Intel® FPGA JESD204B Design Example User Guide for Intel® Arria® 10 Devices

Updated for Intel® Quartus® Prime Design Suite: **17.1**

Subscribe

Send Feedback

UG-DEX-A10-JESD204B | 2017.11.06

Latest document on the web: PDF | HTML

Contents

1 Intel® FPGA JESD204B Design Example User Guide for Intel® Arria® 10 Devices

The JESD204B IP core offers a design example for Intel® Arria® 10 devices. Generate the JESD204B IP core design example through the IP catalog in the Intel Quartus® Prime software.

Note: You can generate this JESD204B IP core design examples through the IP catalog in the Intel Quartus Prime Pro Edition software only.

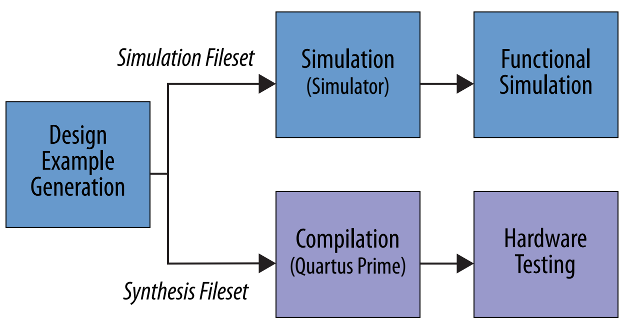
Related Links

- JESD204B IP Core Design Example User Guide
  Intel Arria 10 JESD204B IP Core Design Example User Guide for Intel Quartus Prime Standard Edition
- AN803: Implementing ADC-Arria 10 Multi-Link Design with JESD204B RX IP Core

1.1 Intel FPGA JESD204B Design Example for Intel Arria 10 Devices Quick Start Guide

The JESD204B IP core provides the capability of generating design examples for selected configurations.

Figure 1. Development Stages for the Design Example



1.1.1 Directory Structure

The JESD204B design example file directories contain generated files for the design examples.

Figure 2. Directory Structure for the JESD204B Design Example

Note:

1. *Directory 'system_console' only generated when 'Data Path Only' design example is generated.*
2. *Directory 'software' only generated when 'NIOS Control' design example is generated.*

Table 1. Directory and File Description

Directory/File	Description
`ed_sim`	The folder that contains simulation testbench files
`ed_sim/testbench/models`	The folder that contains the testbench and source files
`ed_sim/testbench/setup_scripts`	The folder that contains the test flow setup scripts
`ed_sim/testbench/pattern`	The folder that contains the source files for the pattern generator/checker
`ed_sim/testbench/transport_layer`	The folder that contains the source files for the transport layer
`ed_sim/testbench/aldec`	The folder that contains the test flow run scripts for Riviera simulator. Also serves as the working directory for the simulator.
`ed_sim/testbench/cadence`	The folder that contains the test flow run scripts for NCSim simulator. Also serves as the working directory for the simulator.

continued...

Directory/File	Description
`ed_sim/testbench/mentor`	The folder that contains the test flow run scripts for Modelsim simulator. Also serves as the working directory for the simulator.
`ed_sim/testbench/synopsys/vcs`	The folder that contains the test flow run scripts for VCS simulator. Also serves as the working directory for the simulator.
`ed_sim/testbench/synopsys/vcsmx`	The folder that contains the test flow run scripts for VCSMX simulator. Also serves as the working directory for the simulator.
`ed_synth`	The folder that contains design example synthesizable components
`ed_synth/ip`	The folder that contains Platform Designer instantiated IP modules
`ed_synth/altera_jesd204_ed_qsys_<data path>`	The folder that contains Platform Designer generated modules from the jesd204_ed_qsys_<data path>.qsys system
`ed_synth/altera_jesd204_subsystem_<data path>`	The folder that contains Platform Designer generated modules from the jesd204_subsystem_<data path>.qsys system
`ed_synth/pattern`	The folder that contains the source files for the pattern generator/checker
`ed_synth/transport_layer`	The folder that contains the source files for the transport layer
`altera_jesd204b_ed_<data path>.qpf` `altera_jesd204b_ed_<data path>.qsf`	Quartus project and settings files
`altera_jesd204b_ed_qsys_<data path>.qsys` `altera_jesd204b_subsystem_<data path>.qsys`	Platform Designer subsystems
`altera_jesd204b_ed_<data path>.sv`	Top level HDL source file
`altera_jesd204_ed_<data path>.sdc`	Top level design constraints file
`ed_synth/system_console`	The folder that contains all files necessary to run scripts in System Console (See Design Example Files for more details on folder content)
`ed_synth/software`	The folder that contains all files necessary to run the software control flow using Nios soft processor (See Design Example Files on page 36 for more details on folder content)
`*.v`	Miscellaneous source files
`ip_sim`	The folder that contains the simulation script to generate the JESD204B IP Core Verilog/VHDL simulation model.

1.1.2 Generating the Design

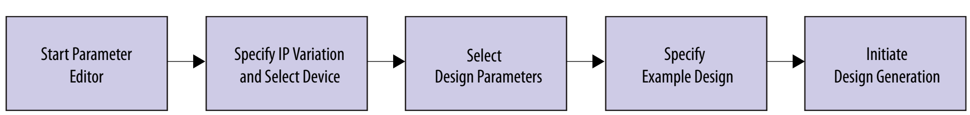
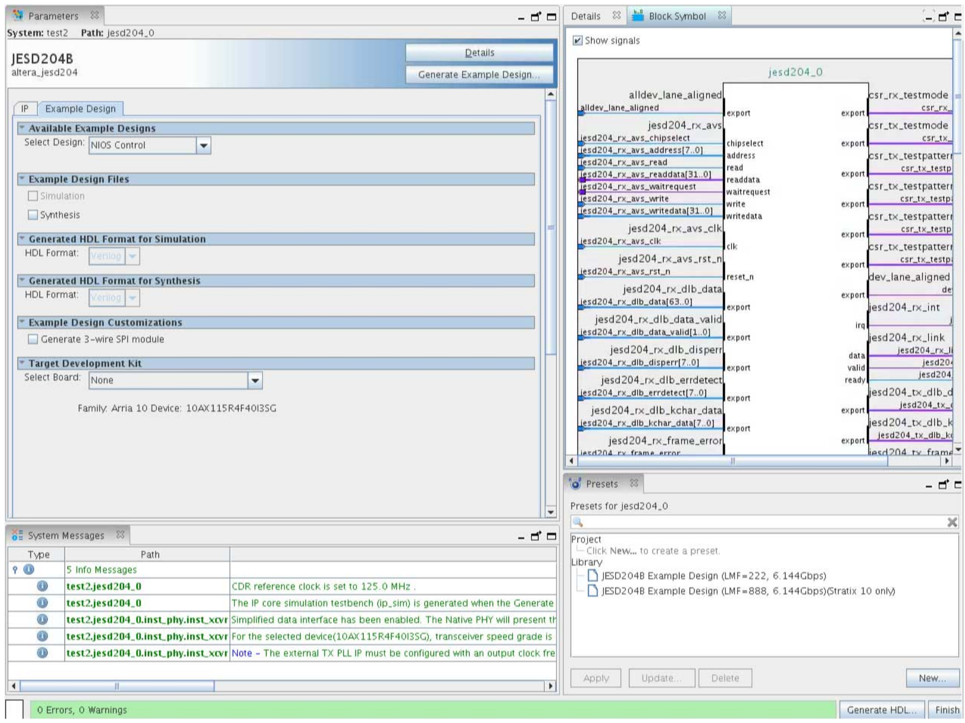


Figure 3. Example Design Tab



To generate the design example from the IP parameter editor:

1. In the IP Catalog (`Tools > IP Catalog`), locate and select **JESD204B**. The IP parameter editor appears.
2. Specify a top-level name and the folder for your custom IP variation, and the target device. Click **OK**.
3. Select a design from the **Presets** library by double-clicking the desired preset. When you select a design, the system automatically populates the IP parameters for the design.

   Note: If you select another design, the settings of the IP parameters change accordingly.
4. You can customize the preset parameter values according to your specifications. Under the **IP** tab, specify the JESD204B IP core parameters for your design.

   Note: Only a limited range of parameter combinations are allowed for design generation. Please refer to the Supported Configurations on page 16 section for more details. Parameter combinations that are not supported results in the **Available Example Designs** drop down box to display only **None** as the default.
5. Under the **Example Design** tab, specify the design example parameters as described in *Design Example Parameters*.

   Note: To generate the design example for hardware testing on selected Intel development kits, select the appropriate target development kit from the **Target Development Kit** drop down box.
6. Click the **Generate Example Design** button.

The software generates all design files in the sub-directories. These files are required to run simulation, compilation, and hardware testing.

Related Links

- Presets on page 18
- Supported Configurations on page 16

1.1.2.1 Design Example Parameters

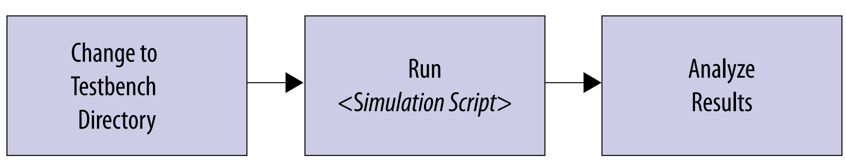
The JESD204B IP parameter editor includes a **Example Design** tab for you to specify certain parameters before generating the design example.

Table 2. Parameters in the Example Design Tab

Parameter	Options	Description
Available Example Designs	None (Default)	No design examples selected.
	System Console Control	Design example with System Console control.
	Nios Control	Design example with Nios soft processor control.[1]
Example Design Files	Simulation	Generate simulation fileset.[2]
	Synthesis	Generate synthesis fileset.
Generated HDL Format for Simulation	Verilog (Default)	Verilog HDL format for entire simulation fileset.
	VHDL	VHDL Platform Designer generated top-level wrapper file set.
Generated HDL Format for Synthesis	Verilog (Default)	Verilog HDL format for synthesis fileset.
Example Design Customizations	Generate 3-wire SPI module	Check to enable 3-wire SPI interface instead of 4-wire SPI interface.
Target Development Kit	None (Default)	No target development kit selected.
	Intel Arria 10GX FPGA Development Kit	Design example targets Intel Arria 10 GX FPGA Development Kit

1.1.3 Simulating the Design

These general steps describe how to run the design example simulation. For specific commands for each design example variant, refer to its respective section.



[1] Only supports synthesis fileset. No simulation fileset is available for this option. Please select the System Console Control design example to generate simulation fileset.

[2] Not applicable for Nios Control design example.

To simulate the design, perform the following steps:

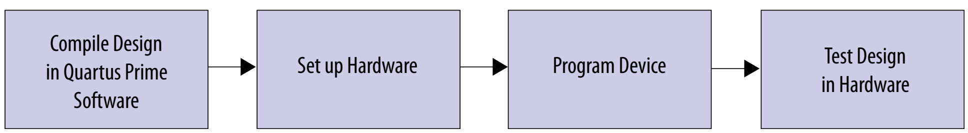
1. Change the working directory to <example_design_directory>**/ed_sim/testbench**/<Simulator>.
2. Run the simulation script for the simulator of your choice shown in the following table

Simulator	Command
Riviera	`do run_tb_top.tcl`
NCSim	`sh run_tb_top.sh`
Modelsim	`do run_tb_top.tcl`
VCS/VSCMX	`sh run_tb_top.sh`

The simulation ends with messages that indicate whether the run was successful or not. Refer to *Simulation Message and Description* table in Testbench on page 34 for more information on messages reported by the simulation flow.

1.1.4 Compiling and Testing the Design

The JESD204B IP Core parameter editor allows you to run the design example on a target development kit.



If you are performing hardware testing on the selected Intel development kits, generate the design example with the appropriate target development kit selected. Refer to the instructions in Generating the Design on page 5.

Note: Running the hardware test with the design generated as-is is only possible when the JESD204B IP core is configured in duplex data path mode (i.e with both TX and RX data paths present). Make your own modifications to the design to run the hardware test if generating a simplex data path design.

The generated design has pre-assigned pins that target the relevant boards. The following table describe the board connectivity of key design ports for all supported target development kits.

Table 3. Intel Arria 10 Signal Integrity Development Kit Board Connectivity

Port Name	Port Description	Board Component	Component Description
global_rst_n	Global reset	S3	User PB0 push-button
device_clk	Reference clock input	U14	Si5338 clock generator (CLK1A/B)
mgmt_clk	Control clock	X3	Si570 clock generator (100Mhz)
tx_serial_data	TX serial data	J1	FMC port A connector
rx_serial_data	RX serial data	J1	FMC port A connector

Perform the following steps to compile the design and program the development board:

1. Launch the Intel Quartus Prime software and compile the design (**Processing ➤ Start Compilation**).

   The timing constraints and pin assignments for the design example and the design components are automatically loaded during design example compilation.

2. Connect the development board to the host computer either by connecting a USB cable to the on-board Intel FPGA Intel FPGA Download Cable II component or using an external Intel FPGA Intel FPGA Download Cable II module to connect to the external JTAG connector.

3. Launch the **Clock Control** application that is included with the development board and set the clock settings according to the selected data rate.

   Note: Refer to the Intel Arria 10 GX FPGA Development Kit documentation for more information on using the **Clock Control** application.

