The Altera® 2.5G Reed-Solomon (RS) II MegaCore® function reference design demonstrates a basic application of the Reed-Solomon algorithm in data transmission between the Altera RS II encoder and decoder.

This reference design targets optical transport network (OTN) 1 applications. This reference design demonstrates channelized data transmission operation of Altera RS II core on Stratix® IV development kit. The reference design provides a platform for you to control, test, and monitor RS coding and decoding operations through simulation and hardware implementation.

The reference design has the following features:

- 8 fix-channel pattern data
- 2.5 Gbits/s data rate. Double data path with 8 bits width at 160 MHz
- 2.5G channelized RS II encoder
- 2.5G channelized RS II decoder
- OTN 1 applications: RS (239, 255)
- Random data generator
- Error injection to data path
- Data checking mechanism
- Statistics and data bits counter

**Functional Description**

The reference design comprises a random data generator system, a 2.5G RS II encoder, an error injection system, a 2.5G RS II decoder, and a checking system. The reference design is built on a single clock domain. The encoder transmits data to the decoder after passing through an error injection module.
Figure 1 on page 2 shows a high-level block diagram of the RS II MegaCore function reference design.

**Figure 1. Block Diagram of RS II MegaCore Reference Design**

The following sections describe the various components of the block diagram.

**RS Data Generator**

The RS data generator generates random data with 8 fix-channel interleave patterns. The codeword for this reference design has the number of channels (k)=8 and number of symbols (N)=239. The data stream is duplicated to provide double data path. The output of the RS data generator connects to the 2.5G RS II encoder component.

Figure 2 shows the output patterns from the RS data generator.

**Figure 2. RS Data Generator Output Waveform**
Figure 3 shows the internal components of the RS data generator.

Figure 3. RS Data Generator Block Diagram

Table 1 lists the functionality of each internal component of the RS data generator.

<table>
<thead>
<tr>
<th>Component</th>
<th>Type and references</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon-Memory Mapped (Avalon-MM) pipeline bridge</td>
<td>Standard component</td>
<td>Provides access to Avalon-MM interface of various components.</td>
</tr>
<tr>
<td>Avalon-Streaming (Avalon-ST) data pattern generator</td>
<td>Standard component</td>
<td>Implements a pseudo-random binary sequences (PRBS) generator to generate random data.</td>
</tr>
<tr>
<td>Avalon-ST data format adapter</td>
<td>Standard component</td>
<td>Adapts data width.</td>
</tr>
<tr>
<td>RS packet generator</td>
<td>Custom component</td>
<td>Appends fix-channel pattern to randomized data.</td>
</tr>
</tbody>
</table>
Table 2 lists the generator components that you can reconfigure to suit your verification objectives. To reconfigure the components, write to their registers using the base addresses listed in Table 2.

**Table 2. RS Data Generator Components Description**

<table>
<thead>
<tr>
<th>Components</th>
<th>Base Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon-ST data pattern generator (dpg_0)</td>
<td>0x0000</td>
</tr>
<tr>
<td>Avalon-ST data pattern generator (dpg_1)</td>
<td>0x0020</td>
</tr>
</tbody>
</table>

For details about the register maps of the RS data generator component, refer to *Avalon Streaming Data Pattern Generator and Checker Cores* chapter in the *Embedded Peripherals IP User Guide*.

### 2.5G RS II Encoder

The RS II encoder block consists of two RS II encoders implemented in parallel. This reference design uses the following variants:

- Number of symbols per codeword = 255
- Number of check symbols per codeword = 16
- Field polynomial = 285

For more information about RS II encoder, refer to the *Reed-Solomon II MegaCore Function User Guide*.

### RS Error Injection System

The RS error injection system comprises two error injection components. Each component injects error independently into each data path.

You can configure the following functions in the RS error injection system:

- Error injection per channel
- Maximum value of the internal counter
- Error magnitude

When the counter expires or reaches the maximum value, the RS error injection system injects errors into the data path of all enabled channels. The RS error injection system corrupts all error symbols with the same error magnitude. All enabled channels have the same number of error symbols per codeword.

