2.5G Reed-Solomon II MegaCore Function Reference Design

AN-642-1.0 Application Note

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

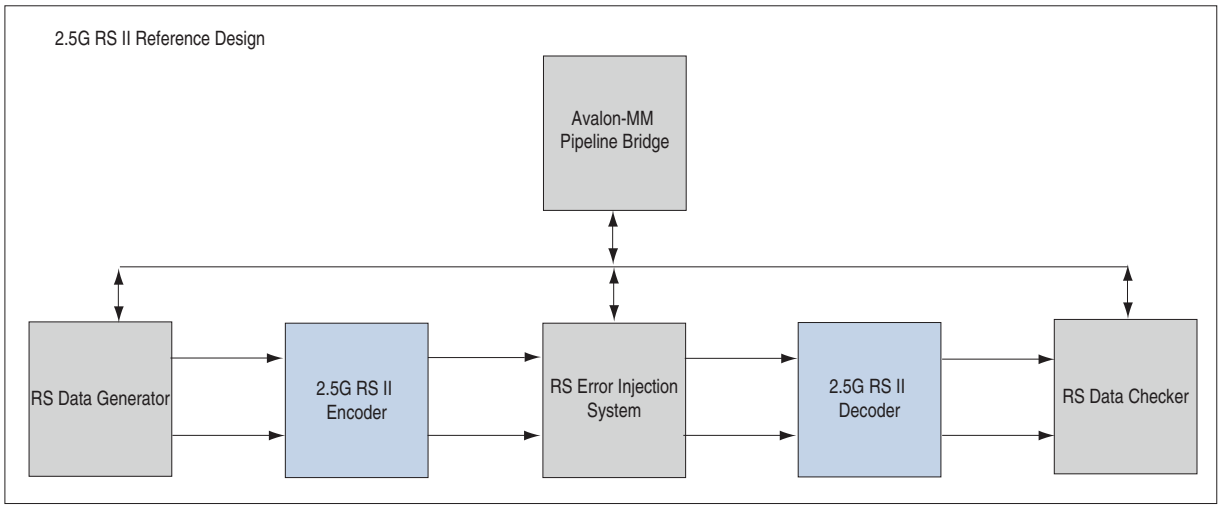
Functional Description

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.

101 Innovation Drive
San Jose, CA 95134
www.altera.com

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

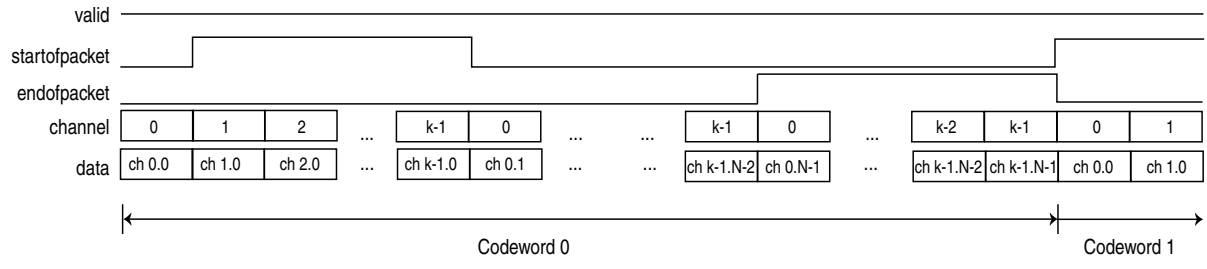


Figure 3 shows the internal components of the RS data generator.