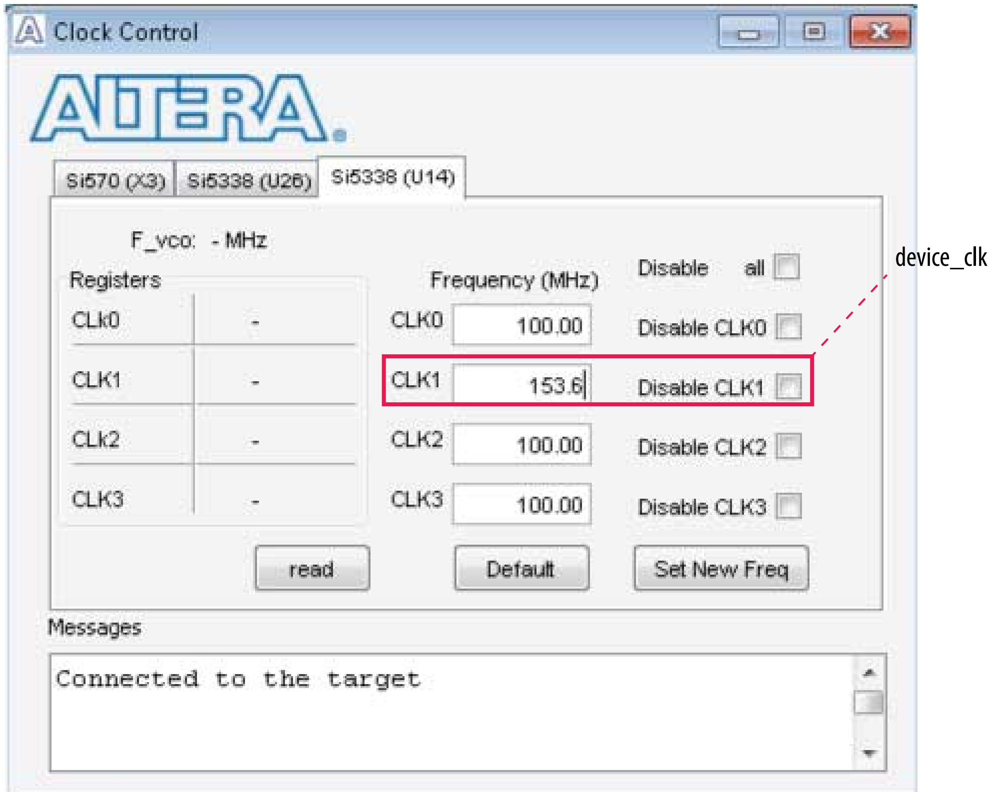
Table 4. Clock Setting

Clock Name	Clock Frequency
`device_clk`	Select the frequencies in the **PLL/CDR Reference Clock Frequency** drop down menu of the IP Parameter Editor.(3)
`mgmt_clk`	100 Mhz

(3) The design example uses 153.6 MHz clock frequency for the design with data rate of 6.144 Gbps.

Figure 4. Clock Control GUI setting

This example shows the clock control GUI setting for 6.144 Gbps data rate:



4. If you are performing external FMC loopback test, affix the FMC loopback card to the FMC port A connector.
5. Configure the FPGA on the development board with the generated programming file (`.sof` file) using the Intel Quartus Prime **Programmer**.

Related Links

- JESD204B IP Core User Guide
- Intel FPGA JESD204B RX Address Map and Register Definitions
- Intel FPGA JESD204B TX Address Map and Register Definitions

1.1.4.1 Hardware Test for System Console Control Design Example

Perform the following instructions to run the hardware test for the design example using the System Control tool.

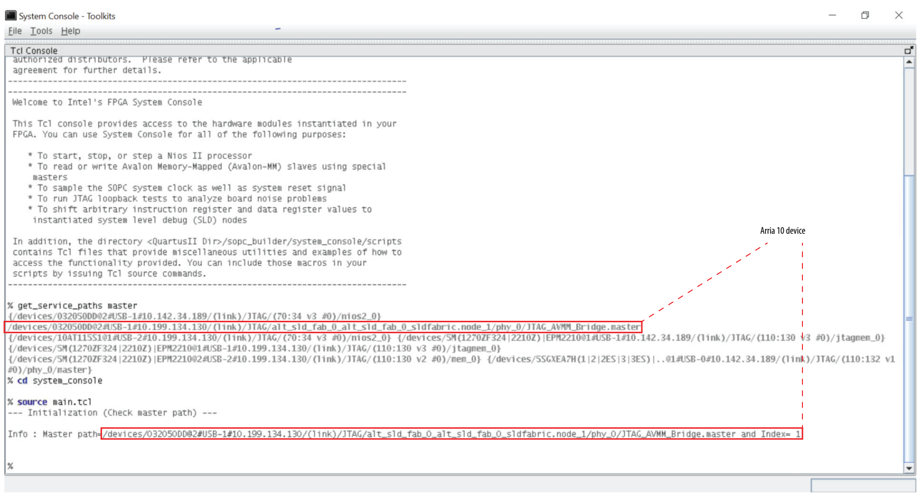
Note: This hardware test assumes that the System Console Control design is configured in duplex mode. Make your own modifications if using simplex mode design.

1. Launch the System Console tool from Intel Quartus Prime (**Tools ➤ System Debugging Tools ➤ System Console**).
2. In the **TCL Console** command prompt, type `get_service_paths master` to print a list of devices connected to your JTAG chain.
3. Open the `main.tcl` Tcl script located in the `System Console` directory in any text editor of your choice and locate the following line.

```
set master_index [expr {$master_list_length - <your offset>}]
```

4. Adjust the `master_index` offset as necessary to reflect your JTAG chain configuration such that the `master_index` always points to the Intel Arria 10 device and save the file.
5. In the **TCL Console** command prompt, navigate to the `system_console` directory (`cd system_console`) and execute the `main.tcl` script (`source main.tcl`). Your **TCL Console** window should resemble the following figure.

Figure 5. Source main.tcl

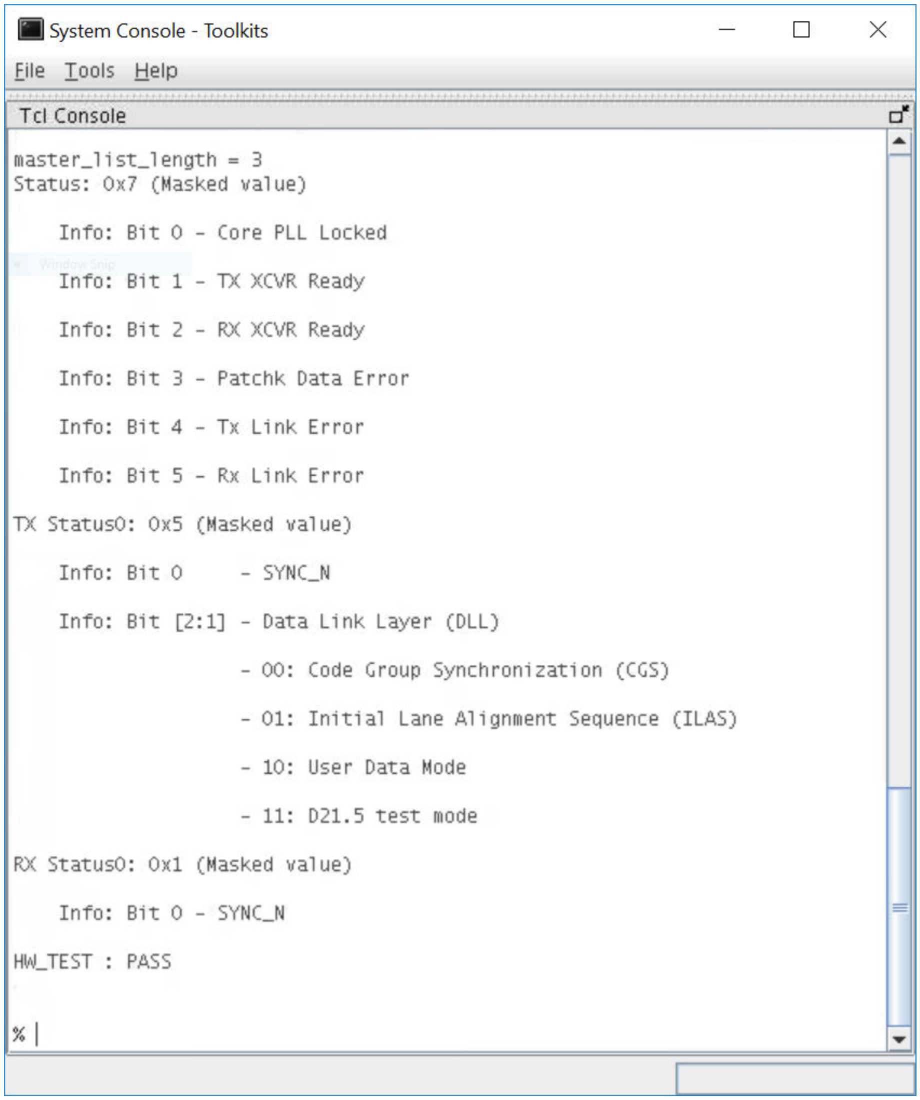


6. Type `start_basic_test` at the command prompt to execute the link setup and test procedure.

   This procedure executes a set of instructions to set up the pattern generator and checker to transmit and check PRBS pattern, configure the JESD204B IP PHY internal serial loopback mode and report link status.

   The following figure illustrates the expected result from a successful link setup and test.

Figure 6. Successful Test in the System Console



7. In the event that the test failed due to a lane deskew error, use the **rbd_offset** procedure (described in the following table) to offset the default RBD setting. Refer to the JESD204B IP User Guide for more details on using the RBD offset.

Table 5. Procedures in the main.tcl System Console Script

Table describes useful procedures in the **main.tcl** that may be helpful in debugging.

Procedure	Values	Description
get_service_paths	{master}	Reports all devices that are connected to the JTAG chain. Use this information to set the master index to point to the Intel Arria 10 device
get_master_index	N/A	Set the targeted device master index. Use get_service_paths master to determine the offset of the Intel Arria 10 device in the JTAG chain, and edit the offset in this procedure accordingly.

continued...

Procedure	Values	Description
start_basic_test	N/A	Main procedure that sets up link serial loopback mode, pattern generator and checker test mode, pulses sysref and reports link status
reset	N/A	Global reset
force_link_frame_reset	{0,1}	0: Deassert link and frame resets 1: Assert and hold link and frame resets *Note:* Link and frame clock domains should be held in reset while writing to JESD204B IP CSR
sloopback	{0,1}	0: Disable internal serial loopback 1: Enable internal serial loopback
set_testmode	{alt, ramp, prbs}	alt: Set pattern generator and checker to alternate pattern ramp: Set pattern generator and checker to ramp pattern prbs: Set pattern generator and checker to PRBS pattern
rbd_offset	{integer}	Adjust RBD offset value to eliminate RX lane deskew error. Refer to the JESD204B IP Core User Guide for more details.
sysref	N/A	Single pulse sysref
read_status_pio	N/A	Read status PIO registers. PIO status configuration: Bit 0 — Core PLL locked Bit 1 — TX transceiver ready Bit 2 — RX transceiver ready Bit 3 — Pattern checker mismatch error Bit 4 — TX link error (use read_err_status procedure to report error description) Bit 5 — RX link error (use read_err_status procedure to report error description)
read_err_status	N/A	Read JESD204B IP error status registers. Refer to the JESD204B IP register maps for detailed description of status registers.
clear_err_status	N/A	Clear JESD204B IP error status registers
read_rx_status0	N/A	Read JESD204B IP rx_status0 register. Refer to the JESD204B IP register maps for detailed description of status registers
read_tx_status0	N/A	Read JESD204B IP tx_status0 register. Refer to the JESD204B IP register maps for detailed description of status registers.
read_rx_syncn_sysref_ctrl	N/A	Read JESD204B IP syncn_sysref_ctrl register. Refer to the JESD204B IP register maps for detailed description of status registers
wait_seconds	{integer}	Wait for {integer} seconds
wait_minutes	{integer}	Wait for {integer} minutes

Related Links

- JESD204B IP Core User Guide
- Intel FPGA JESD204B RX Address Map and Register Definitions
- Intel FPGA JESD204B TX Address Map and Register Definitions

1.1.4.2 Hardware Test for Nios Control Design Example

Follow the instructions below to run the hardware test for the Nios Control design example.

Note: This hardware test assumes that the Nios Control design is configured in duplex mode. Make your own modifications if using simplex mode design.

1. 1. Launch the Nios II Software Build Tools for Eclipse tool from Intel Quartus Prime (**Tools ➤ Nios II Software Build Tools for Eclipse**).
2. In the **Select a workspace** dialog box, navigate to the software workspace **<design example>/software**.
3. Create a new Nios II application and board support package (BSP) from the template (**File ➤ New ➤ Nios II Application and BSP From Template**).
4. In the **Nios II Application and BSP From Template** window, enter the following information:
   a. **SOPC Information File Name**: `<design example>/altera_jesd204_ed_qsys_RX_TX/altera_jesd204_ed_qsys_RX_TX.sopcinfo`
   b. **Project name**: `<software project>`
   c. **User default location**: **Checked**
   d. **Templates**: Blank Project
5. Click **Next**. Verify that the default BSP name is `<software project>_bsp`, then click **Finish**. The Nios II application project and BSP appears in the **Project Explorer** window.
6. In the **Project Explorer** window, right-click the `<software project>_bsp project`, navigate to Nios II and click **Generate**. This regenerates the BSP files based on your most current compiled Intel Quartus Prime project settings.

   Note: Whenever you modify and recompile the Intel Quartus Prime project, you must regenerate the BSP files.
7. Import the design example source (`*.c`) and header (`*.h`) files into the application directory. In the **Project Explorer** window, right click on the `<software project>` project and click **Import**.
8. In the Import window, select **General ➤ File System as the import source** and click **Next**.
9. Browse to the `<design example>/software/source` directory. Check the source box on the left panel. This selects all the source and header files in the source directory. Verify that the list of source and header files are as follows:
   a. `altera_jesd204_regs.h`
   b. `functions.h`
   c. `macros.h`
   d. `main.h`
   e. `macros.c`
   f. `main.c`
10. Verify that the destination folder is `<software project>`. Click **Finish**.

    All the source and header files should be imported into the <software project> project directory.

11. Right-click the `<software project>_bsp` project, navigate to **Nios II ➤ BSP Editor** . Under the **Drivers** tab, check the **enable_small_driver** box of the **altera_avalon_jtag_uart_driver** group and click **Generate**. This setting allows the compilation to proceed without connecting the interrupt ports of the JTAG UART module. After the BSP files have been generated, click **Exit**.
12. Expand the `<software project>` application project in the **Project Explorer** window and verify that the folder contains all the source and header files.
13. To compile the C code, navigate to **Project ➤ Build All**.

    The compiler now compiles the C code into executable code.
14. To download the executable code to the development board, navigate to the **Run ➤ Run Configurations**. In the **Run Configurations** window, double-click **Nios II Hardware** on the left panel.
15. Verify that all run configurations are correct, then click **Run**.

The Intel Quartus Prime software downloads the executable code onto the board and the Nios II processor executes the code. The code performs the JESD204B link initialization sequence and exits. You can view the code execution results on the **Nios II Console** tab. The **Nios II Console** is the standard input/output for the executable code. At the end of the initialization sequence, the code prints the JESD204B link status to the console. The following figure illustrates the expected result from a successful link initialization.