Table 3 lists the RS error injection components that you can reconfigure to suit your verification objectives. To reconfigure the components, write to their registers using the base addresses listed in Table 3 and the register maps listed in Table 4.

**Table 3. RS Error Injection Reconfigurable Components and Base Addresses**

<table>
<thead>
<tr>
<th>Components</th>
<th>Base Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS error injection (err_injection_0)</td>
<td>0x0000</td>
</tr>
<tr>
<td>RS error injection (err_injection_1)</td>
<td>0x0020</td>
</tr>
</tbody>
</table>
Table 4 lists the RS error injection register maps.

Table 4. RS Error Injection Register Maps

<table>
<thead>
<tr>
<th>Registers</th>
<th>Description</th>
<th>Base Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Enables or disables each channel</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>1—Enables a channel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0—Disables a channel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit [0]—Enables or disables channel 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit [1]—Enables or disables channel 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit [2]—Enables or disables channel 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit [3]—Enables or disables channel 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit [4]—Enables or disables channel 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit [5]—Enables or disables channel 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit [6]—Enables or disables channel 6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit [7]—Enables or disables channel 7</td>
<td></td>
</tr>
<tr>
<td>error</td>
<td>Bit [7:0]—Error magnitude</td>
<td>0x04</td>
</tr>
<tr>
<td></td>
<td>Bit [15:8]—Maximum counter value</td>
<td></td>
</tr>
</tbody>
</table>

2.5G RS II Decoder

The RS II decoder block comprises two RS II decoders implemented in parallel. This reference design uses the following variants:

- Number of symbols per codeword = 255
- Number of check symbols per codeword = 16
- Field polynomial = 285

The field polynomial must be similar to the value you configure for the encoder.

For more information about RS II decoder, refer to the Reed-Solomon II MegaCore Function User Guide.
**RS Data Checker**

The RS data checker contains double data path, and it verifies each data path independently. The RS data checker removes the channel information and check symbols from the codeword, and verifies the correctness of the original data when the core receives a correctable or no error codeword. When the core receives no error codeword, the error counter of the RS data checker remains at zero. When the core receives an uncorrectable codeword, the error counter increments by the number of error bits. The RS data checker also provides the number of bits the core receives.

Figure 4 shows the internal components of the RS data checker.

---

**Figure 4. RS Data Checker Block Diagram**
Table 5 lists the functionality of each internal component of the RS data checker.

**Table 5. RS Data Checker Components Description**

<table>
<thead>
<tr>
<th>Component</th>
<th>Type and references</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon-MM pipeline bridge</td>
<td>Standard component</td>
<td>Provides access to the Avalon-MM interface of various components.</td>
</tr>
<tr>
<td>Avalon-ST data pattern checker</td>
<td>Standard component</td>
<td>Checks the correctness of the data. You must configure this component to match the PRBS of the data pattern generator.</td>
</tr>
<tr>
<td>RS statistics collector</td>
<td>Custom component</td>
<td>Accumulates the number of uncorrectable codeword. The core monitors each channel independently. The internal counter of a channel increments when the core asserts the <code>dec_fail</code> signal for the particular channel. If every codeword has no error or is correctable, all counters result to zero. If the core receives uncorrectable codeword, the counters reflect the number of uncorrectable codeword per channel. You can clear all internal counters by setting the soft reset bit of the control register high. The soft reset automatically clears upon reset.</td>
</tr>
<tr>
<td>RS packet remover</td>
<td>Custom component</td>
<td>Removes channel port and check symbols from the codeword. This method allows the data pattern checker to check for correctness of the original data.</td>
</tr>
</tbody>
</table>

You can configure the internal registers that control the RS data checker components through the Avalon-MM interface. The value of these registers depends on the time the RS data checker captures and reads the data.

Table 6 lists the RS data checker components that you can reconfigure to suit your verification objectives. To reconfigure the components, write to their registers using the base addresses listed in Table 6 and the register maps listed in Table 7.