Figure 3. RS Data Generator Block Diagram

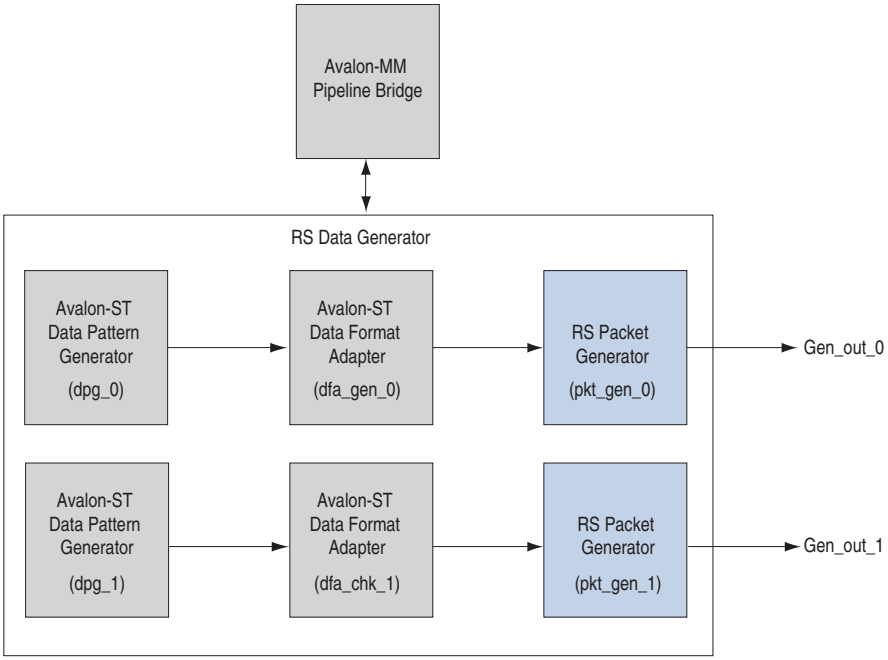


Table 1 lists the functionality of each internal component of the RS data generator.

Table 1. RS Data Generator Components Description

Component	Type and references	Description
Avalon-Memory Mapped (Avalon-MM) pipeline bridge	Standard component	Provides access to Avalon-MM interface of various components.
Avalon-Streaming (Avalon-ST) data pattern generator	Standard component	Implements a pseudo-random binary sequences (PRBS) generator to generate random data.
Avalon-ST data format adapter	Standard component	Adapts data width.
RS packet generator	Custom component	Appends fix-channel pattern to randomized data.

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

Components	Base Address
Avalon-ST data pattern generator (`dpg_0`)	0x0000
Avalon-ST data pattern generator (`dpg_1`)	0x0020

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

Components	Base Address
RS error injection (`err_injection_0`)	0x0000
RS error injection (`err_injection_1`)	0x0020

Table 4 lists the RS error injection register maps.

Table 4. RS Error Injection Register Maps

Registers	Description	Base Address
enable	Enables or disables each channel 1—Enables a channel 0—Disables a channel Bit [0]—Enables or disables channel 0 Bit [1]—Enables or disables channel 1 Bit [2]—Enables or disables channel 2 Bit [3]—Enables or disables channel 3 Bit [4]—Enables or disables channel 4 Bit [5]—Enables or disables channel 5 Bit [6]—Enables or disables channel 6 Bit [7]—Enables or disables channel 7	0x00
error	Bit [7:0]—Error magnitude Bit [15:8]—Maximum counter value	0x04