The following tables list the expected values of the link status register report.

Table 6. TX Status 0 Register Expected Values

Bit	Name	Description	Expected Binary Value
[0]	SYNC_N value	0: Receiver is not in sync 1: Link is in sync	1
[2:1]	Data Link Layer (DLL) state	00: Code Group Synchronization (CGS) 01: Initial Lane Alignment Sequence (ILAS) 10: User Data Mode 11: D21.5 test mode	10

Table 7. RX Status 0 Register Expected Values

Bit	Name	Description	Expected Binary Value
[0]	SYNC_N value	0: Receiver is not in sync 1: Link is in sync	1
Others	N/A	N/A	*Don't care*

1.2 Design Example Detailed Description

1.2.1 Features

This design example has the following key features:

- Control mechanisms:
  - System Console using Tcl script control mechanism
  - Nios II soft processor using embedded C code
- Synthesis and simulation flows—Nios II soft processor control design only supports synthesis flow
- Configurable transport layer and pattern generator and checker modules
- Power-on self test with the following configurable test patterns:
  - Alternating
  - Ramp
  - PRBS
- Supports simplex (RX only, TX only) and duplex (both RX and TX) data path modes
- Supports transceiver dynamic reconfiguration mode
- Supports option for 3-wire SPI

1.2.2 Hardware and Software Requirements

Intel uses the following hardware and software to test the example designs:

- Intel Quartus Prime software
- Intel Arria 10 GX FPGA Development Kit

1.2.3 Supported Configurations

The design examples only support a limited set of JESD204B IP core parameter configurations. The IP parameter editor allows you to generate a design example only if the parameter configurations matches the following table.

Note: If you are not able to generate a design example that fully matches your desired parameter settings, choose the closest allowable parameter values for generation. Modify the post-generated design parameters manually in the Intel Quartus Prime software to match your desire parameter settings. Refer to the *JESD204B IP Core User Guide* for more details on the rules and ranges that govern each IP core and transport layer parameter. Refer to *Customizing the Design Example* for more information about customizing the design example.

Table 8. Supported JESD204B IP Core Parameter Configurations

Table lists the parameters for the JESD204B IP core. The JESD204B IP core parameters are governed by various rules and ranges that are described in the *JESD204B IP Core User Guide*. Please refer to the *JESD204B IP Core User Guide* for more details on the legal parameter values. The value ranges given below should be considered as a subset of the allowable values described in the *JESD204B IP Core User Guide*.

JESD204B IP Parameters	Values
Wrapper Options	Both Base and Phy
Data Path	• Receiver • Transmitter • Duplex
JESD204B Subclass	1
Data Rate	Any valid value[4]
PCS Option	• Enabled Hard PCS • Enabled Soft PCS
Bonding Mode	• Bonded • Non-bonded
PLL/CDR Reference Clock Frequency	Any valid value
Enable Bit Reversal and Byte Reversal	Any valid value
Enable Transceiver Dynamic Reconfiguration	Any valid value
L	• 1 • 2 • 4 • 6[5] • 8
M	• 1 • 2 • 3[6] • 4 • 8 • 16 • 32
Enable manual F configuration	• No • Yes only for the following configuration: L=8, M=8, F=8, S=5, N'=12, N=12
F	• Auto calculated • Manual F configuration only allowed for the following configuration: L=8, M=8, F=8, S=5, N'=12, N=12
N	Integer, range 12 – 16

continued...

(4) Refer to *JESD204B IP Core User Guide* for more details on maximum and minimum data rates for your target device.

(5) L=6 is only allowed when F=1

(6) M=3 is only allowed for L=6

JESD204B IP Parameters	Values
N'	• 16 • 12 only for the following configuration: L=8, M=8, F=8, S=5, N=12
S	Any valid value
K	Any valid value
Enable Scramble (SCR)	Any valid value
CS	Integer, range 0 – 3
CF	0
High Density User Data Format (HD)	• 0 • 1 only for F=1
Enable Error Code Correction (ECC_EN)	Any valid value

Related Links

- JESD204B IP Core User Guide
- Customizing the Design Example on page 49

1.2.4 Presets

Standard presets allow instant entry of pre-selected parameter values in the **IP** and **Example Design** tabs. Select the presets at the lower right window in the parameter editor shown in Figure 3 on page 6.

The parameter values chosen for the presets belong to the group of supported JESD204B IP configurations for design example generation. You can select one of the presets available for your target device to quickly generate a design example without having to manually set each parameter in the IP tab and verifying that the parameter matches the supported configurations set. You can manually change any of the IP and example design parameters in the Platform Designer user interface after selecting a preset. However, you must ensure that your parameter selection falls within the supported configuration ranges detailed in Supported Configurations on page 16 for example design generation to be successful.

Note: Selecting a preset overwrites any pre-existing parameter selections for the IP core under the IP tab.

Table 9. Preset Settings

JESD204B IP Parameters	Preset 1	Preset 2
Wrapper Options	Both Base and Phy	Both Base and Phy
Data Path	Duplex	Duplex
JESD204B Subclass	1	1
Data Rate	6144 Mbps	6144 Mbps
PCS Option	Enabled Hard PCS	Enabled Hard PCS
Bonding Mode	Non-bonded	Non-bonded
PLL/CDR Reference Clock Frequency	153.6 Mhz	153.6 Mhz

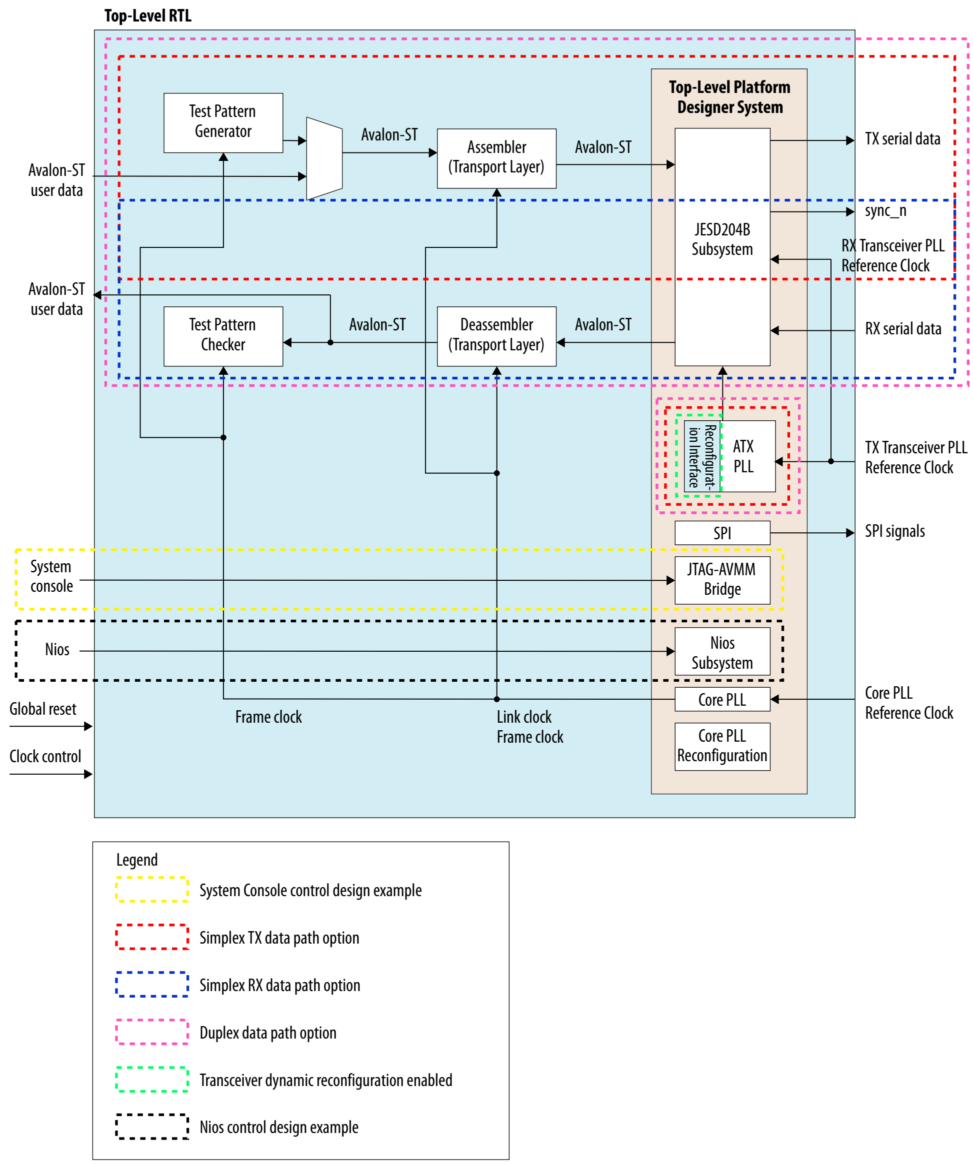
continued...

JESD204B IP Parameters	Preset 1	Preset 2
Enable Bit Reversal and Byte Reversal	No	No
Enable Transceiver Dynamic Reconfiguration	No	No
L	2	8
M	2	8
Enable manual F configuration	No	Yes
F	2	8
N	16	12
N'	16	12
S	1	5
K	16	32
Enable Scramble (SCR)	No	No
CS	0	0
CF	0	0
High Density User Data Format (HD)	0	0
Enable Error Code Correction (ECC_EN)	No	No

1.2.5 Functional Description

The design examples consist of various components. The following block diagrams show the design components and the top-level signals of the design examples.

Figure 7. JESD204B Design Example Block Diagram



1.2.5.1 Design Components

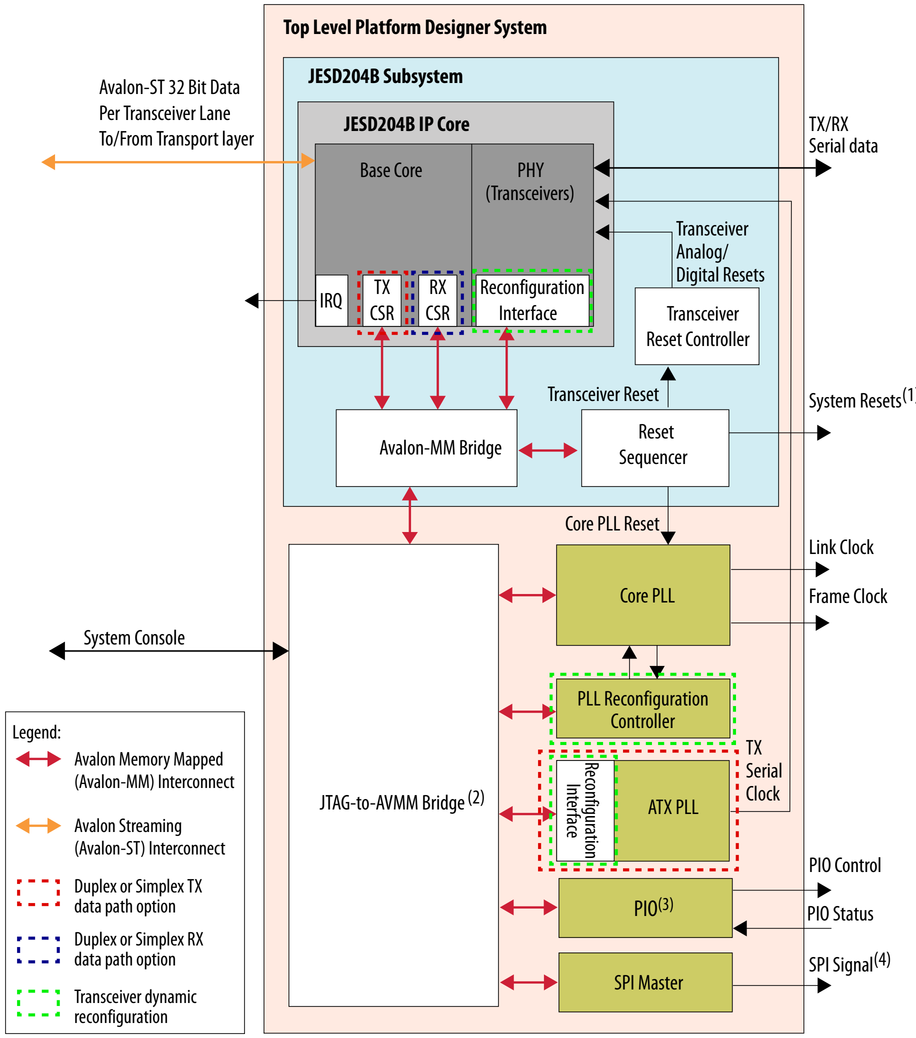
The JESD204B design example for consists of the following components:

- Platform Designer system
    - JESD204B subsystem
    - JTAG to Avalon master bridge—For System Console Control design example only.
    - Nios subsystem—For Nios Control design example only
    - Parallel I/O (PIO)
    - ATX PLL
    - Core PLL
    - PLL reconfiguration module (For transceiver dynamic reconfiguration enabled mode only)
    - Serial Port Interface (SPI)—master module
- Test pattern generator (For duplex and simplex TX data path only)
- Test pattern checker (For duplex and simplex RX data path only)
- Assembler—TX transport layer (For duplex and simplex TX data path only)
- Deassembler—RX transport layer (For duplex and simplex RX data path only)

1.2.5.1.1 Platform Designer System Component

The Platform Designer system instantiates the JESD204B IP core data path and supporting peripherals.

