed-Solomon encoder and a deinterleaver just before the decoder.

The MegaCore functions in this demonstration are configured with the following parameters:

- Half, standard, streaming Reed-Solomon decoder (204, 188)
 - 8-bit symbol width
 - 204 symbols per codeword (N = 204)
 - 16 check symbols (R = 16)
- Block, discrete interleaver (232 × 5)

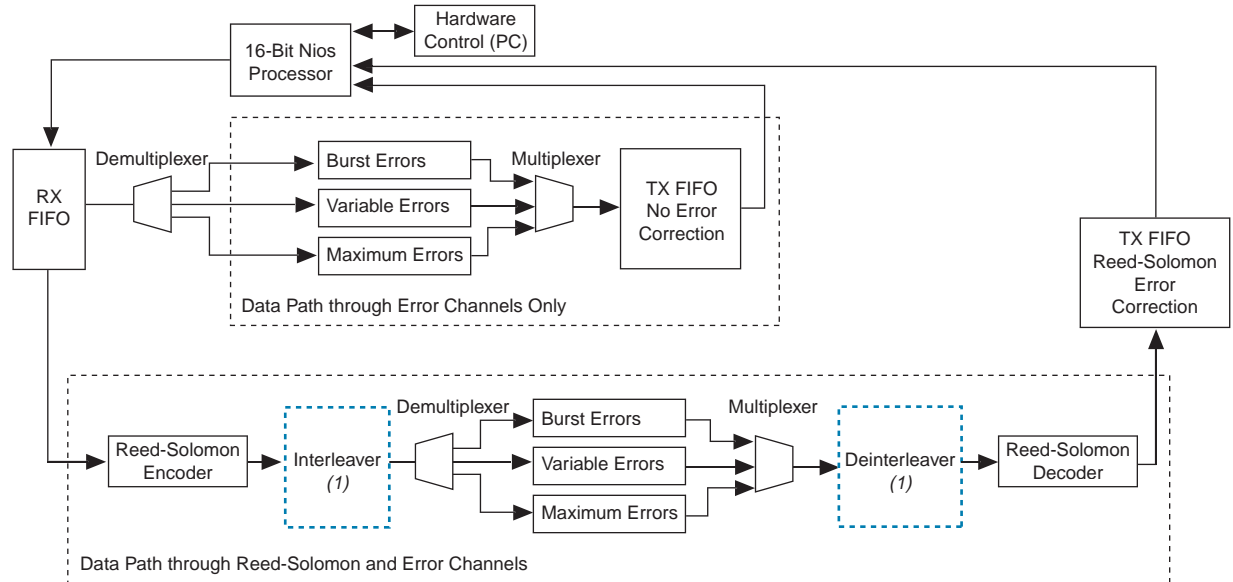
The demonstration runs at 33 MHz (generated by the APEX DSP development board from the 40-MHz clock). [Figure 9](#) shows an overview of the demonstration setup.

Figure 9. Reed-Solomon Demonstration Setup



[Figure 10](#) shows a block diagram of the design implemented in the APEX device on the board.

Figure 10. Block Diagram of Design Implemented in APEX Device on the Board

**Note:**

(1) The interleaver/deinterleaver are added if you select the **Burst Errors** option and turn on **Add Interleaver/Deinterleaver** in the application.

Definitions

The following definitions apply to this demonstration only:

- *Symbol*—An 8-bit piece of data
- *Codeword*—204 bytes consisting of 188 data symbols and 16 Reed-Solomon check symbols
- *Check symbol*—A symbol that the Reed-Solomon encoder adds to the data so the decoder can correct any errors that occur during the data transmission
- *Data packet*—A 940-byte piece of data
- *BER*—Bit error rate

Connect the Cables to the Board

Before running the demonstration, you must connect cables to the AEPX DSP development board. Refer to the *APEX DSP Development Kit Getting Started User Guide* for detailed instructions on how to connect the cables to the board. Perform the following steps to connect the cables:

1. Connect the power adapter cable to the board and plug it into a power outlet.
2. Connect the ByteBlasterMV cable to your PC and to the board's 10-pin JTAG header for APEX configuration.
3. Connect an RS-232 cable to your PC and to the board.