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

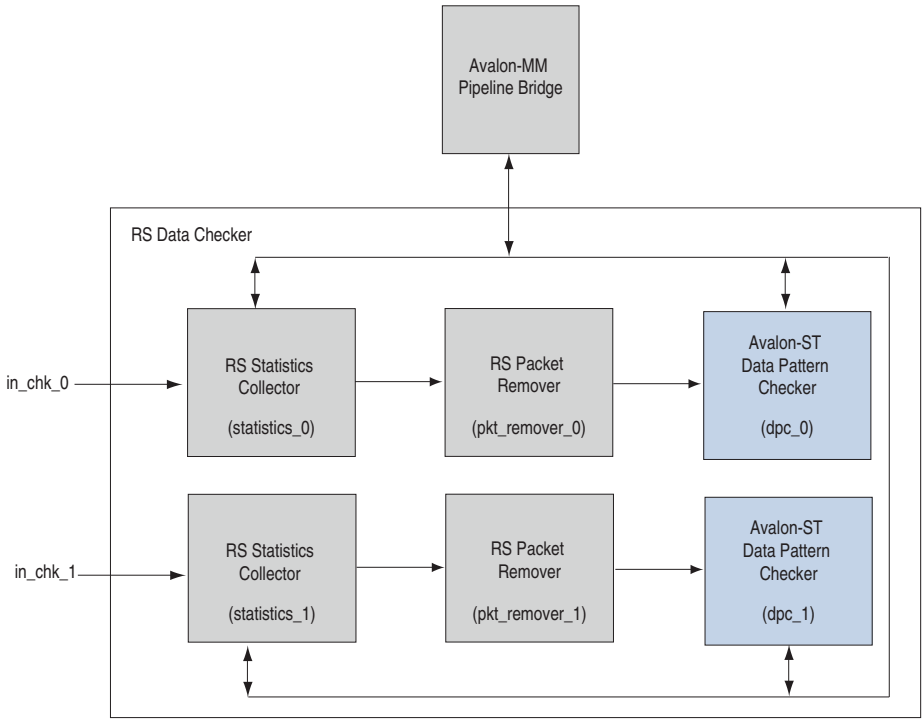


Table 5 lists the functionality of each internal component of the RS data checker.

Table 5. RS Data Checker Components Description

Component	Type and references	Description
Avalon-MM pipeline bridge	Standard component	Provides access to the Avalon-MM interface of various components.
Avalon-ST data pattern checker	Standard component	Checks the correctness of the data. You must configure this component to match the PRBS of the data pattern generator.
RS statistics collector	Custom component	Accumulates the number of uncorrectable codeword. The core monitors each channel independently. The internal counter of a channel increments when the core asserts the `dec_fail` 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.
RS packet remover	Custom component	Removes channel port and check symbols from the codeword. This method allows the data pattern checker to check for correctness of the original data.

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

Components	Base Address
Avalon-ST data pattern checker (`dpc_0`)	0x0000
RS statistics collector (`statistics_0`)	0x0040
Avalon-ST data pattern checker (`dpc_1`)	0x0080
RS statistics collector (`statistics_1`)	0x0100

Table 7 lists the statistics register maps.

Table 7. Statistics Register Maps

Registers	Description	Base Address
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

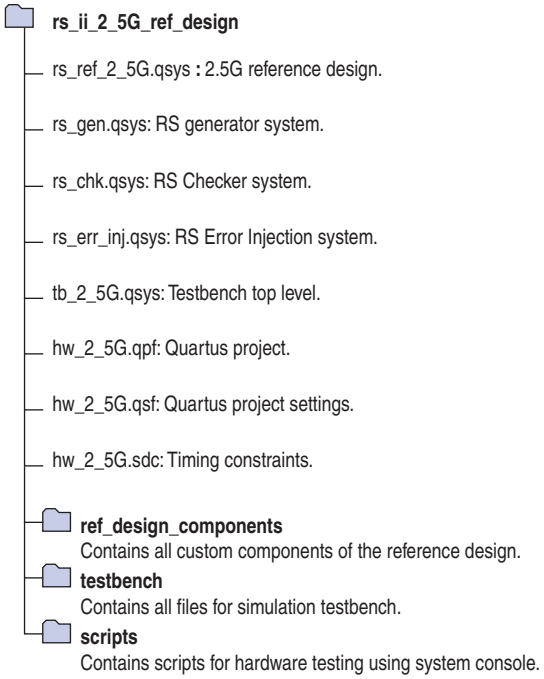
The reference design requires the following hardware and software:

