JESD204B IP Core Design Example User Guide

2016.10.31

UG-20029 Subscribe Send Feedback

The Altera JESD204B IP core offers two design examples:

- RTL State Machine Control (supports Arria V, Cyclone V, Stratix V, and Arria 10 devices only)
- Nios II Control (supports Arria 10 devices only)

You can generate the JESD204B IP core design examples through the IP catalog in the Quartus® Prime software.

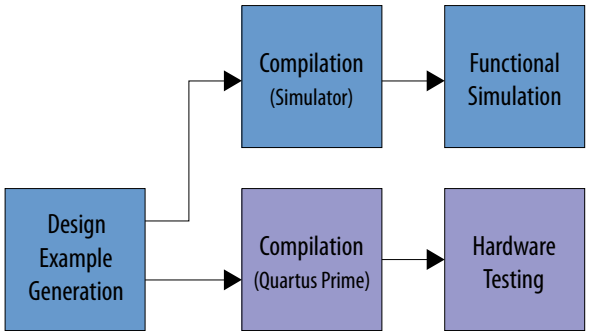
Related Information

JESD204B IP Core User Guide

JESD204B Design Example 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



Directory Structure

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

Figure 2: Directory Structure for the JESD204B Design Example

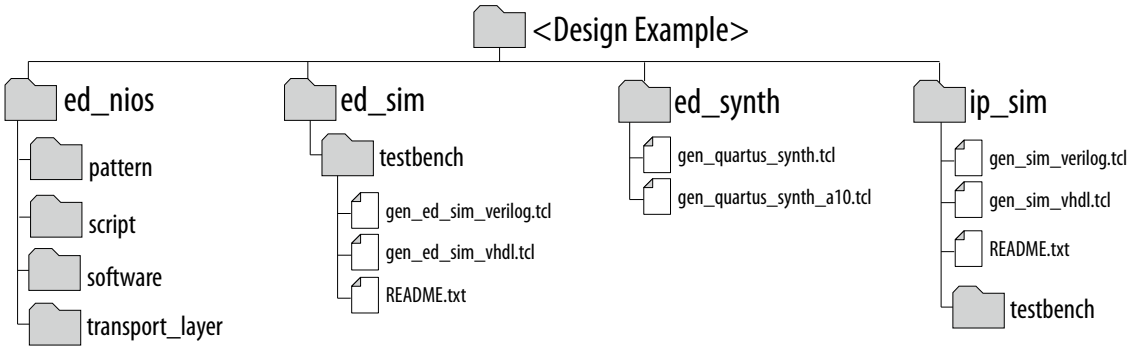


Table 1: Directory and File Description

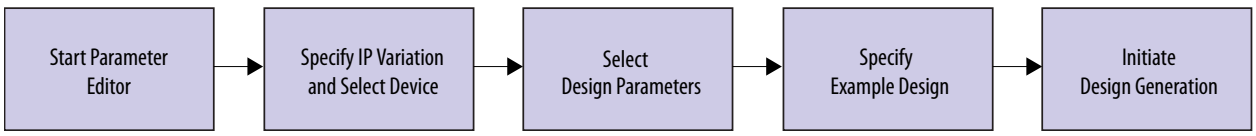
Directory/File	Description
`ed_nios`	The folder that contains the compilation scripts to generate the Nios II design example for compilation.
`ed_sim` (1)	The folder that contains the testbench files.

`ed_sim/testbench/cadence`
`ed_sim/testbench/mentor`
`ed_sim/testbench/synopsys/vcs`
	The folder that contains the simulation script. It also serves as a working area for the simulator.

`ed_synth` (1)	The folder that contains the design example synthesizable components.
`ip_sim`	The folder that contains the simulation script to generate the JESD204B IP Core Verilog/VHDL simulation model.