**Table 6. RS Data Checker Reconfigurable Components and Base Addresses**

<table>
<thead>
<tr>
<th>Components</th>
<th>Base Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon-ST data pattern checker (dpc_0)</td>
<td>0x0000</td>
</tr>
<tr>
<td>RS statistics collector (statistics_0)</td>
<td>0x0040</td>
</tr>
<tr>
<td>Avalon-ST data pattern checker (dpc_1)</td>
<td>0x0080</td>
</tr>
<tr>
<td>RS statistics collector (statistics_1)</td>
<td>0x0100</td>
</tr>
</tbody>
</table>
Table 7 lists the statistics register maps.

<table>
<thead>
<tr>
<th>Registers</th>
<th>Description</th>
<th>Base Address</th>
</tr>
</thead>
</table>
| Control   | Bit [0]—Enables or disables a channel  
            1—Enables a channel  
            0—Disables a channel  
            Bit [1]—Soft reset on all internal counters  
            1—Reset  
            0—No reset  
            Soft reset automatically clears upon reset | 0x00 |
| Total error 0 | Total uncorrectable codeword for channel 0 | 0x04 |
| Total error 1 | Total uncorrectable codeword for channel 1 | 0x08 |
| Total error 2 | Total uncorrectable codeword for channel 2 | 0x0c |
| Total error 3 | Total uncorrectable codeword for channel 3 | 0x10 |
| Total error 4 | Total uncorrectable codeword for channel 4 | 0x14 |
| Total error 5 | Total uncorrectable codeword for channel 5 | 0x18 |
| Total error 6 | Total uncorrectable codeword for channel 6 | 0x1C |
| Total error 7 | Total uncorrectable codeword for channel 7 | 0x20 |
Getting Started

This section discusses the requirements and related procedures to demonstrate the reference design with the Stratix IV GX development board. This section contains the following topics:

- Hardware and Software Requirements
- Directory Structure
- Testbenches
- Reference Design Compilation and Verification in Hardware

Hardware and Software Requirements

The reference design requires the following hardware and software:

- Stratix IV GX FPGA development kit
- RS II MegaCore function
- Quartus® II software, version 10.1
- ModelSim®-AE 6.6, ModelSim-SE 6.6 or later

Directory Structure

Figure 5 shows the directory structure of the reference design and testbenches.

Figure 5. Directory Structure

```
rs_ii_2_5G_ref_design
  rs_ref_2_5G.qsys : 2.5G reference design.
  rs_gen.qsys: RS generator system.
  rs_chk.qsys: RS Checker system.
  rs_err_inj.qsys: RS Error Injection system.
  tb_2_5G.qsys: Testbench top level.
  hw_2_5G.qpf: Quartus project.
  hw_2_5G.qsf: Quartus project settings.
  hw_2_5G.sdc: Timing constraints.

ref_design_components
  Contains all custom components of the reference design.

testbench
  Contains all files for simulation testbench.

scripts
  Contains scripts for hardware testing using system console.
```
Testbenches

This section describes the architecture of the testbench, the components, and the simulation results. The register transfer level (RTL) simulation testbench provides RTL design verification at software level before implementing the reference design onto the development board.

Architecture

Figure 6 shows the architecture of the testbench.

Figure 6. Simulation Testbench Block Diagram

Components

The testbench comprises the following bus functional models (BFMs) and components:

- Avalon Clock Source BFM—Generates a 160MHz clock
- Avalon Reset Source BFM—Generates an active low reset signal
- Avalon-MM Master BFM—Provides access to the Avalon-MM interfaces.
- 2.5G RS II reference design

For more information about BFMs, refer to the Avalon Verification IP Suite User Guide.
Testbench Files

Table 8 lists the testbench files that are stored in clear text in the ..\testbench directory.

<table>
<thead>
<tr>
<th>Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs_reg_map_pkg.sv</td>
<td>A SystemVerilog HDL package that maps addresses to the Avalon-MM control registers.</td>
</tr>
<tr>
<td>rs_avalon_driver.sv</td>
<td>A SystemVerilog HDL driver that uses the BFMs to access the Avalon-MM interface.</td>
</tr>
<tr>
<td>tb.sv</td>
<td>The top-level testbench file. This file defines the test scenarios.</td>
</tr>
<tr>
<td>tb_run.tcl</td>
<td>A Tcl script that starts a simulation session in the ModelSim simulation software. It uses the generated mti_setup.tcl file.</td>
</tr>
<tr>
<td>wave.do</td>
<td>A signal tracing macro script used with the ModelSim simulation software to display testbench signals.</td>
</tr>
</tbody>
</table>

Test Scenarios

In this reference design, configure the error injection module to inject errors on all channel of data path 0. On data path 1, enable channels 4 to 7 only. Therefore, data on channels 0 to 3 of data path 1 is not corrupted.