Configure the APEX Device


Perform the following steps to configure the device:

1. Choose **Programs > Altera > Quartus II <version> Limited Edition** (Windows Start Menu) or **Programs > Altera > Quartus II <version>** (Windows Start Menu).

2. Choose **Open Programmer** (Processing menu).
3. Click **Add File**.
4. Browse to the `c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\application\pofs_n_sof` directory.
5. Select the file **dsp_board_starter.sof** if you are using the APEX DSP development board starter version; select the file **dsp_board_professional.sof** if you are using the professional version.
6. Click **Open**.
7. Turn on the **Program/Configure** option.
8. Click **Start** to configure the APEX device.


Run the Demonstration


This section describes how to run the demonstration.

 Before you run the demonstration, ensure that the jumpers at JP25 are set up such that pins 1 and 2 are connected and pins 3 and 4 are connected.

The application does not display properly on systems that use large fonts. To change your font settings, perform the following steps. If you are already using small fonts, skip to step 7.

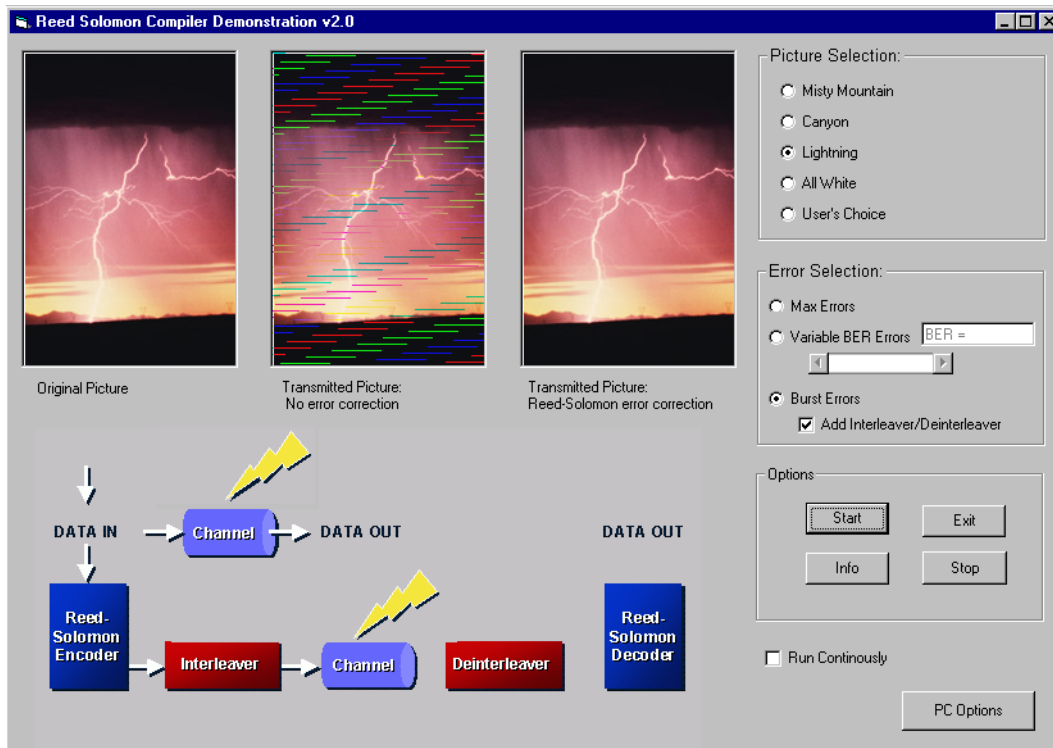
1. Choose **Settings > Control Panel** (Windows Start menu).
2. Double-click the **Display** icon.
3. Click the **Settings** tab.
4. Choose **Small Fonts** under **Font Size**.
5. Click **OK**.
6. Reboot your system for the changes to take effect.
7. Execute the file `c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\applicationrs_demo_v2.exe` to start the demonstration software.
8. Click the **PC Options** button, indicate which COM port you are using, and click **OK**.
9. Select the picture to transmit in the **Picture Selection** box.