- 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



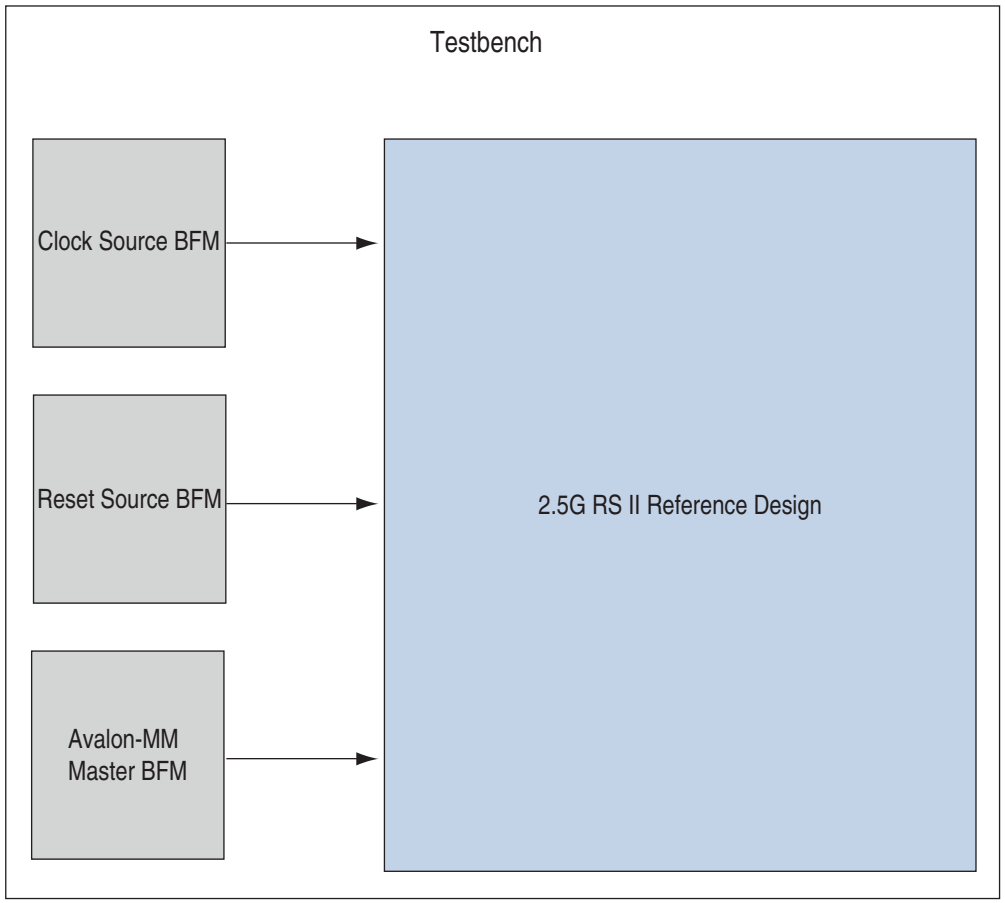
Testbenches

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 8. Testbench Files

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

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.
6. Performs test scenario 2: Uncorrectable codeword.
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.qsy**s. 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:

```
do tb_run.tcl ↵
```

The ModelSim transcript pane in the Main window displays messages from the testbench reflecting the current task being performed. The **tb_run.tcl** file compiles the generated **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 **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

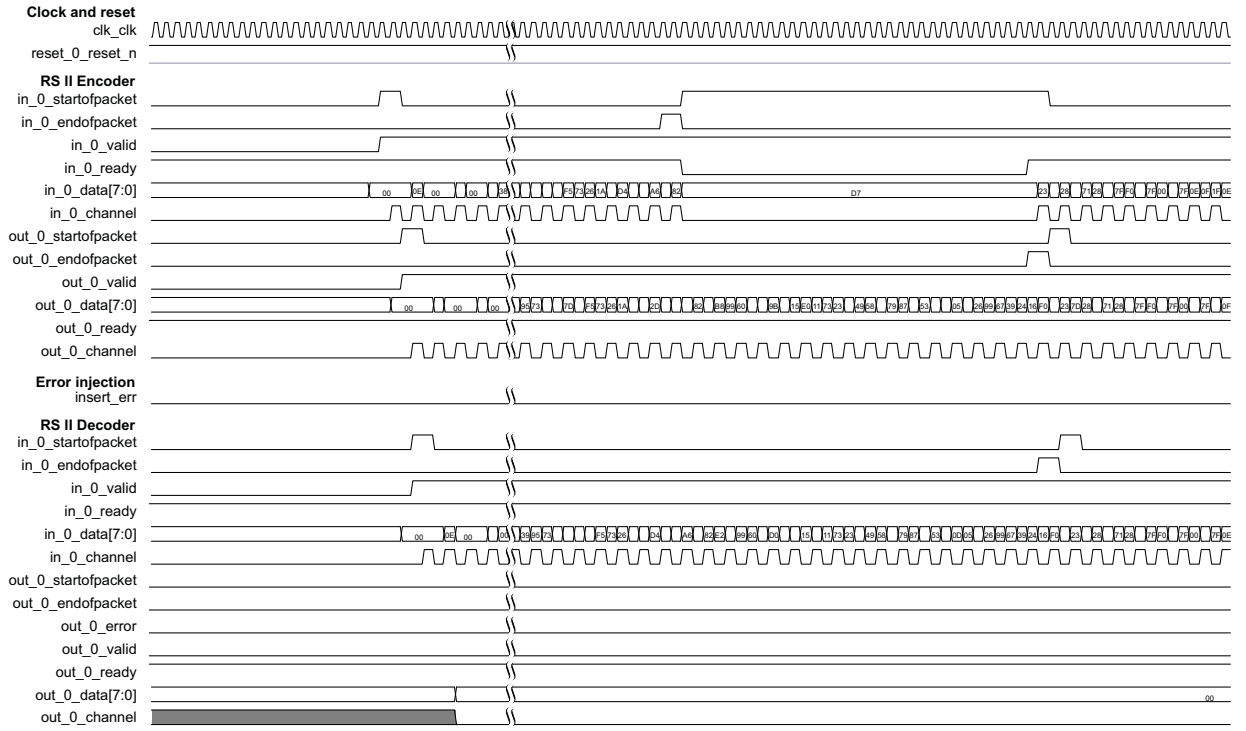
```
# -------------
# Test scenario 1: Correctable codeword
# -------------
#
# Configuring Error Injection
#
#
# -------------
# Statistics for Datapath 0
# -------------
#       Total dec_fail channel 0    = 0
#       Total dec_fail channel 1    = 0
#       Total dec_fail channel 2    = 0
#       Total 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    = 8
#       Total dec_fail channel 1    = 8
#       Total dec_fail channel 2    = 8
#       Total dec_fail channel 3    = 8
#       Total dec_fail channel 4    = 8
#       Total dec_fail channel 5    = 8
#       Total dec_fail channel 6    = 8
#       Total dec_fail channel 7    = 8
#       Total number of bits received   = 149952
#       Total number of errors bits received = 80
#
```