The following two test scenarios occur:

- Correctable codeword
  The number of error symbols that are corrupted is within the capacity of the decoder to recover the original data. In this reference design, the number of error symbols must be less or equal to 8 symbols per codeword and per channel.

- Uncorrectable codeword
  The number of error symbols that are corrupted is greater than the capacity of the decoder to recover the original data. In this reference design, the number of error symbols is greater than 8 symbols per codeword and per channel.

Simulation Flow

After a simulated power-on reset, a typical software test case performs the following operations:

1. Clears all counters.
2. Initializes the RS II MegaCore function by performing these operations:
   a. Selects prbs-32 and enables checker.
   b. Enables error injection.
   c. Selects prbs-32 and enables generator
3. Performs test scenario 1: Correctable codeword.
4. Reads result from the checker.
5. Clears counters.
7. Reads result from the checker.
8. Stops the generator and checker.

**Simulating the Testbench with the ModelSim Simulator**

To simulate the reference design, follow these steps:

1. Extract the reference design to your working directory.
2. Launch the Quartus II software.
3. On the Tools menu, click **Options**. The **Options** dialog box appears.
4. In the **Category** list, select **EDA Tool Options**.
5. Specify the location of executable for the **ModelSim** and **ModelSim-Altera** EDA tools to your ModelSim installation path.
6. Click **OK**.
7. On the File menu, click **Open**.
8. Browse to the **tb_2_5G.qsys** file. Click **Open**.
9. On the Tools menu, click **Qsys**.
10. On the Tools menu in Qsys, click **Options**.
11. Add IP search path to `c:\<reference design directory>/ref_design_components`.
12. To launch Qsys and visualize the systems of the reference design, follow these optional steps:
   a. On the Tools menu, click **Qsys**.
   b. On the File menu in Qsys, click **Open**.
   c. Browse to `rs_ref_2_5G.qsys`, and click **Open** to visualize the 2.5G reference design.
   d. Browse to `rs_gen.qsys`, and click **Open** to visualize the RS data generator system.
   e. Browse to `rs_chk.qsys`, and click **Open** to visualize the RS data checker system.
13. Go back to `tb_2_5G.qsys`. On the Generation tab, ensure that you turn on **Create Verilog Simulation Model** checkbox, and click **Generate** to generate the system.
14. After generation is completed, launch the Modelsim-Altera software.
15. On the File menu, click **Change Directory**. Change the directory to `c:\<reference design directory>/testbench`.
16. In the Transcript console, activate your cursor on the `ModelSim>` command prompt.
17. Run the simulation by typing the following command at the command prompt:

   ```bash
do tb_run.tcl```
The ModelSim transcript pane in the Main window displays messages from the testbench reflecting the current task being performed. The \texttt{tb\_run.tcl} file compiles the generated \texttt{mti\_setup.tcl} script and other necessary testbench files. Then the Tcl file loads, generates a waveform, and runs the simulation. A pre-saved simulated waveform appears as \texttt{wave.do}.
Simulation Results and Timing Diagram

Upon successful simulation, the ModelSim-Altera software displays the simulation statistics on the Transcript console.

Figure 7 shows an extract of the simulation results.

Figure 7. Extract of Simulation Results from the Transcript Console