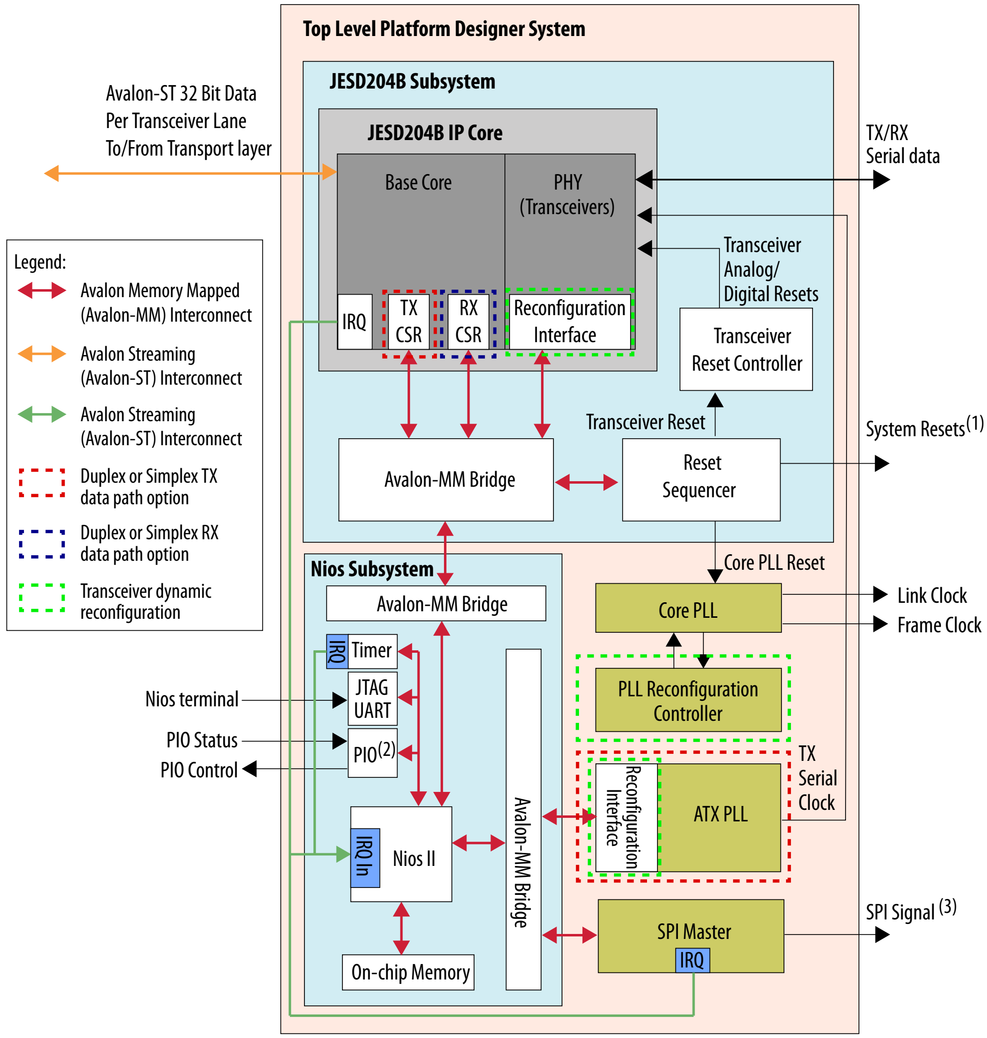
Figure 8. Platform Designer System for System Console Control Design Example



Notes:

1. System resets comprise the following resets: Core PLL reset JESD204B IP core SerDes PHY reset, TX/RX JESD204B IP core CSR resets, TX/RX link resets, TX/RX frame resets.
2. This module is replaced by Avalon-MM Bus Functional Module (BFM) in the simulation flow.
3. Parallel input/output modules. Parallel 32-bit output for control signals from JTAG to -Avalon master bridge to HDL components. Parallel 32-bit input for status signals from HDL components to JTAG to Avalon master.
4. If Generate 3-Wire SPI Module option is not selected, 4-wire SPI signal to external converter SPI interface. If Generate 3-Wire SPI Module option is selected, 3-wire SPI signal to external converter SPI interface.

Figure 9. Platform Designer System for Nios Control Design Example



Notes:

1. System resets comprise the following resets: Core PLL reset JESD204B IP core SerDes PHY reset, TX/RX JESD204B IP core CSR resets, TX/RX link resets, TX/RX frame resets.
2. Parallel input/output modules. Parallel 32-bit output for control signals from Nios subsystem to HDL components. Parallel 32-bit input for status signals from HDL components to Nios subsystem.
3. If Generate 3-Wire SPI Module option is not selected, 4-wire SPI signal to external converter SPI interface. If Generate 3-Wire SPI Module option is selected, 3-wire SPI signal to external converter SPI interface.

The top level Platform Designer system instantiates the following modules:

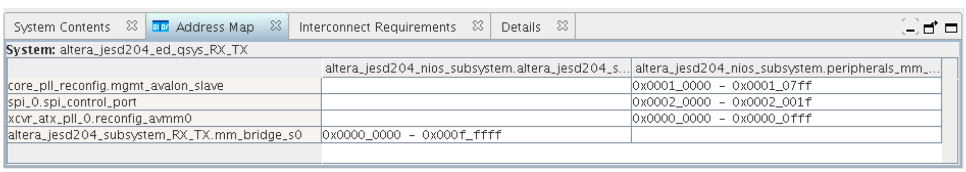
- Platform Designer system
  - JESD204B subsystem
  - JTAG to Avalon master bridge—for System Console Control design example only
  - Nios subsystem—Nios Control design example only
  - Parallel I/O (PIO)
  - ATX PLL
  - Core PLL
  - PLL reconfiguration module (For transceiver dynamic reconfiguration enabled mode only)
  - Serial Port Interface (SPI)—master module

The following are the key features of the top level Platform Designer system:

- Supports 2 design example types:
  - System Console control
  - Nios control
- Supports 3 data path types:
  - Duplex—Both TX and RX data paths present
  - Simplex TX—Only TX data path present
  - Simplex RX—Only RX data path present
- Supports transceiver dynamic reconfiguration enabled mode:
  - When enabled, connects the JTAG to Avalon master bridge (System Console control) or Nios subsystem (Nios control) module to the following interfaces:
    - Transceiver PHY reconfiguration interface
    - ATX PLL reconfiguration interface
    - Core PLL reconfiguration controller
  - When disabled, reconfiguration interfaces not present in design example
- The JESD204B subsystem, PLL reconfiguration controller, ATX PLL dynamic reconfiguration interface, parallel I/O and SPI master modules are connected to the JTAG to Avalon master bridge (System Console control) or Nios subsystem (Nios control) module via the Avalon Memory-Mapped (Avalon-MM) interface.
- JTAG to Avalon master bridge provides a link to the user via System Console. You can control the behavior of the design example via Tcl scripts executed in the System Console interface.
- Nios subsystem provides a way for the user to control the behavior of the design example using embedded C programming.
- TX data path flow:
  - Input: 32-bit per transceiver lane Avalon Streaming (Avalon-ST) input from assembler (TX transport layer)
  - Output: TX serial data

- RX data path flow:
  - Input: RX serial data from either external converter source or internal serial loopback
  - Output: 32-bit per transceiver lane Avalon Streaming (Avalon-ST) input to deassembler (RX transport layer)
- SPI master module links out to the SPI configuration interface of external converters via a 3- or 4-wire SPI interconnect (depending on Generate 3-Wire SPI Module setting).
- SPI master module handles the serial transfer of configuration data to the SPI interface on the converter end
- The ATX PLL generates the serial clock for clocking the TX serial data
  - ATX PLL module generated for duplex and simplex TX data path only
  - ATX PLL reconfiguration interface only present when transceiver dynamic reconfiguration option is enabled.
  - When present, ATX PLL reconfiguration interface connects to the JTAG to Avalon master bridge (System Console control) or Nios subsystem (Nios control) module via the Avalon Memory-Mapped (Avalon-MM) interface.
- The core PLL generates the following clocks for the system:
  - Link clock
  - Frame clock

Figure 10. Top Level Platform Designer Address Map



	altera_jesd204_nios_subsystem.altera_jesd204_s...	altera_jesd204_nios_subsystem.peripherals_mm_...
core_pll_reconfig.mgmt_avalon_slave		0x0001_0000 – 0x0001_07ff
spi_0.spi_control_port		0x0002_0000 – 0x0002_001f
xcvr_atx_pll_0.reconfig_avmm0		0x0000_0000 – 0x0000_0fff
altera_jesd204_subsystem_RX_TX.mm_bridge_s0	0x0000_0000 – 0x000f_ffff	

JESD204B Subsystem in Platform Designer

The JESD204B subsystem instantiates the following modules:

- JESD204B IP core
- Reset sequencer
- Transceiver PHY reset controller
- Avalon-MM bridge

JESD204B IP Core

The generated design example is a self-contained system with its own JESD204B IP core instantiation that is separate from the IP core that is generated from the **IP** tab. The design example JESD204B IP core inherits the IP parameter settings as set when you generate the design example. The JESD204B IP base core and PHY layer connect to System Console or Nios subsystem via the Avalon-MM interconnect. There are three separate Avalon-MM ports for the JESD204B IP core:

- Base core TX data path—For dynamic reconfiguration of the TX CSR parameters
- Base core RX data path—For dynamic reconfiguration of the RX CSR parameters
- PHY layer—For dynamic reconfiguration of transceiver PHY CSR

You can dynamically change the configuration of the JESD204B IP core base and PHY layers via TCL scripts using the System Console or via embedded C programming using Nios subsystem.

The following JESD204B IP core parameter selections affect the structure of the design example:

- Data path:
  - Duplex—Both TX and RX data paths and CSR interfaces present
  - TX only—Only TX data path and CSR interface present
  - RX only—Only RX data path and CSR interface present
- Transceiver dynamic reconfiguration mode:
  - When enabled, transceiver PHY reconfiguration interface is present in the design example and connected the JTAG to Avalon master bridge (System Console control) or Nios subsystem (Nios control) module.
  - When disabled, transceiver PHY reconfiguration interface not present in design example.

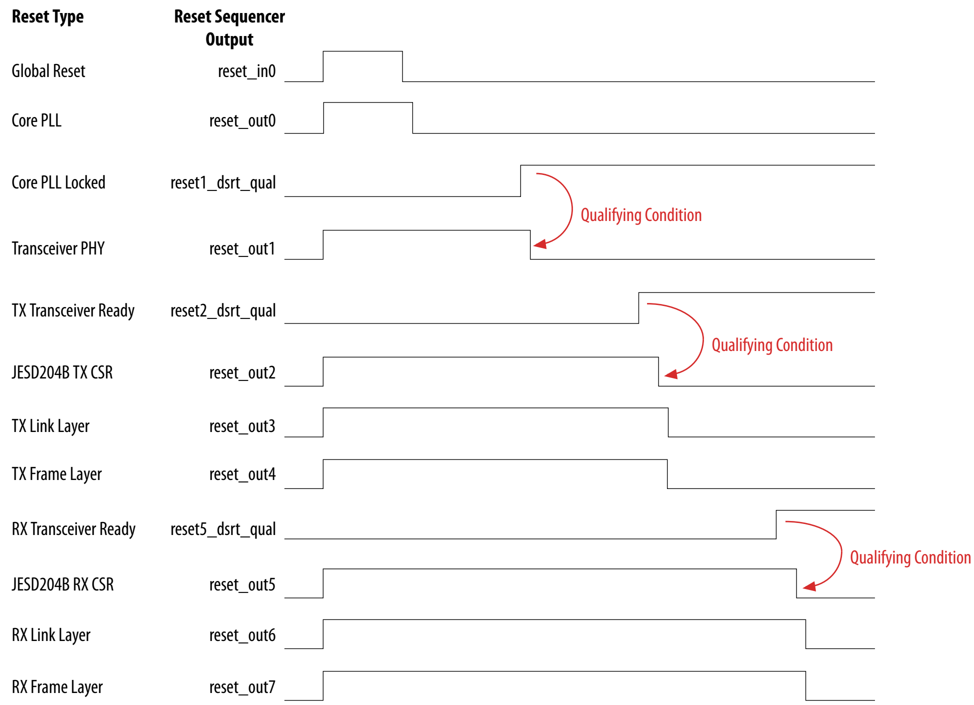
Reset Sequencer

The reset sequencer is a standard Platform Designer component in the **IP Catalog** standard library. The reset sequencer generates the following system resets to reset various modules in the system:

1. Core PLL reset—resets the core PLL
2. Transceiver reset—resets the JESD204B IP core PHY module
3. TX/RX JESD204B IP core CSR reset—resets the TX/RX JESD204B IP core CSRs
4. TX/RX link reset—resets the TX/RX JESD204B IP core base module and transport layer
5. TX/RX frame reset—resets the TX/RX transport layer, upstream and downstream modules

The reset sequencer has hard and soft reset options. The hard reset port connects to the global reset input pin in the top level design. The soft reset is activated via Avalon-MM interface either by TCL scripts (System Console control) or embedded C programming (Nios control). When you assert a hard or soft reset, the reset sequencer cycles through all the various module resets based on a pre-set sequence. The figure below illustrates the sequence and also shows how the reset sequencer output ports correspond to the modules that are being reset.

Figure 11. Reset Sequence



Transceiver PHY Reset Controller

The transceiver PHY reset controller is a standard Platform Designer component in the **IP Catalog** standard library. This module takes the transceiver PHY reset output from the reset sequencer and generates the proper analog and digital reset sequencing for the transceiver PHY module.

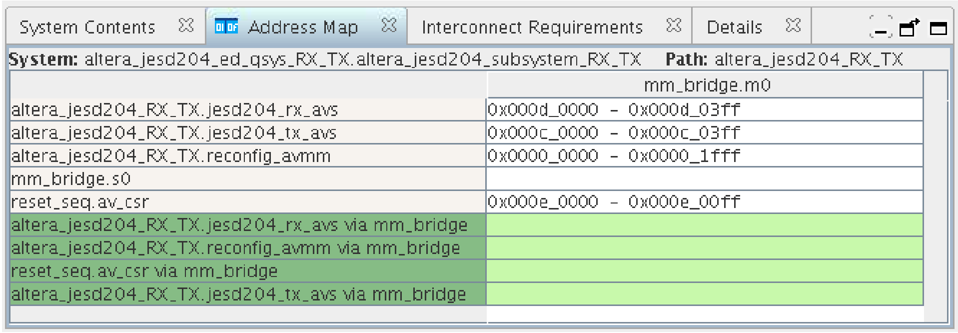
Avalon-MM Bridge

All the Avalon-MM submodules in the JESD204B subsystem are connected via Avalon-MM interconnect to a single Avalon-MM bridge. This bridge is the single interface for Avalon-MM communications into and out of the subsystem.

JESD204B Subsystem Address Map

Access the address map of the submodules in the JESD204B subsystem by clicking on the **Address Map** tab in the Platform Designer window.

Figure 12. JESD204B Subsystem Address Map



JTAG to Avalon Master Bridge

Note: This module is only available in the System Console Control design example.