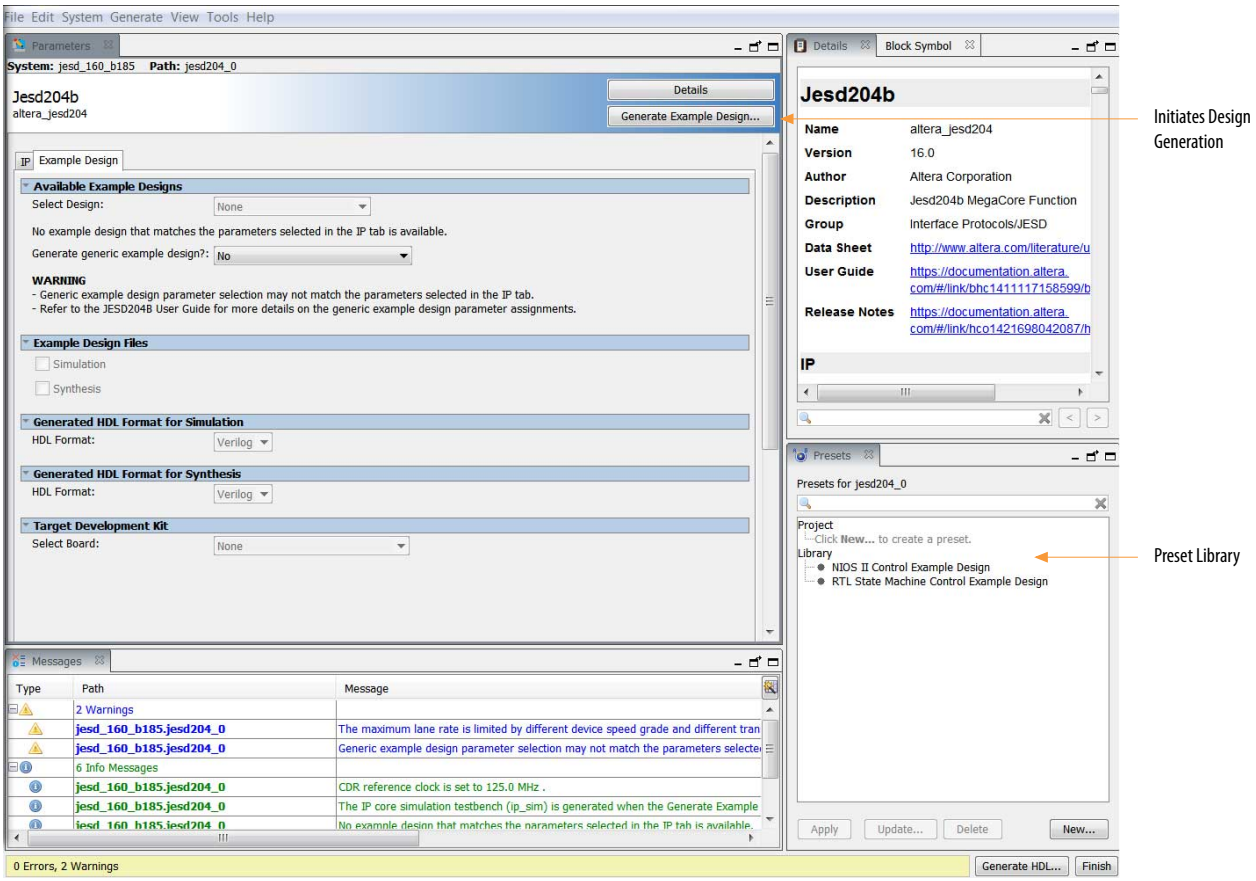
Generating the Design

You can use the JESD204B IP core parameter editor in the Quartus Prime software to generate the design example.



(1) Only for RTL State Machine Control design example.

Figure 3: Example Design Tab



Procedure

This is a general procedure on how to generate the JESD204B design example.

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. 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. Specify the parameters for your design.
5. 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 Information

Selecting and Generating the Design Example on page 11
Detailed procedure on how to generate the design examples.

Design Example Parameters

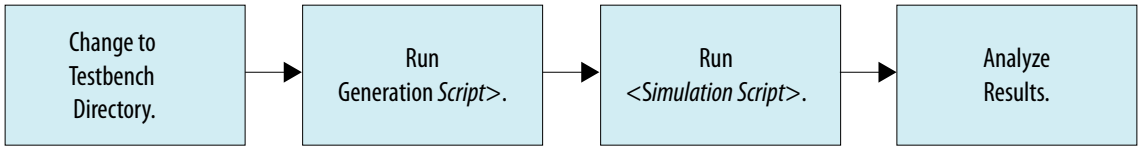
The JESD204B IP parameter editor includes a *Example Design* tab for you to specify certain parameters before generating the design example.

Table 2: Parameters in the Example Design Tab

Parameter	Description
Select Design	Available example designs for the IP parameter settings. When you select a design from the Preset library, this field shows the selected design.
Generate generic example design?	Option to generate a generic design example. This parameter is available when the **Select Design** option displays **None**.
Example Design Files	The files to generate for different development phase. Simulation—when selected, the necessary files for simulating the design example are generated. Synthesis—when selected, the synthesis files are generated. Use these files to compile the design in the Quartus Prime software for hardware testing.
Generate HDL Format for Simulation	The format of the RTL files for simulation—Verilog or VHDL.
Generate HDL Format for Synthesis	The format of the RTL files for synthesis—Verilog or VHDL.
Target Development Kit	Supported hardware for design implementation.

Compiling and Simulating the Design

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



Procedure

To compile and simulate the design:

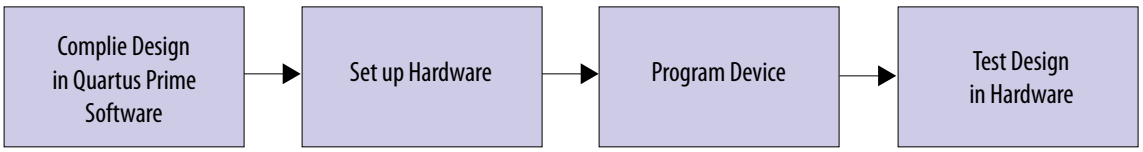
1. Change the working directory to <*example_design_directory*>**/ed_sim/**
2. Type one of the following to generate the simulation files:
   - For verilog: `quartus_sh -t gen_ed_sim_verilog.tcl`
   - For vhdl: `quartus_sh -t gen_ed_sim_vhdl.tcl`
3. Change the working directory to <*example_design_directory*>**/ed_sim/testbench/**<*Simulator*>.
4. Run the simulation script for the simulator of your choice. Refer to the table below.