```verbatim
# Test scenario 1: Correctable codeword
# --------------------------
# Configuring Error Injection
# # Statistics for Datapath 0
# --------------------------
# Total dec_fail channel 0 = 0
# Total dec_fail channel 1 = 0
# Total dec_fail channel 2 = 0
# Total dec_fail channel 3 = 0
# Total dec_fail channel 4 = 0
# Total dec_fail channel 5 = 0
# Total dec_fail channel 6 = 0
# Total dec_fail channel 7 = 0
# Total number of bits received = 127200
# Total number of errors bits received = 0
# # Statistics for Datapath 1
# --------------------------
# Total dec_fail channel 0 = 0
# Total dec_fail channel 1 = 0
# Total dec_fail channel 2 = 0
# Total dec_fail channel 3 = 0
# Total dec_fail channel 4 = 0
# Total dec_fail channel 5 = 0
# Total dec_fail channel 6 = 0
# Total dec_fail channel 7 = 0
# Total number of bits received = 127776
# Total number of errors bits received = 0
# # Test scenario 2: Uncorrectable codeword
# --------------------------
# Configuring Error Injection
# # Statistics for Datapath 0
# --------------------------
# Total dec_fail channel 0 = 3
# Total dec_fail channel 1 = 3
# Total dec_fail channel 2 = 3
# Total dec_fail channel 3 = 3
# Total dec_fail channel 4 = 3
# Total dec_fail channel 5 = 3
# Total dec_fail channel 6 = 3
# Total dec_fail channel 7 = 3
# Total number of bits received = 149952
# Total number of errors bits received = 80
```
You can observe that the encoder appends the codeword with the appropriate check symbols. The core inserts errors into the data path before the core transmits the data to the decoder. The decoder either successfully corrects the corrupted data or issues a `dec_fail` on its error port when the decoder receives an uncorrectable codeword.
For OTN applications, there is no backpressure on the data transmission so the ready signal input to the encoder and decoder must always remain high. The encoder backpressures the generator to pause data generation. This method simplifies the implementation of the reference design. In an actual system, you might need to ensure full streaming by adding the appropriate delay in between codewords to compensate the number of clock cycles the core needs for the encoder to generate the check symbols.

**Reference Design Compilation and Verification in Hardware**

Figure 10 shows the hardware test system.

The hardware comprises a PLL, a JTAG master, the 2.5G RS II reference design, and the Stratix IV GX FPGA development board (EP4SGX230KF40C2). The PLL output generates a 160-MHz clock from a 50-MHz input clock to clock the reference design.
Hardware Setup

Figure 10 shows the hardware setup for the reference design.

Figure 10. Hardware Setup

For more information about the Stratix IV GX FPGA development board, refer to the \textit{Stratix IV GX FPGA Development Board Reference Manual}.

Pins Assignment

Table 10 lists the pins assignment for the hardware system.

Table 10. Pins Assignment

<table>
<thead>
<tr>
<th>Pin</th>
<th>Board Reference</th>
<th>Schematic Signal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC34</td>
<td>X7</td>
<td>CLKIN_50</td>
<td>50-MHz clock</td>
</tr>
<tr>
<td>AK35</td>
<td>S5</td>
<td>USER_PB0</td>
<td>Reset</td>
</tr>
</tbody>
</table>

Timing Considerations

The reference design provides an Synopsys Design Constraints (.sdc) file together with the project. This file contains timing constraints for the 50-MHz clock.
Hardware Test Files

Table 11 lists the Tcl files for the system console that are stored in the ..\scripts directory.

<table>
<thead>
<tr>
<th>Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs_reg_map.tcl</td>
<td>A Tcl script that maps address to the Avalon-MM control registers.</td>
</tr>
<tr>
<td>rs_api.tcl</td>
<td>A Tcl script that contains basic functions built upon the system console application programming interfaces (APIs) to access the registers through the Avalon-MM interface. This script contains APIs that the test.tcl file uses.</td>
</tr>
<tr>
<td>test.tcl</td>
<td>A Tcl script that runs two test scenarios, similar to the simulation testbench.</td>
</tr>
</tbody>
</table>

Hardware Test Flow

The hardware test flow is similar to the simulation flow. After a simulated power-on reset, a hardware test case performs the following operations:

1. Opens JTAG communication.
2. Initializes the RS II MegaCore function by performing these operations:
   a. Selects prbs-32 and enables the checker.
   b. Enables error injection.
   c. Selects prbs-32 and enables the generator.
3. Performs test scenario 1: Correctable codeword.
4. Reads result from the checker.
5. Clears the counters.
7. Reads result from the checker.
8. Stops the generator and the checker.
9. Closes JTAG communication.
Compiling the Reference Design
To compile and program the reference design onto the target device, follow these steps:

1. Connect the power supply to the board and connect the USB-Blaster™ download cable to the board’s USB Type-B connector as shown in Figure 10 on page 17.

2. Launch the Quartus II software and compile the reference design. To compile the reference design, follow these steps:
   a. On the File menu, click Open Project, navigate to \<directory>\hw_2_5G.qpf, and then click Open.
   b. On the Processing menu, click Start Compilation.
   c. A hw_2_5G .sof file is generated upon successful compilation.

3. To program the reference design onto your target FPGA device, follow these steps:
   a. On the Tools menu, click Programmer.
   b. The software automatically detects hw_2_5G .sof during compilation and it appears on the pop-up window.
   c. Click Start to download the Quartus II-generated file to the board. If hw_2_5G .sof does not appear in the pop-up window, click Add File, navigate to \<directory>\hw_2_5G .sof, and then click Open.

If you are not using the Stratix IV GX FPGA development board (EP4SGX230KF40C2), modify the Quartus II Settings File (.qsf) to suit your hardware.

Verifying the Reference Design in Hardware
After you have programmed the reference design onto your target FPGA device, follow these steps to verify your design and collect the statistics:

1. On the Tools menu, click Qsys.
2. On the Tools menu in Qsys, click System Console.
3. In the Tcl Console pane, change the directory by typing the following command:
   cd <reference design directory>/scripts

4. Type the following command in the Tcl Console pane to run the script and view the statistics:
   source test.tcl
Figure 11 shows an extract of the simulation statistics from the TCL Console pane. The hardware test statistics are similar to the testbench simulation statistics.

**Figure 11. Extract of Simulation Statistics from the TCL Console**