The JTAG to Avalon master bridge is a standard Platform Designer component in the IP Catalog standard library. This module provides a connection between a host system and the Platform Designer system via the respective physical interfaces; JTAG on the host system end and Avalon-MM on the Platform Designer system end. Host systems can initiate Avalon Memory-Mapped (Avalon-MM) transactions by sending encoded streams of bytes via JTAG interface. The module supports reads and writes, but not burst transactions.

Related Links

Platform Designer System Component on page 21

Parallel I/O

Note: This module is instantiated in the top level Platform Designer system in the System Console Control design example. This module is instantiated in the Nios subsystem in the Nios Control design example.

Parallel I/O (PIO) modules provide general input/output (I/O) access from the Avalon master (JTAG to Avalon master bridge for System Console control or Nios subsystem for Nios control). There are two sets of 32-bit PIO registers:

- Status registers—input from the HDL components to the Avalon master
- Control registers—output from the Avalon master to the HDL components

The registers are assigned in the top level HDL file (`io_status` for status registers, `io_control` for control registers). The tables below describe the signal connectivity for the status and control registers.

Table 10. Signal Connectivity for Status Registers

Bit	Signal
0	Core PLL locked
1	TX transceiver ready (for duplex and simplex TX data path only)
2	RX transceiver ready (for duplex and simplex RX data path only)

continued...

Bit	Signal
3	Test pattern checker data error (for duplex and simplex RX data path only)
4	TX link error (for duplex and simplex TX data path only)
5	RX link error (for duplex and simplex RX data path only)

Table 11. Signal Connectivity for Control Registers

Bit	Signal
0	RX serial loopback enable (for duplex data path only)
30	Global reset
31	Sysref

ATX PLL

Note: This module is only available in the design example when the duplex or simplex TX data path option is selected.

The ATX PLL is a standard Platform Designer component in the **IP Catalog** standard library. This module supplies a low-jitter serial clock to the transceiver PHY module. The reference clock input to the ATX PLL comes from an external source. If the transceiver dynamic reconfiguration option is selected during design example generation, the ATX PLL has an Avalon-MM interface that connects to the Avalon master (JTAG to Avalon master bridge for System Console control or Nios subsystem for Nios control) via the Avalon-MM interconnect and can receive configuration instructions from the Avalon master.

For simplex TX variant, the frequency selection in the **PLL/CDR Reference Clock Frequency** drop-down list in the JESD204B IP parameter editor is disabled. The design example generates the ATX PLL with the reference clock frequency of either:

- Hard PCS: `data_rate/20`
- Soft PCS: `data_rate/40`

Refer to *Changing the Data Rate or Reference Clock Frequency* section for more information about modifying the ATX PLL reference clock frequency to suit your application.

For duplex variant, the ATX PLL reference clock frequency shares the frequency with the CDR reference clock. You must select the frequency from the **PLL/CDR Reference Clock Frequency** drop-down list in the IP parameter editor.

For the ATX PLL reference clock frequencies supported range, refer to the Intel Arria 10 Device Datasheet.

Core PLL

The core PLL is a clock generation module that generates the clocks for the FPGA core fabric. An IOPLL module is instantiated as core PLL.

The core PLL uses an external clock input as its reference clock to generate two derivative clocks from a single VCO:

- Link clock
- Frame clock

Table 12. Core PLL Ouputs

Clock	Formula	Description
Link Clock	Serial data rate/40	The link clock clocks the JESD204B IP core link layer and the link interface of the transport layer.
Frame Clock	Serial data rate/(10 × F)	The frame clock clocks the transport layer, test pattern generators and checkers, and any downstream modules in the FPGA core fabric.

For the frame clock, when JESD204B IP core parameter F=1 or F=2, the resulting frame clock frequency can easily exceed the capability of the core PLL to generate and close timing. The top level RTL file, (`altera_jesd204_ed_<data path>.sv`), defines the frame clock division factor parameters, F1_FRAMECLK_DIV (for cases with F = 1) and F2_FRAMECLK_DIV (for cases with F =2). This factor enables the transport layer and test pattern generator to operate at a divided factor of the required frame clock rate by widening the data width accordingly.

For this design example, the F1_FRAMECLK_DIV is set to 4 and F2_FRAMECLK_DIV is set to 2. As an example, the actual frame clock for a serial data rate of 10 Gbps and F =1 is:

(10000/(10 × 1)) / F1_FRAMECLK_DIV = 1000 / 4 = 250 MHz

Frame Clock and Link Clock Relationship

The frame clock and link clock are synchronous. For the derived F mode, the ratio of `link_clk` period to `frame_clk` period is given by this formula:

```
link_clk period to frame_clk period ratio = 32xL/(MxSxN')
```

Table 13. $f_{TXframe}$ and $_{RXframe}$ for Different F Parameter Settings

- f_{TXlink} is the TX link clock frequency
- f_{RXlink} is the RX link clock frequency

F Parameter	$f_{TXframe}$(txframe_clk frequency)	$f_{RXframe}$(rxframe_clk frequency)
1	f_{TXlink}x(4/F1_FRAMECLK_DIV)	f_{RXlink}x(4/F1_FRAMECLK_DIV)
2	f_{TXlink}x(2/F2_FRAMECLK_DIV)	f_{RXlink}x(2/F2_FRAMECLK_DIV)
4	f_{TXlink}	f_{RXlink}
8	f_{TXlink}/2	f_{RXlink}/2

SPI Master

The SPI master module is a standard Platform Designer component in the **IP Catalog** standard library. This module uses the SPI protocol to facilitate the configuration of external converters (for example, ADC, DAC, external clock modules) via a structured register space inside the converter device. The SPI master has an Avalon-MM interface that connects to the Avalon master (JTAG to Avalon master bridge for System Console control or Nios subsystem for Nios control) via the Avalon-MM interconnect and can receive configuration instructions from the Avalon master.

This module is configured to a 4-wire, 24-bit width interface. If the **Generate 3-Wire SPI Module** option is selected, an additional module is instantiated to convert the 4-wire output of the SPI master to 3-wire.

For more details on the SPI master module, refer to the *JESD204B IP Core User Guide*.

Related Links

JESD204B IP Core User Guide

Nios Subsystem

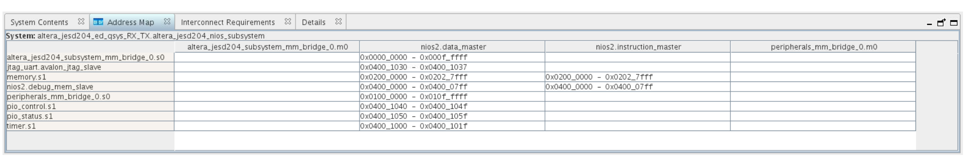
Note: This module is only available in the Nios Control design example

The Nios subsystem enables an embedded software-based control flow for the design example. Using the Nios control flow, you can develop and compile embedded C code to control the behavior of the design example. The Nios subsystem is a Platform Designer system that instantiates the following peripherals:

- Nios II processor
- On-chip memory—provides both instruction and data memory space
- Timer—provides a general timer function for the software
- JTAG UART—serves as the main communications portal between the user and the Nios II processor via the terminal console in **Nios II Software Build Tools** for Eclipse tool
- Avalon-MM bridges—two Avalon-MM bridge modules;
  - — To interface to the JESD204B subsystem
  - — To interface to Platform Designer components (core PLL reconfiguration controller, ATX PLL dynamic reconfiguration interface and SPI master module) in the top level Platform Designer project.
- Parallel I/O (PIO)—provides general input/output (I/O) access from the Nios II processor to the HDL components in the FPGA. Refer to the *Parallel I/O* section for more details.

Nios Subsystem Address Map

Figure 13. Nios Subsystem Address Map


System Contents | Address Map | Interconnect Requirements | Details

System: altera_jesd204_ed_qsys_RX_TX.altera_jesd204_nios_subsystem

	altera_jesd204_subsystem_mm_bridge_0.m0	nios2.data_master	nios2.instruction_master	peripherals_mm_bridge_0.m0
altera_jesd204_subsystem_mm_bridge_0.s0		0x0000_0000 - 0x000f_ffff		
jtag_uart.avalon_jtag_slave		0x0400_1030 - 0x0400_1037		
memory.s1		0x0200_0000 - 0x0202_7fff	0x0200_0000 - 0x0202_7fff	
nios2.debug_mem_slave		0x0400_0000 - 0x0400_07ff	0x0400_0000 - 0x0400_07ff	
peripherals_mm_bridge_0.s0		0x0100_0000 - 0x010f_ffff		
pio_control.s1		0x0400_1040 - 0x0400_104f		
pio_status.s1		0x0400_1050 - 0x0400_105f		
timer.s1		0x0400_1000 - 0x0400_101f		

1.2.5.1.2 Transport Layer

The transport layer in the design example consists of an assembler at the TX path and a deassembler at the RX path. The transport layer for both the TX and RX path is instantiated in the top level RTL file, not in the Platform Designer project.

Note: When the simplex TX data path option is selected, only the assembler is instantiated in the design example. When the simplex RX data path option is selected, only the deassembler is instantiated in the design example. When the duplex data path option is selected, both assembler and deassembler is instantiated in the design example.

The transport layer provides the following services to the application layer (AL) and the data link layer (DLL):

- Assembler at the TX path:
  - Maps the conversion samples from the AL (through the Avalon-ST interface) to a specific format of non-scrambled octets, before streaming them to the DLL.
  - Reports AL error to the DLL if it encounters a specific error condition on the Avalon-ST interface during TX data streaming.
- Deassembler at the RX path:
  - Maps the descrambled octets from the DLL to a specific conversion sample format before streaming them to the AL (through the Avalon-ST interface).
  - Reports AL error to the DLL if it encounters a specific error condition on the Avalon-ST interface during RX data streaming.

The transport layer has many customization options and you can modify the transport layer RTL to customize it to your specifications. Furthermore, for certain parameters like L, F, and N, the transport layer shares the CSR values with the JESD204B IP core.

For more details on the implementation of the transport layer in RTL and customization options, refer to the *JESD204B IP Core User Guide*.

Related Links

JESD204B IP Core User Guide

1.2.5.1.3 Test Pattern Generator

Note: This module is only available in the design example when the duplex or simplex TX data path option is selected.

The test pattern generator generates either a parallel PRBS, alternate checkerboard, or ramp wave and sends it to the transport layer during test mode. The test pattern generator is implemented in the top level RTL file, not in the Platform Designer project.

The test pattern generator has many customization options and you can modify the test pattern generator RTL to customize it to your specifications. Furthermore, for parameters like M, S, N, and test mode, the test pattern generator shares the CSR values with the JESD204B IP core. This means that any dynamic reconfiguration operation that affects those values for the JESD204B IP core, affects the test pattern generator in the same way. This includes the pattern type (PRBS, alternate checkerboard, ramp) which is controlled by the test mode CSR.

For more details on the JESD204B IP core CSR (register map), refer to the *JESD204B IP Core User Guide*.

Related Links

JESD204B IP Core User Guide

1.2.5.1.4 Test Pattern Checker

Note: This module is only available in the design example when the duplex or simplex RX data path option is selected.

The test pattern checker checks either a parallel PRBS, alternate checkerboard, or ramp wave from the transport layer during test mode and outputs an error flag if there are any data mismatches. The test pattern checker is implemented in the top level RTL file, not in the Platform Designer project.

The test pattern checker has many customization options and you may modify the test pattern checker RTL to customize it to your specifications. Furthermore, for parameters like M, S, N, and test mode, the test pattern checker shares the CSR values with the JESD204B IP core. This means that any dynamic reconfiguration operation that affects those values for the JESD204B IP core, affects the test pattern checker in the same way. This includes the pattern type (PRBS, alternate checkerboard, ramp) which is controlled by the test mode CSR.

For more details on the JESD204B IP core CSR (register map), refer to the *JESD204B IP Core User Guide*.

Related Links

JESD204B IP Core User Guide

1.2.5.2 Clocking Scheme

The main reference clock for the design example is `device_clk`. This clock must be supplied from an external source. The `device_clk` is the reference clock for the core PLL, ATX PLL and the TX/RX transceiver PHY. The core PLL generates the `link_clk` and `frame_clk` from `device_clk`. The `link_clk` clocks the JESD204B IP core link layer and link interface of the transport layer. The `frame_clk` clocks the transport layer, test pattern generator and checker modules, and any downstream modules. An external source supplies a clock called the `mgmt_clk` to clock the Avalon-MM interfaces of Platform Designer components.

Table 14. System Clocking for the Design Example

Note: The IOPLL input reference clock is sourcing from device clock through the global clock network. Sourcing reference clock from a cascaded PLL output, global clock or core clock network might introduce additional jitter to the IOPLL and transceiver PLL output. Refer to this KDB Answer for a workaround you should apply to the IP core in your design.

Clock	Description	Source	Modules Clocked
`device_clk`	Reference clock for the core PLL, ATX PLL and RX transceiver PHY	External	Core PLL, ATX PLL, RX transceiver PHY
`link_clk`	Link layer clock	`device_clk`	JESD204B IP core link layer, transport layer link interface
`frame_clk`	Frame layer clock	`device_clk`	Transport layer, test pattern generator and checker, downstream modules
`mgmt_clk`	Control plane clock	External	Avalon-MM interfaces

1.2.6 Simulation

Note: The simulation flow is only supported for System Console Control design example only. The simulation flow is not supported for Nios Control design example.

Execute the simulation by running the relevant simulation run scripts in the supported simulator environment. The following table shows the simulators supported along with the relevant run scripts.