Table 9. Test Case Results

Test Case	Results
Test Case 1 : Correctable Codeword	Because no uncorrectable codeword is received, all statistics counters are zero. No error bits are received. The original data received matches the generated data. The number of bits received counter has been incremented.
Test Case 2: Uncorrectable Codeword	Because all channels of data path 0 has been enabled, all statistics counters are incremented. The values correspond to the number of uncorrectable codeword received at the moment when the counters are read. On data path 1, statistics counters of channels 4 to 7 are incremented because error injection was not enabled on channels 0 to 3. The number of error bits received and the number of bits received on both data path are displayed on the console.

Figure 8 shows the simulation timing diagram of the reference design on data path 0.

Figure 8. Simulation Timing Diagram On Data Path 0



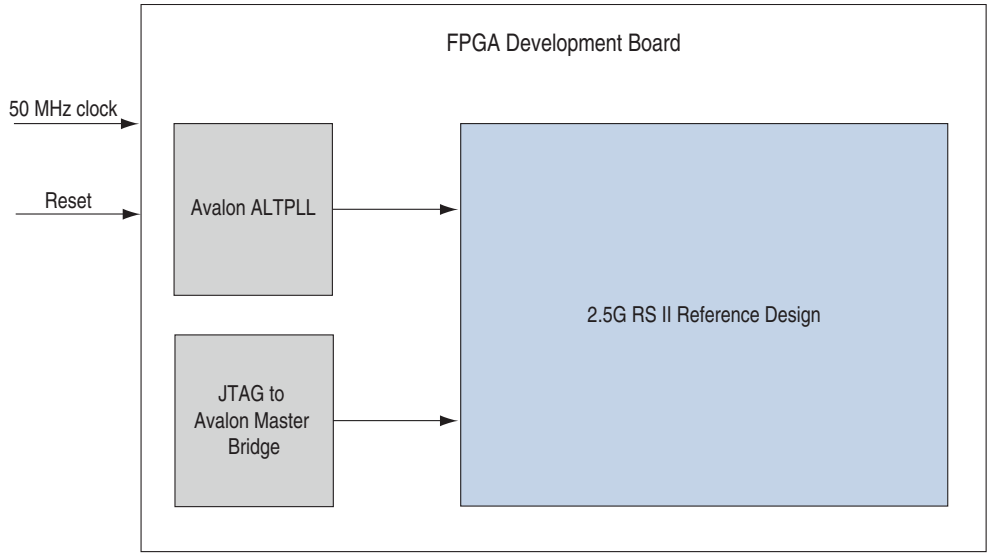
You can observe that the encoder appends the codeword with the appropriate check symbols. The core inserts errors into the data path before the core transmits the data to the decoder. The decoder either successfully corrects the corrupted data or issues a `dec_fail` on its error port when the decoder 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.