```
Info: Opened JTAG Master Service

Configuration

Configuring Checker
Clear internal counters on checker 0
Clear internal counters on checker 1
Clear and enable statistics 0
Clear and enable statistics 1
Select prbs-32 on checker 0
Select prbs-32 on checker 1
Enable checker 0
Enable checker 1

Configuring Error Injection
Enable error injection on all channels for datapath 0
Enable error injection on channels 4 to 7 for datapath 1

Configuring Generator
Select prbs-32 on generator 0
Select prbs-32 generator 1
Enable generator 0
Enable generator 1

Test scenario 1: Correctable codewords

Configuring Error Injection
Configuring error symbols to 1 and error magnitude to 1 on datapath 0
Configuring error symbols to 2 and error magnitude to 2 on datapath 1

Statistics for Datapath 0

----------
Total dec_fail channel 0 = 0x00000000
Total dec_fail channel 1 = 0x00000000
Total dec_fail channel 2 = 0x00000000
Total dec_fail channel 3 = 0x00000000
Total dec_fail channel 4 = 0x00000000
Total dec_fail channel 5 = 0x00000000
Total dec_fail channel 6 = 0x00000000
Total dec_fail channel 7 = 0x00000000
Total number of bits received = 0x01e44640
Total number of errors bits received = 0x00000000