Table 15. Supported Simulators

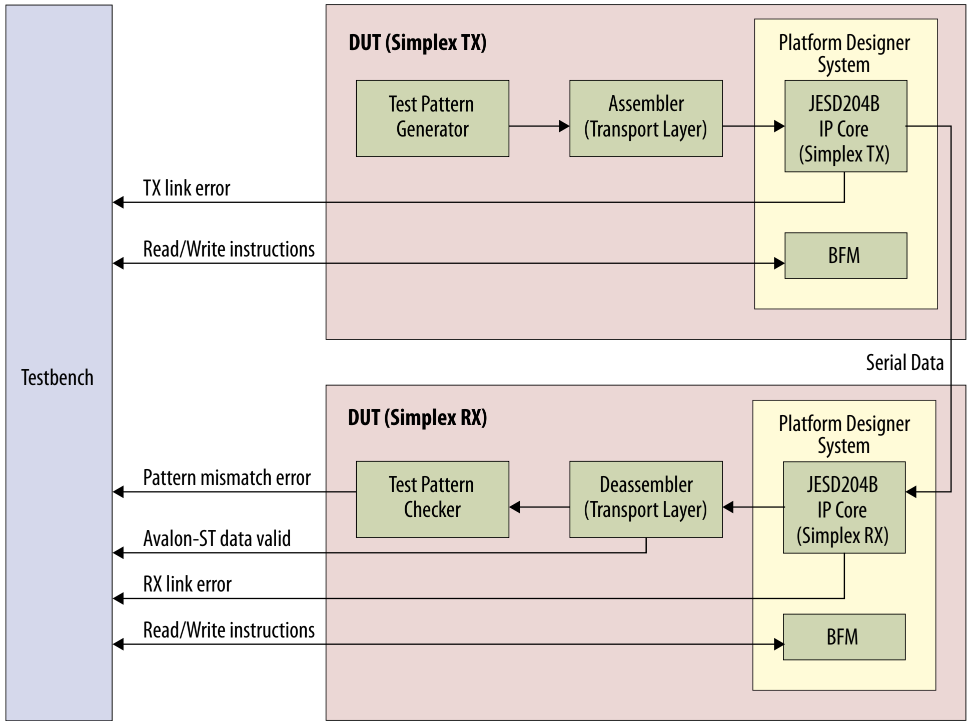
Simulators	Simulation Directory	Run Script
Riviera	`/testbench/aldec/`	`run_tb_top.tcl`
NCSim	`/testbench/cadence/`	`run_tb_top.sh`
Modelsim	`/testbench/mentor/`	`run_tb_top.tcl`
VCS	`/testbench/synopsys/vcs/`	`run_tb_top.sh`
VCSMX	`/testbench/synopsys/vcsmx/`	`run_tb_top.sh`

The design generates the simulation results which include the transcript or log files in the relevant simulation directory.

1.2.6.1 Testbench

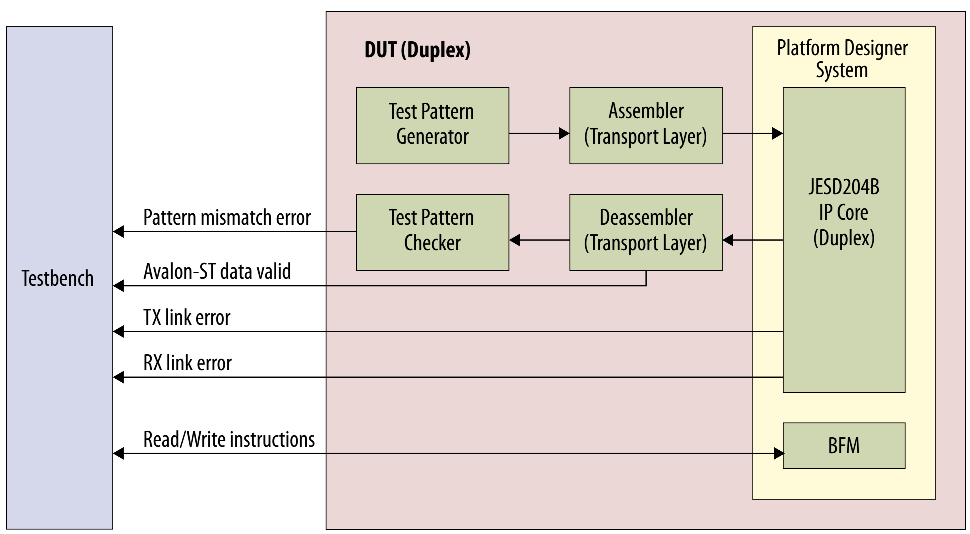
The simulation design-under-test (DUT) is the generated design example which includes a synthesizable pattern generator and checker. The figures below show the testbench block diagram for simplex and duplex options.

Figure 14. Simulation Testbench Block Diagram (Simplex TX or RX)



Note: Both simplex TX and simplex RX design examples generate the same testbench. The testbench instantiates two DUTs: one simplex TX DUT, one simplex RX DUT. The TX serial data output of the simplex TX DUT is connected to the RX serial data input of the simplex RX DUT. The testbench issues separate Avalon-MM read/write instructions to the simplex TX and simplex RX DUTs respectively.

Figure 15. Simulation Testbench Block Diagram (Duplex)



The simulation flow replaces the JTAG to Avalon master bridge module in the Platform Designer system of the System Console Control design example with the Avalon-MM master bus functional model (BFM). This BFM enables a testbench to send Avalon-MM read/write commands to the design example registers to mimic the functionality of System Console.

The testbench provided in the simulation flow (`/testbench/models/tb_top.sv`) executes the following steps:

- Reset DUT
- Initialize BFM
- Execute Avalon-MM commands to initialize the DUT in the following mode:
  - Internal serial loopback mode (for duplex option only)
  - Pattern generator/checker set to PRBS pattern
- Wait for DUT to initialize to user mode
- Report JESD204B link status

When simulation ends, the following messages are shown at end.

Table 16. Simulation Messages and Description

Message	Description
`Pattern Checker(s): Data error(s) found!`	Pattern mismatch errors found on the pattern checker
`Pattern Checker(s): OK!`	No errors found on the pattern checker
`Pattern Checker(s): No valid data found!`	No valid data received by pattern checker
`JESD204B Tx Core(s): Tx link error(s) found!`	Link errors reported by JESD204B IP TX
`JESD204B Tx Core(s): OK!`	No link errors reported by JESD204B IP TX
`JESD204B Rx Core(s): Rx link error(s) found!`	Link errors reported by JESD204B IP RX
`JESD204B Rx Core(s): OK!`	No link errors reported by JESD204B IP RX
`TESTBENCH_PASSED: SIM PASSED!`	Overall simulation passed
`TESTBENCH_FAILED: SIM FAILED!`	Overall simulation failed

1.2.7 Design Example Files

There are two flows for the design example: simulation and synthesis.

Table 17. Design Example Flows and Directory

Design Example Flow	Directory
Simulation	`<your project>/ed_sim`
Synthesis	`<your project>/ed_synth`

The following tables list the important folders and files for simulation and synthesis.

Table 18. Design Example Files for Simulation

Note: The simulation flow is only supported for System Console Control design example only. The simulation flow is not supported for Nios Control design example.

File Type	File/Folder	Description
Run script files	/testbench/aldec/run_tb_top.tcl	TCL run script for Riviera simulator
	/testbench/cadence/run_tb_top.sh	Shell run script for NCSim simulator
	/testbench/mentor/run_tb_top.tcl	TCL run script for Modelsim simulator
	/testbench/synopsys/vcs/run_tb_top.sh	Shell run script for VCS simulator
	/testbench/synopsys/vcsmx/run_tb_top.sh	Shell run script for VCSMX simulator
Source files	/testbench/models/altera_jesd204_ed_qsys_<data path>.qsys	Top level Platform Designer system project
	/testbench/models/altera_jesd204_subsystem_<data path>.qsys	JESD204B subsystem
	/testbench/models/ip/	IP folder containing instantiated IP modules
	/testbench/models/altera_jesd204_ed_<data path>.sv	Top level HDL
	/testbench/models/tb_top.sv	Top level testbench
	/testbench/spi_mosi_oe.v	Output buffer HDL
	/testbench/switch_debouncer.v	Switch debouncer HDL
	/testbench/pattern/	Folder containing the test pattern generator and checker HDL
	/testbench/transport_layer	Folder containing assembler and de-assembler HDL.

Table 19. Design Example Files for Synthesis

File Type	File/Folder	Description
Intel Quartus Prime project files	altera_jesd204_ed_<data path>.qpf	Intel Quartus Prime project file
	altera_jesd204_ed_<data path>.qsf	Intel Quartus Prime settings file
Source files	altera_jesd204_ed_<data path>.sv	Top level HDL
	altera_jesd204_ed_<data path>.sdc	Synopsys* Design Constraints (SDC) file containing all timing/placement constraints
	transport_layer/	Folder containing assembler and de-assembler HDL
	pattern/	Folder containing the test pattern generator and checker HDL
	spi_mosi_oe.v	Output buffer HDL
	switch_debouncer.v	Switch debouncer HDL
	altera_jesd204_ed_qsys_<data path>.qsys	Top level Platform Designer system project
	altera_jesd204_subsystem_<data path>.qsys	JESD204B subsystem

1.2.8 Register

Refer to the *Intel Stratix® 10 JESD204B RX Address Map and Register Definitions* and *Intel Stratix 10 JESD204B TX Address Map and Register Definitions* for the list of registers.

Related Links

- Intel FPGA JESD204B RX Address Map and Register Definitions
- Intel FPGA JESD204B TX Address Map and Register Definitions

1.2.9 Signals

Table 20. System Interface Signals

Signal	Clock Domain	Direction	Description
Clocks and Resets			
device_clk	—	Input	Reference clock for design example data path.
mgmt_clk	—	Input	Reference clock for all peripherals connected via Avalon-MM interconnect.
global_rst_n	mgmt_clk	Input	Global reset signal from the push button. This reset is an active low signal and the deassertion of this signal is synchronous to the rising-edge of mgmt_clk.

Signal	Clock Domain	Direction	Description
Serial Data			
rx_serial_data[LINK*L-1:0]	device_clk	Input	Differential high speed serial input data. The clock is recovered from the serial data stream.
tx_serial_data[LINK*L-1:0]	device_clk	Output	Differential high speed serial output data. The clock is embedded in the serial data stream.

Signal	Clock Domain	Direction	Description
JESD204B			
sysref_out	mgmt_clk	Output	*SYSREF* signal for JESD204B Subclass 1 implementation.
sync_n_out	link_clk	Output	Indicates a SYNC_N from the receiver. This is an active low signal and is asserted 0 to indicate a synchronization request or error reporting.
tx_link_error	link_clk	Output	Error interrupt from JESD204B IP core indicating TX link error
rx_link_error	link_clk	Output	Error interrupt from JESD204B IP core indicating RX link error

Signal	Clock Domain	Direction	Description
Avalon- ST User Data			
avst_usr_din[LINK*TL_DATA_BUS_WIDTH-1:0]	frame_clk	Input	TX data from the Avalon-ST source interface. The TL_DATA_BUS_WIDTH is determined by the following formulas:

continued...

Signal	Clock Domain	Direction	Description
			• If F = 1, TL_DATA_BUS_WIDTH = F1_FRAMECLK_DIV*8*1*L*N/N_PRIME • If F = 2, TL_DATA_BUS_WIDTH = F2_FRAMECLK_DIV*8*2*L*N/N_PRIME • If F = 4, TL_DATA_BUS_WIDTH = 8*4*L*N/N_PRIME • If F = 8, TL_DATA_BUS_WIDTH = 8*8*L*N/N_PRIME
avst_usr_din_valid[LINK-1:0]	frame_clk	Input	Indicates whether the data from the Avalon-ST source interface to the transport layer is valid or invalid. • 0—data is invalid • 1—data is valid
avst_usr_din_ready[LINK-1:0]	frame_clk	Output	Indicates that the transport layer is ready to accept data from the Avalon-ST source interface. • 0—transport layer is not ready to receive data • 1—transport layer is ready to receive data
avst_usr_dout[LINK*TL_DATA_BUS_WIDTH-1:0]	frame_clk	Output	RX data to the Avalon-ST sink interface. The TL_DATA_BUS_WIDTH is determined by the following formulas: • If F = 1, TL_DATA_BUS_WIDTH = F1_FRAMECLK_DIV*8*1*L*N/N_PRIME • If F = 2, TL_DATA_BUS_WIDTH = F2_FRAMECLK_DIV*8*2*L*N/N_PRIME • If F = 4, TL_DATA_BUS_WIDTH = 8*4*L*N/N_PRIME • If F = 8, TL_DATA_BUS_WIDTH = 8*8*L*N/N_PRIME
avst_usr_dout_valid[LINK-1:0]	frame_clk	Output	Indicates whether the data from the transport layer to the Avalon-ST sink interface is valid or invalid. • 0—data is invalid • 1—data is valid
avst_usr_dout_ready[LINK-1:0]	frame_clk	Input	Indicates that the Avalon-ST sink interface is ready to accept data from the transport layer. • 0—Avalon-ST sink interface is not ready to receive data • 1—Avalon-ST sink interface is ready to receive data
avst_patchk_data_error [LINK-1:0]	frame_clk	Output	Output signal from pattern checker indicating a pattern check error.

Signal	Clock Domain	Direction	Description
SPI			
spi_MISO (7)	spi_SCLK	Input	Input data from external slave to the master.
spi_MOSI(7)	spi_SCLK	Output	Output data from the master to the external slaves.
			continued...

(7) When **Generate 3-Wire SPI Module** option is not enabled.

Signal	Clock Domain	Direction	Description
spi_SDIO[8]	spi_SCLK	Input/ Output	Output data from the master to external slave. Input data from external slave to master
spi_SCLK	mgmt_clk	Output	Clock driven by the master to slaves, to synchronize the data bits.
spi_SS_n[2:0]	spi_SCLK	Output	Active low select signal driven by the master to individual slaves, to select the target slave. Defaults to 3 bits.

1.2.10 Software Control Flow

Note: The software control flow is only supported by the Nios Control design example.

The key feature of the Nios Control design example is the ability to control the behavior of the JESD204B system using a C-based, software control flow.

The software control flow allows you to perform the following tasks:

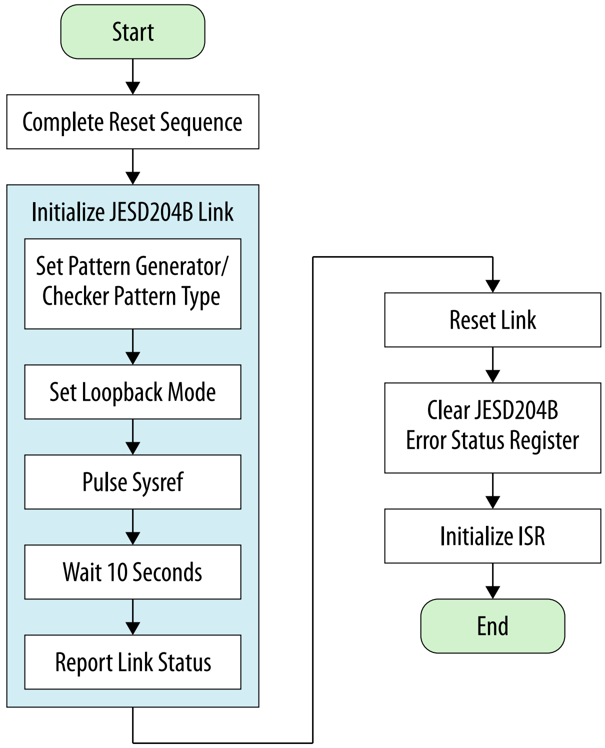
- System reset—ability to reset individual modules (core PLL, transceiver PHY, JESD204B base Avalon-MM interface, link clock domain, and frame clock domain) independently or in sequence.
- Initial and dynamic, real-time configuration of external converter devices via SPI interface.
- Dynamic reconfiguration of key modules in the design example subsystem (for example, JESD204B IP core base layer, transceiver PHY, core PLL).
- Error handling via interrupt service routines (ISR).
- Status register readback.
- Dynamic switching between real-time operation and test mode.