Figure 9. Hardware Test System Block Diagram



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 *Stratix IV GX FPGA Development Board Reference Manual*.

Pins Assignment

Table 10 lists the pins assignment for the hardware system.

Table 10. Pins Assignment

Pin	Board Reference	Schematic Signal Name	Description
AC34	X7	CLKIN_50	50-MHz clock
AK35	S5	USER_PB0	Reset

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 11. Hardware Test Files

Files	Description
rs_reg_map.tcl	A Tcl script that maps address to the Avalon-MM control registers.
rs_api.tcl	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.
test.tcl	A Tcl script that runs two test scenarios, similar to the simulation testbench.

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.
6. Performs test scenario 2: Uncorrectable codeword.
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

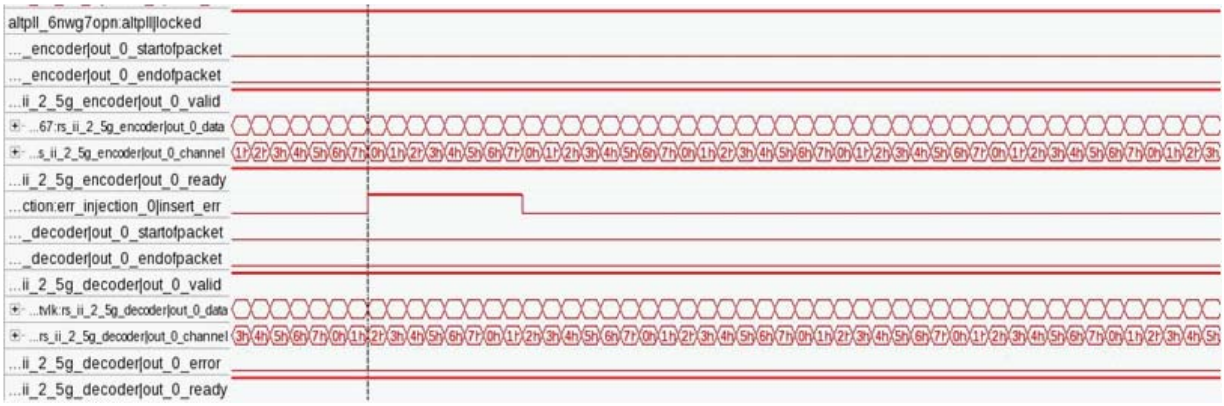
```
Info: Opened JTAG Master Service

================================================================================
                      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 8 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

------------

Read 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 = 0x000315af
```

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.

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

Figure 12. SignalTap II Logic Analyzer Waveform



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 12. Stratix IV GX Performance and Resource Utilization

Components	Combinational ALUTs	Memory ALUTs	Logic Registers	Memory Block (M9K/M144K)	f_{MAX} (MHz)
RS data generator	659	0	558	0/0	>160
2.5G RS II encoder	374	40	186	6/0	>160
RS error injection	143	0	152	0/0	>160
2.5G RS II decoder	3,051	96	1,978	40/0	>160
RS data checker	1,684	0	1,742	0/0	>160
Summary	6,148	136	4,657	46/0	>160

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

Components	Combinational ALUTs	Logic Registers	Memory Block (M9K)	f_{MAX} (MHz)
RS data generator	664	558	0	>160
2.5G RS II encoder	374	94	8	>160
RS error injection	143	152	0	>160
2.5G RS II decoder	3,062	1,750	46	>160
RS data checker	1,679	1,742	0	>160
Summary	6,159	4,337	54	>160

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

Components	Combinational ALUTs	Logic Registers	Memory Block (M9K)	f_{MAX} (MHz)
RS data generator	1,126	558	0	>160
2.5G RS II encoder	448	88	8	>160
RS error injection	209	152	0	>160
2.5G RS II decoder	4,796	1,720	46	>160
RS data checker	2,678	1,742	0	>160
Summary	9,551	4,301	54	>160

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

Date	Version	Changes
March 2011	1.0	Initial Release.