----------
Statistics for Datapath 1

----------
Reed statistics
Total dec_fail channel 0 = 0x00000000
Total dec_fail channel 1 = 0x00000000
Total dec_fail channel 2 = 0x00000000
Total dec_fail channel 3 = 0x00000000
Total dec_fail channel 4 = 0x00000000
Total dec_fail channel 5 = 0x00000000
Total dec_fail channel 6 = 0x00000000
Total dec_fail channel 7 = 0x00000000
Total number of bits received = 0x05131360
Total number of errors bits received = 0x000315ef
```
You can use the SignalTap® II Logic Analyzer to view and debug internal signals. The SignalTap file (.stp) is not provided with this project. You must be familiar with the SignalTap II Logic Analyzer to configure and run the logic analyzer.

Figure 12 shows an example of the simulation waveform of the reference design. Observe that the core inserts errors to the datapath before the core transmits the data to the decoder.

![SignalTap II Logic Analyzer Waveform](image)

To execute the hardware test successfully, the `pll_locked` signal must be asserted.

### Performance and Resource Utilization

Table 12 lists the estimated resource utilization and performance of the reference design for Stratix IV GX (EP4SGX230KF40C2) device with speed grade -2.

<table>
<thead>
<tr>
<th>Components</th>
<th>Combinational ALUTs</th>
<th>Memory ALUTs</th>
<th>Logic Registers</th>
<th>Memory Block (M9K/M144K)</th>
<th>fMAX (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS data generator</td>
<td>659</td>
<td>0</td>
<td>558</td>
<td>0/0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>2.5G RS II encoder</td>
<td>374</td>
<td>40</td>
<td>186</td>
<td>6/0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>RS error injection</td>
<td>143</td>
<td>0</td>
<td>152</td>
<td>0/0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>2.5G RS II decoder</td>
<td>3,051</td>
<td>96</td>
<td>1,978</td>
<td>40/0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>RS data checker</td>
<td>1,684</td>
<td>0</td>
<td>1,742</td>
<td>0/0</td>
<td>&gt;160</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td><strong>6,148</strong></td>
<td><strong>136</strong></td>
<td><strong>4,657</strong></td>
<td><strong>46/0</strong></td>
<td><strong>&gt;160</strong></td>
</tr>
</tbody>
</table>
Table 13 lists the estimated resource utilization and performance of the reference design for Arria II GX (EP2AGX45DF25I3) device with speed grade -3.

### Table 13. Arria II GX Performance and Resource Utilization

<table>
<thead>
<tr>
<th>Components</th>
<th>Combinational ALUTs</th>
<th>Logic Registers</th>
<th>Memory Block (M9K)</th>
<th>fMAX (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS data generator</td>
<td>664</td>
<td>558</td>
<td>0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>2.5G RS II encoder</td>
<td>374</td>
<td>94</td>
<td>8</td>
<td>&gt;160</td>
</tr>
<tr>
<td>RS error injection</td>
<td>143</td>
<td>152</td>
<td>0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>2.5G RS II decoder</td>
<td>3,062</td>
<td>1,750</td>
<td>46</td>
<td>&gt;160</td>
</tr>
<tr>
<td>RS data checker</td>
<td>1,679</td>
<td>1,742</td>
<td>0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>Summary</td>
<td>6,159</td>
<td>4,337</td>
<td>54</td>
<td>&gt;160</td>
</tr>
</tbody>
</table>

Table 14 lists the estimated resource utilization and performance of the reference design for Cyclone IV GX (EP4CGX22BF14C6) device with speed grade -6.

### Table 14. Cyclone IV GX Performance and Resource Utilization

<table>
<thead>
<tr>
<th>Components</th>
<th>Combinational ALUTs</th>
<th>Logic Registers</th>
<th>Memory Block (M9K)</th>
<th>fMAX (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS data generator</td>
<td>1,126</td>
<td>558</td>
<td>0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>2.5G RS II encoder</td>
<td>448</td>
<td>88</td>
<td>8</td>
<td>&gt;160</td>
</tr>
<tr>
<td>RS error injection</td>
<td>209</td>
<td>152</td>
<td>0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>2.5G RS II decoder</td>
<td>4,796</td>
<td>1,720</td>
<td>46</td>
<td>&gt;160</td>
</tr>
<tr>
<td>RS data checker</td>
<td>2,678</td>
<td>1,742</td>
<td>0</td>
<td>&gt;160</td>
</tr>
<tr>
<td>Summary</td>
<td>9,551</td>
<td>4,301</td>
<td>54</td>
<td>&gt;160</td>
</tr>
</tbody>
</table>

### Conclusion

This application note demonstrates a basic application of Reed-Solomon algorithm to transmit data between the RS II encoder and decoder with the Stratix IV GX development board. You can use this reference design to control, test, and monitor RS coding and decoding operations through simulation and hardware implementation for integration into Altera FPGA designs.

### Document Revision History

Table 15 shows the revision history for this document.

### Table 15. Document Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 2011</td>
<td>1.0</td>
<td>Initial Release.</td>
</tr>
</tbody>
</table>