The software C code included as part of the design example only performs basic JESD204B link initialization. You can modify the code to perform some or all of the tasks above as per your system specifications. The software C code (`main.c`) executes a sequence of tasks as shown in the figure below.

Note: The software C code assumes that the Nios Control design is configured in duplex mode. Make your own modifications if using simplex mode design.

(8) When **Generate 3-Wire SPI Module** option enabled.

Figure 16. Software C Code Task Sequence



The JESD204B link initialization sequence accomplishes the following tasks:

- Set the pattern type for the pattern generator and checker. The default pattern type is set to PRBS.
- Set the loopback mode. The default is internal serial loopback mode.
- Pulse SYSREF (required to meet Subclass 1 requirements)
- Wait 10 seconds to allow for changes to take effect.
- Report the link status.

1.2.10.1 Software Parameters

The software parameters defined in the main header file (`main.h`) control various behaviors of the C code.

Table 21. Software Parameters

Parameter	Default Value	Description
DEBUG_MODE	0	Set to 1 to print debug messages, else set to 0.
PRINT_INTERRUPT_MESSAGES	1	Set to 1 to print JESD204B error interrupt messages, else set to 0.
PATCHK_EN	1	Set to 1 when test pattern checker is included in the initial design data path configuration, else set to 0.
DATAPATH	3	Set to indicate the JESD204B IP configuration: 1 – TX data path only.

continued...

Parameter	Default Value	Description
		2 – RX data path only. 3 – Duplex data path (TX and RX data path).
MAX_LINKS	1	Set to indicate the number of links in the design (for example, for dual link, set MAX_LINKS=2). See Implementing a Multi-Link Design section for more detailed instructions on implementing multi-link use case. *Note:* When using the design as-is, the maximum value of MAX_LINKS is 16. To increase the limit, redesign the address map in Platform Designer.
LOOPBACK_INIT	1	Initial value of the loopback. Set to 1 for internal serial loopback mode, else set to 0.
SOURCEDEST_INIT	PRBS	Initial value of source/destination. Set to indicate test pattern generator or checker type or user mode: USER – User mode (no test pattern generator or checker in data path). ALT – Test pattern generator or checker set in alternate checkerboard mode. RAMP – Test pattern generator or checker set in ramp wave mode. PRBS – Test pattern generator or checker set in parallel PRBS mode.

1.2.10.2 Interrupt Service Routines (ISR)

One key feature of the Nios Control design example is the ability to handle interrupt requests (IRQ) from peripherals through the software interrupt service routines (ISR).

In this design example, the following peripherals have their IRQ output ports connected to the IRQ input port of the Nios processor:

- JESD204B IP core TX base layer
- JESD204B IP core RX base layer
- SPI master
- Timer
- Reset sequencer

The software C code included as part of the design example defines the ISRs for the following peripherals:

- JESD204B IP core TX base layer
- JESD204B IP core RX base layer
- SPI master

The ISRs in the C code is a basic routine that performs two tasks:

- Clear IRQ error flag
- Print error type and message (for JESD204B IP core TX and RX base layer ISR only)

Error types and messages printed by the JESD204B IP core TX base layer ISR:

- SYNC_N error
- SYSREF LMFC error
- DLL data invalid error
- Transport layer data invalid error

- SYNC_N link reinitialization request
- Transceiver PLL locked error
- Phase compensation FIFO full error
- Phase compensation FIFO empty error
- Error types and messages printed by the JESD204B IP core RX base layer ISR:
- SYSREF LMFC error
- DLL data ready error
- Transport layer data ready error
- Lane deskew error
- RX locked to data error
- Phase compensation FIFO full error
- Phase compensation FIFO empty error
- Code group synchronization error
- Frame alignment error
- Lane alignment error
- Unexpected K character
- Not in table error
- Running disparity error
- Initial Lane Alignment Sequence (ILAS) error
- DLL error reserve status
- ECC error corrected
- ECC error fatal

The error types correspond to the `tx_err`, `rx_err0`, and `rx_err1` status registers in the JESD204B IP core TX and RX register maps respectively. Refer to the *JESD204B RX Address Map and Register Definitions* and *JESD204B TX Address Map and Register Definitions* for more details on the TX and RX error registers. The PRINT_INTERRUPT_MESSAGES parameter in the `main.h` header file controls the printing of interrupt error messages to the standard output. Set the parameter to 1 (default) to print error messages, else set to 0. Refer to Software Parameters on page 41 for more details. You can modify the ISRs in the C code to customize the interrupt handling response based on your system specifications.

1.2.10.3 Software Functions Description

The software C code generated with the design example performs basic JESD204B link initialization and exits. This section describes the functions used in the `main.c` code and also the macros library that facilitates access to the configuration and status registers (CSR) of the JESD204B design example system. These functions and macros provide the building blocks for you to customize the software code to your system specifications.

1.2.10.3.1 Functions in main.c Source File

The function prototypes of the functions listed in the table below can be found in the `functions.h` header file located in the software folder.

Table 22. Functions in main.c

Function Prototype	Description
int StringIsNumeric (char *string)	Tests whether the string is numeric. Returns 1 if true, 0 if false.
void DelayCounter(alt_u32 count)	Delay counter. Counts up to count ticks, each tick is roughly 1 second.
int Status (char *options[][])	Executes report link status command according to the options. Returns 0 if success, 1 if fail, 2 if sync errors found, 4 if pattern checker errors found, 6 if both sync errors and pattern checker errors found
int Loopback (char *options[][], int *held_resets, int dnr)	Executes loopback command according to the options. Returns 0 if success, 1 if fail
int SourceDest (char *options[][], int *held_resets, int dnr)	Executes source or destination datapath selection command according to the options. Returns 0 if success, 1 if fail
int Test (char *options[][], int *held_resets)	Executes test mode command according to the options. Test mode: • Set source/destination datapath selection to PRBS test pattern generator or checker. • Set transceiver to serial loopback mode. Returns 0 if success, 1 if fail.
void Sysref (void)	Pulse SYSREF signal one time (one-shot)
void ResetHard (void)	Triggers full hardware reset sequence through the PIO control registers.
int ResetSeq (int link, int *held)	Performs full hardware reset sequence through the software interface on the indicated link. Returns 0 if success, 1 if fail.
int ResetForce (int link, int reset_val, int hold_release, int *held_resets)	Forces reset assertion or deassertion on submodule resets indicated by `reset_val` for the indicated link. The function also decides whether to assert and hold (hold_release=2), deassert (hold_release=1), or pulse (hold_release=0) the indicated resets. The function has mechanisms using the `global held_resets` flag to ensure that held resets that are not the target of the reset force function are not affected by it. Returns 0 if success, 1 if fail.
int Reset_X_L_F_Release (int link, int *held_resets)	Deassert the transceiver, link, and frame resets. The function deasserts the TX transceiver reset first, waits until the TX transceiver ready signal asserts, then deasserts the TX link and TX frame resets. The function then repeats the above actions for the RX side. Returns 0 if success, 1 if fail.
void InitISR (void)	Initializes the interrupt controllers for the following peripherals: • JESD204B IP core TX CSR • JESD204B IP core RX CSR • SPI Master The timer and JTAG UART interrupt controllers are disabled. Modify the function to enable it. Refer to the Nios II Software Developer's Handbook for more details on writing ISRs.

continued...

Function Prototype	Description
static void ISR_JESD_RX (void * context)	JESD204B IP core RX ISR. Upon an interrupt event (IRQ asserted), the function reads the RX JESD204B CSR `rx_err0` and `rx_err1` registers and reports the error code. After that, the ISR clears all valid and active status registers in the `rx_err0` and `rx_err1` registers. Refer to the Nios II Software Developer's Handbook for more details on writing ISRs.
static void ISR_JESD_TX (void * context)	JESD204B IP core TX ISR. Upon an interrupt event (IRQ asserted), the function reads the TX JESD204B CSR `tx_err` registers and reports the error code. After that, the ISR clears all the valid and active status registers in the `tx_err` registers. Refer to the Nios II Software Developer's Handbook for more details on writing ISRs.
static void ISR_SPI (void * context)	SPI Master interrupt service routine (ISR). Upon interrupt event (IRQ assert), clears IRQ flag and return. Refer to the Nios II Software Developer's Handbook for more details on writing ISRs.

1.2.10.3.2 Custom Peripheral Access Macros in macros.c Source File

A set of peripheral access macros are provided for you to access specific information in the CSR of the following peripherals:

- Reset sequencer
- JESD204B TX
- JESD204B RX
- PIO control
- PIO status
- Transceiver Native PHY IP core
- ATX PLL
- Core PLL Reconfiguration

The function prototypes of the macros listed in the table below can be found in the macros.h header file located in the software folder.

Table 23. Custom Peripheral Access Macros in macros.c

Function Prototype	Description
int CALC_BASE_ADDRESS_LINK (int *base* , int *link*)	Calculates and returns the base address based on the *link* provided. In the Platform Designer system (`jesd204b_ed_qsys.qsys`) address map, bits 16-19 are reserved for multi-link addressing. The address map allocation allows for up to a maximum of 16 links to be supported using the existing address map. The number of multi-links in the design is defined by the `MAX_LINKS` parameter in the `main.h` header file. You are responsible to set the parameter correctly to reflect the system configuration.
int CALC_BASE_ADDRESS_XCVR_PLL (int *base* , int *instance*)	Calculates and returns the base address of the TX transceiver PLL (ATX PLL) based on the *instance* number. In the JESD204B subsystem (`jesd204b_subsystem.qsys`) address map, bits 12-13 are reserved for multi ATX PLL addressing. The address map allocation allows for up to a maximum of four ATX PLLs per link to be supported using the existing address map. The number of ATX PLLs per link in the design is

continued...

Function Prototype	Description
	defined by the XCVR_PLL_PER_LINK parameter in the `main.h` header file. You are responsible to set the parameter correctly to reflect the system configuration.
int IORD_RESET_SEQUENCER_STATUS_REG (int *link*)	Read reset sequencer status register at *link* and return the value.
int IORD_RESET_SEQUENCER_RESET_ACTIVE (int *link*)	Read reset sequencer status register at *link* and return 1 if the reset active signal is asserted, else return 0.
void IOWR_RESET_SEQUENCER_INIT_RESET_SEQ (int *link*)	Write reset sequencer at *link* to trigger full hardware reset sequence.
void IOWR_RESET_SEQUENCER_FORCE_RESET (int *link* , int *val*)	Write reset sequencer at *link* to force assert or deassert resets based on the *val* value.
int IORD_JESD204_TX_STATUS0_REG (int *link*)	Read the JESD204B TX CSR tx_status0 register at *link* and return the value.
int IORD_JESD204_TX_SYNCN_SYSREF_CTRL_REG (int *link*)	Read the JESD204B TX CSR `syncn_sysref_ctrl` register at *link* and return the value.
void IOWR_JESD204_TX_SYNCN_SYSREF_CTRL_REG (int *link* , int *val*)	Write *val* value into the JESD204B TX CSR `syncn_sysref_ctrl` register at link *link*.
int IORD_JESD204_TX_DLL_CTRL_REG (int *link*)	Read JESD204B TX CSR `dll_ctrl` register at *link* and return value.
void IOWR_JESD204_TX_DLL_CTRL_REG (int *link* , int *val*)	Write *val* value into the JESD204B TX CSR `dll_ctrl` register at *link*.
int IORD_JESD204_RX_STATUS0_REG (int *link*)	Read JESD204B RX CSR `rx_status0` register at *link* and return value.
int IORD_JESD204_RX_SYNCN_SYSREF_CTRL_REG (int *link*)	Read JESD204B RX CSR `syncn_sysref_ctrl` register at *link* and return value.
void IOWR_JESD204_RX_SYNCN_SYSREF_CTRL_REG (int *link*, int *val*)	Write *val* value into the JESD204B RX CSR `syncn_sysref_ctrl` register at *link*.
int IORD_JESD204_TX_ILAS_DATA1_REG (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the value.
int IORD_JESD204_RX_ILAS_DATA1_REG (int *link*)	Read the JESD204B RX CSR `ilas_data1` register at *link* and return the value.
void IOWR_JESD204_TX_ILAS_DATA1_REG (int *link*, int *val*)	Write *val* value into the JESD204B TX CSR `ilas_data1` register at *link*.
void IOWR_JESD204_RX_ILAS_DATA1_REG (int *link*, int *val*)	Write *val* value into the JESD204B RX CSR `ilas_data1` register at *link*.
int IORD_JESD204_TX_ILAS_DATA2_REG (int *link*)	Read the JESD204B TX CSR `ilas_data2` register at *link* and return the value.
int IORD_JESD204_RX_ILAS_DATA2_REG (int *link*)	Read the JESD204B RX CSR `ilas_data2` register at *link* and return the value.
void IOWR_JESD204_TX_ILAS_DATA2_REG (int *link*, int *val*)	Write *val* value into the JESD204B TX CSR `ilas_data2` register at *link*.
void IOWR_JESD204_RX_ILAS_DATA2_REG (int *link*, int *val*)	Write *val* value into the JESD204B RX CSR `ilas_data2` register at *link*.
int IORD_JESD204_TX_ILAS_DATA12_REG (int *link*)	Read the JESD204B TX CSR `ilas_data12` register at *link* and return the value.

continued...