 If you prefer, you can use a picture of your choice instead of the ones provided with the demonstration. See [“Adding a Custom Picture”](#) on page 19.
10. Select the error type in the **Error Selection** box (the demonstration provides several types of errors that can be injected into the channel; see [CD Green\xd2 Error Channels](#)” on page Error Channels for more details), and click **Start**.

 The flashing LED flashes twice as fast while the data is being processed.

The data paths are shown in the block diagram in the demonstration software GUI. The transmitted pictures are displayed as the data bits are processed. See [Figure 11](#).

Figure 11. Reed-Solomon Demonstration User Interface



Transmission Details

Each bitmap image transmitted in this demonstration has three data bytes representing each pixel. Data is transmitted in blocks of 5 Reed-Solomon codewords (188 bytes each for a total data packet of 940 bytes). After a block is transmitted to the board and processed, it is sent back to the demonstration software and displayed in the appropriate window before another data packet is sent. A 16-bit Nios embedded processor receives the data using a UART peripheral from the Nios peripheral library. The Nios processor writes the data into a memory-mapped FIFO buffer, where it is read out and passed to the appropriate data path. Then, the data is written to a pair of memory-mapped FIFO buffers from which the Nios processor reads the data and sends it back to the PC.

Error Channels

You can choose from three provided error channels. The **Max Errors** channel deterministically injects 8 errors into each Reed-Solomon codeword. The Reed-Solomon encoder used in this implementation has 16 check symbols, which supports the detection and correction of up to 8 errors per codeword. Therefore, the **Max Errors** channel option injects the maximum number of errors into the data path that the Reed-Solomon decoder can correct.

The **Variable BER** channel lets you select one of several different bit error rates (BERs). This channel injects errors routinely into each codeword, spreading the errors farther apart as lower BERs are selected. The highest BER setting adds 12 symbol errors to each codeword, which is more than this implementation of the Reed-Solomon core can correct. Therefore, this setting causes the Reed-Solomon decoder to fail, and both of the transmitted pictures return with errors.

The **Burst Error** channel injects an 40-symbol error into the first codeword of each data packet. If the **Add Interleaver/Deinterleaver** option is turned off, this channel causes the Reed-Solomon decoder to fail and both pictures return with errors. However, if the interleaver is added to the data path, the 40-symbol error is spread across 5 different codewords and the Reed-Solomon decoder can detect and correct all of the errors.

Adding a Custom Picture

All bitmaps of the same size have the same structure, therefore you can create a custom picture to transmit. Save your image as **my_file.bmp** in the **c:\MegaCore\dsp_development_kit-v1.0.0\labs\reedsolomon\application\bmps** directory. The image must be 183 pixels wide by 260 pixels tall.

For example, you can create an image using the Microsoft Paint software by performing the following steps:

1. View the image on your PC, for example in a web browser or other viewer.
2. Press the **Print Screen** button on your keyboard to copy the image to the clipboard.
3. Open Paint and paste the clipboard contents into a new file.
4. Choose the **Attributes** command (Image menu), set the height to 260 pixels and the width to 183 pixels, and click **OK**. If your current picture is bigger than the new size, it is cut from the right side and bottom to fit within the smaller area. If the current picture is smaller than the new size, the extra area is filled with the selected background color.

Revision History

Table 2 shows the document revision history.

Table 2. Revision History

Date	Description
September 2001	First publication.
February 2002	Updated the LE usage in exercise 2.



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
<http://www.altera.com>

Copyright © 2002 Altera Corporation. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services. All rights reserved.