Function Prototype	Description
int IORD_JESD204_RX_ILAS_DATA12_REG (int *link*)	Read the JESD204B RX CSR `ilas_data12` register at *link* and return the value.
void IOWR_JESD204_TX_ILAS_DATA12_REG (int *link*, int *val*)	Write *val* value into the JESD204B TX CSR `ilas_data12` register at *link*.
void IOWR_JESD204_RX_ILAS_DATA12_REG (int *link*, int *val*)	Write *val* value into the JESD204B RX CSR `ilas_data12` register at *link*.
int IORD_JESD204_TX_GET_L_VAL (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the L value.
int IORD_JESD204_RX_GET_L_VAL (int *link*)	Read the JESD204B RX CSR `ilas_data1` register at *link* and return the L value.
int IORD_JESD204_TX_GET_F_VAL (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the F value.
int IORD_JESD204_RX_GET_F_VAL (int *link*)	Read the JESD204B RX CSR `ilas_data1` register at *link* and return the F value.
int IORD_JESD204_TX_GET_K_VAL (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the K value.
int IORD_JESD204_RX_GET_K_VAL (int *link*)	Read JESD204B RX CSR `ilas_data1` register at link *link* and return K value.
int IORD_JESD204_TX_GET_M_VAL (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the M value.
int IORD_JESD204_RX_GET_M_VAL (int *link*)	Read the JESD204B RX CSR `ilas_data1` register at *link* and return the M value.
int IORD_JESD204_TX_GET_N_VAL (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the N value.
int IORD_JESD204_RX_GET_N_VAL (int *link*)	Read the JESD204B RX CSR `ilas_data1` register at *link* and return the N value.
int IORD_JESD204_TX_GET_NP_VAL (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the NP value.
int IORD_JESD204_RX_GET_NP_VAL (int *link*)	Read the JESD204B RX CSR `ilas_data1` register at *link* and return the NP value.
int IORD_JESD204_TX_GET_S_VAL (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the S value.
int IORD_JESD204_RX_GET_S_VAL (int *link*)	Read theJESD204B RX CSR `ilas_data1` register at *link* and return the S value.
int IORD_JESD204_TX_GET_HD_VAL (int *link*)	Read the JESD204B TX CSR `ilas_data1` register at *link* and return the HD value.
int IORD_JESD204_RX_GET_HD_VAL (int *link*)	Read the JESD204B RX CSR `ilas_data1` register at *link* and return the HD value.
int IORD_JESD204_TX_LANE_CTRL_REG (int *link*, int *offset*)	Read the JESD204B TX CSR `lane_ctrl_*` register at *link* and return the value.
int IORD_JESD204_RX_LANE_CTRL_REG (int *link*, int *offset*)	Read the JESD204B RX CSR `lane_ctrl_*` register at *link* and return the value.
void IOWR_JESD204_TX_LANE_CTRL_REG (int *link*, int *offset*, int *val*)	Write *val* value into the JESD204B TX CSR `lane_ctrl_*` register at *link*.
void IOWR_JESD204_RX_LANE_CTRL_REG (int *link*, int *offset*, int *val*)	Write *val* value into the JESD204B RX CSR `lane_ctrl_*` register at *link*.

continued...

Function Prototype	Description
int IORD_PIO_CONTROL_REG (void)	Read the PIO control register and return the value.
void IOWR_PIO_CONTROL_REG (int *val*)	Write *val* value into the PIO control register.
int IORD_PIO_STATUS_REG (void)	Read the PIO status register and return thevalue.
int IORD_JESD204_TX_TEST_MODE_REG (int *link*)	Read the JESD204B TX CSR `tx_test` register at *link* and return the value.
int IORD_JESD204_RX_TEST_MODE_REG (int *link*)	Read the JESD204B RX CSR `rx_test` register at *link* and return the value.
void IOWR_JESD204_TX_TEST_MODE_REG (int *link*, int *val*)	Write *val* value into the JESD204B TX CSR `tx_test` register at *link*.
void IOWR_JESD204_RX_TEST_MODE_REG (int *link*, int *val*)	Write *val* value into the JESD204B RX CSR `rx_test` register at *link*.
int IORD_JESD204_RX_ERR0_REG (int *link*)	Read the JESD204B RX CSR `rx_err0` register at *link* and return the value.
void IOWR_JESD204_RX_ERR0_REG (int *link*, int *val*)	Write *val* value into the JESD204B RX CSR `rx_err0` register at *link*.
int IORD_JESD204_RX_ERR1_REG (int *link*)	Read the JESD204B RX CSR `rx_err1` register at *link* and return the value.
void IOWR_JESD204_RX_ERR1_REG (int *link*, int *val*)	Write *val* value into the JESD204B RX CSR `rx_err1` register at *link*.
int IORD_JESD204_TX_ERR_REG (int *link*)	Read the JESD204B TX CSR `tx_err` register at *link* and return the value.
void IOWR_JESD204_TX_ERR_REG (int *link*, int *val*)	Write *val* value into the JESD204B TX CSR `tx_err` register at *link*.
int IORD_JESD204_TX_ERR_EN_REG (int *link*)	Read the JESD204B TX CSR `tx_err_enable` register at *link* and return the value.
void IOWR_JESD204_TX_ERR_EN_REG (int *link*, int *val*)	Write *val* value into the JESD204B TX CSR `tx_err_enable` register at *link*.
int IORD_JESD204_RX_ERR_EN_REG (int *link*)	Read the JESD204B RX CSR `rx_err_enable` register at *link* and return the value.
void IOWR_JESD204_RX_ERR_EN_REG (int *link*, int *val*)	Write *val* value into the JESD204B RX CSR `rx_err_enable` register at *link*.
int IORD_XCVR_NATIVE_A10_REG (int *link*, int *offset*)	Read the transceiver reconfiguration register at *link* and address offset at *offset* and return the value.
void IOWR_XCVR_NATIVE_A10_REG (int *link*, int *offset*, int *val*)	Write *val* value into the transceiver reconfiguration register at *link* and address offset at *offset*.
int IORD_XCVR_ATX_PLL_A10_REG (int *link*, int *instance*, int *offset*)	Read the ATX PLL reconfiguration register indicated by the instance number *instance* at *link* and address offset at *offset* and return the value.
void IOWR_XCVR_ATX_PLL_A10_REG (int *link*, int *instance*, int *offset*, int *val*)	Write *val* value into the ATX PLL reconfiguration register indicated by instance number *instance* at *link* and address offset at *offset*.
int IORD_CORE_PLL_RECONFIG_C0_COUNTER_REG (void)	Read the core PLL reconfiguration `C0` counter register and return the value.
int IORD_CORE_PLL_RECONFIG_C1_COUNTER_REG (void)	Read the core PLL reconfiguration `C1` counter register and return the value.

continued...

Function Prototype	Description
void IOWR_CORE_PLL_RECONFIG_C0_COUNTER_REG (int *val*)	Write *val* value into the core PLL reconfiguration `C0` counter register.
void IOWR_CORE_PLL_RECONFIG_C1_COUNTER_REG (int *val*)	Write *val* value into the core PLL reconfiguration `C1` counter register.
void IOWR_CORE_PLL_RECONFIG_START_REG (int *link*)	Write to core PLL reconfiguration CSR to start the reconfiguration operation.

1.2.11 Customizing the Design Example

Use the following guidelines to customize the design example post-generation.

Related Links

AN803: Implementing ADC-Arria 10 Multi-Link Design with JESD204B RX IP Core

1.2.11.1 Modifying the JESD204B IP Core Parameters

The Platform Designer tool allows only a limited set of design examples to be generated based on the JESD204B IP core parameters selected.

Perform the following instructions to modify the JESD204B IP core parameters post-generation:

1. Open the generated design example project in the Intel Quartus Prime software.
2. Open the `altera_jesd204_subsystem_<data path>.qsys` system in Platform Designer.
3. In the **System Contents** tab, double-click the **altera_jesd204_<data path>** module. This brings up the parameter editor that shows the current parameter settings of the JESD204B IP core.
4. Modify the parameters of the JESD204B IP core module as per your system specifications. When you are done, save the Platform Designer system (**File ➤ Save**).

   Note: The JESD204B IP core and transport layer imposes certain limits on the values that can be entered as parameters. Please refer to the JESD204B IP Core User Guide for a complete listing of the legal parameter values.
5. Click the **Generate HDL** to generate the HDL files needed for Intel Quartus Prime compilation.
6. After the HDL generation is completed, click the **Finish** to save your settings and exit Platform Designer.
7. You have to manually change the system parameters in the top level RTL file to match the parameters that you set in the Platform Designer project, if applicable. Open the top level RTL file (`altera_jesd204_ed_<data path>.sv`) in any text editor of your choice.
8. Modify the system parameters at the top of the file to match the new JESD204B IP core settings in the Platform Designer project, if applicable.
9. Save the file and compile the design in Intel Quartus Prime software as per the instructions in the Compiling and Testing the Design on page 8.

Related Links

JESD204B IP Core User Guide

1.2.11.2 Changing the Data Rate or Reference Clock Frequency

When changing the data rate or reference clock frequency, you must consider the following:

- The relationships between the serial data rate, link clock, and frame clock as described in the *JESD204B IP Core User Guide*.
- Change the PLL output clock settings according to Table 13 on page 30.
- Take note when changing the **F1_FRAMECLK_DIV** and **F2_FRAMECLK_DIV** frame clock division factor parameters in the top level RTL file `altera_jesd204_ed_<data path>.sv` for cases when F=1 or F=2. These parameters further divide-down the frame clock frequency requirement so the resulting clock frequency is within bounds of timing closure for the FPGA core fabric.

The frame clock and the link clock for the following cases share the same frequency:

- F=1—the default parameter value for F1_FRAMECLK_DIV=4
- F=2—the default parameter value for F2_FRAMECLK_DIV=2
- F=4

Perform the following instructions to modify the JESD204B IP core parameters post-generation:

1. Open the generated design example project in the Intel Quartus Prime software.
2. Open the top level `altera_jesd204_ed_qsys_<data path>.qsys` in the Platform Designer.
3. In the **System Contents** tab, right-click the **altera_jesd204_subsystem_<data path>** module and select **Drill into Subsystem**. This opens the `altera_jesd204_subsystem_<data path>.qsys` Platform Designer subsystem.
4. Double-click the **altera_jesd204_<data path>** module. This brings up the parameter editor that shows the current parameter settings of the JESD204B IP core.
5. Change the **Data rate** and **PLL/CDR Reference Clock Frequency** values to meet your system requirements.
6. Modify the clock frequency values of the **device_clk**, **link_clk**, **frame_clk** and **mgmt_clk** clock source modules as necessary to meet your system requirements. Double-click the clock source module to bring up the parameters editor and change the **Clock frequency** value as necessary. Ensure that the values match the clock frequency values that you have entered for the other modules above.
7. Navigate back to the top level **altera_jesd204_ed_qsys_<data path>.qsys** hierarchy.
8. Double-click the **xcvr_atx_pll_0** module to bring up the parameters editor for the ATX PLL module.

   This is the module that generates the serial clock for the TX transceiver PHY.

9. Under the **PLL** subtab, locate the **Output Frequency** group and change the **PLL output frequency** and **PLL integer reference clock frequency** values to meet your system requirements.

   The PLL output frequency is half of the PLL output data rate. Ensure that the data rate and PLL reference clock values match the parameters that you entered into the JESD204B IP core module.

10. Double-click the **core_pll module** to bring up the parameters editor for the core PLL module.

    This is the module that generates the `link_clk` and `frame_clk` clocks that clock the core components.

11. Under the **PLL** subtab, change the **Reference Clock Frequency** value in the **General** group to meet your system requirements.

    Ensure that the reference clock frequency value matches the ones set for the JESD204B IP core and ATX PLL modules.

12. Change the `outclk0` group settings (which correspond to the `link_clk`) and `outclk1` group settings (which correspond to the `frame_clk`) where necessary.

    Ensure that the `link_clk` and `frame_clk` values satisfy the frequency requirements as described in the JESD204B IP Core User Guide.

13. Modify the clock frequency values of the **device_clk**, , **link_clk**, **frame_clk** and **mgmt_clk** clock source modules as necessary to meet your system requirements. Double-click the clock source module to bring up the parameters editor and change the **Clock frequency** value as necessary. Ensure that the values match the clock frequency values that you have entered for the other modules in earlier steps.

14. Click the **Generate HDL** button to generate the HDL files needed for Intel Quartus Prime compilation.

15. After the HDL generation is completed, click the **Finish** to save your Platform Designer settings and exit the Platform Designer window.

16. If the frame_clk settings (outclk1 of the core_pll module) are such that **F1_FRAMECLK_DIV** or **F2_FRAMECLK_DIV** values are changed, change the parameters in the top level design file, `altera_jesd204_ed_<data path>.sv`.

17. Modify the clock constraints in the SDC constraints file (`altera_jesd204_ed_<data path>.sdc`) to reflect your new clock frequency values, if applicable. The following constraints should be modified:

    ```
     create_clock -name device_clk -period <clock period value in ns>
    [get_ports device_clk]

    create_clock -name mgmt_clk -period <clock period value in ns> [get_nodes
    mgmt_clk]
    ```

18. Save the file and compile the design in Intel Quartus Prime software as per the instructions in the Compiling and Testing the Design on page 8.

Related Links

JESD204B IP Core User Guide

1.3 Intel FPGA JESD204B Design Example User Guide for Intel Arria 10 Devices Document Archives

If a design example version is not listed, the user guide for the previous IP core version applies.

IP Core Version	User Guide
17.0	JESD204B IP Core User Guide

1.4 Document Revision History for Intel FPGA JESD204B Design Example User Guide for Intel Arria 10 Devices

Date	Version	Changes
November 2017	2017.11.06	• Added information about simplex and duplex ATX reference clock frequencies. • Defined (`altera_jesd204_ed_<data path>.sv`) as the top level RTL file in *Core PLL*. • Added *Frame Clock and Link Clock Relationship* subsection. • Defined top level RTL file in *Changing the Data Rate or Reference Clock Frequency*. • Updated SDC constraint to be modified in *Changing the Data Rate or Reference Clock Frequency*. • Added *get_master_index* procedure in *Procedures in the main.tcl System Console Script* table. • Updated document title. • Updated instances of Qsys to Platform Designer. • Added note to *System Clocking for the Design Example* table about additional jitter introduced to the ATX, fPLL, and transmit PLL output when using reference clock from a cascaded PLL output, global clock or core clock network.
May 2017	2017.05.08	Initial release.