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

A successful simulation ends with the following message, "Simulation stopped due to successful completion! Simulation passed."

Compiling and Testing the Design

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



Follow these steps to compile and test the design in hardware:

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

   The timing constraints for the design example and the design components are automatically loaded during compilation.
2. Connect the development board to the host computer.
3. Configure the FPGA on the development board using the generated **.sof** file (**Tools**> **Programmer**).

   The Quartus Prime version 15.1 only supports programming file generation for Arria 10 engineering devices. For more information on support for Arria 10 production devices, contact your local Altera representative or use the support link from Altera website.

For details on how to implement the JESD204B design example on the Arria 10 GX FPGA Development Kit, refer to **Implementing the Design on the Development Kit** on page 88.

Supported Configurations

The design examples only support a limited set of JESD204B IP core parameter configurations.

The IP Catalog parameter editor allows you to generate a design example only if the parameter configurations matches those in the tables below.

Table 3: Supported JESD204B IP Core Parameter Configurations (L, M, F Values)

JESD204B IP Parameters			Applicable Devices
L	M	F	
1	1	2	V series and Arria® 10
1	1	4	V series and Arria 10
1	1	8	Arria 10
1	2	4	V series and Arria 10
1	2	8	Arria 10
1	4	8	V series and Arria 10
2	1	1	V series and Arria 10
2	1	2	V series and Arria 10
2	1	4	V series and Arria 10
2	1	8	Arria 10
2	2	2	V series and Arria 10
2	2	4	V series and Arria 10
2	2	8	Arria 10
2	4	4	V series and Arria 10
2	4	8	Arria 10
2	8	8	Arria 10
4	1	1	Arria 10
4	1	2	Arria 10
4	2	1	V series and Arria 10
4	2	2	V series and Arria 10
4	2	4	Arria 10
4	2	8	Arria 10
4	4	2	V series and Arria 10
4	4	4	V series and Arria 10
4	4	8	Arria 10
4	8	4	V series and Arria 10
4	8	8	Arria 10
4	16	8	Arria 10
6	1	1	Arria 10
6	3	1	Arria 10
8	1	1	V series and Arria 10
8	1	2	Arria 10

JESD204B IP Parameters			Applicable Devices
L	M	F	
8	2	1	V series and Arria 10
8	2	2	Arria 10
8	2	4	Arria 10
8	4	1	V series and Arria 10
8	4	2	V series and Arria 10
8	4	4	Arria 10
8	4	8	Arria 10
8	8	2	Arria 10
8	8	4	Arria 10
8	8	8	Arria 10
8	16	4	Arria 10
8	16	8	Arria 10
8	32	8	Arria 10

Table 4: Supported JESD204B IP Core Parameter Configurations

JESD204B IP Parameters	Value
Wrapper Options	Both Base and Phy
Data Path	Duplex
JESD204B Subclass	1
Data Rate	• 6144 (Arria V, Stratix V, and Arria 10) • 5000 (Cyclone V)
PCS Option	Enabled Hard PCS
PLL Type	CMU
Bonding Mode	• Bonded (For **Enable Transceiver Dynamic Reconfiguration** option set to No) • Non-bonded (For **Enable Transceiver Dynamic Reconfiguration** option set to Yes)

Enable Transceiver Dynamic Reconfiguration	• No (Bonding Mode must be set to Bonded)
	• Yes (Bonding Mode must be set to Non-bonded)
	• For Arria 10:
	  • No (only the RTL state machine control design example is available for generation)
	  • Yes (only Nios control design example is available for generation)

JESD204B IP Parameters	Value
PLL/CDR Reference Clock Frequency	• 153.6 (Arria V, Stratix V, and Arria 10; all supported L parameter values except L=8) • 307.2 (Arria V, Stratix V, and Arria 10; L=8) • 125 (Cyclone V; all supported L parameter values except L=8) • 250 (Cyclone V; L=8)
Enable Bit Reversal And Byte Reversal	No
N	• 16 (V series) • 12, 13, 14, 15, 16 (Arria 10)
N'	16
CS	• 0 (V series) • 0-3 (Arria 10)
CF	0
High Density User Data Format (HD)	• 0 (V series) • 0 for F = 2, 4, 8 (Arria 10) • 1 for F = 1 (Arria 10)
Enable scramble (SCR)	Yes
Enable Error Code Correction (ECC_EN)	Yes

Table 5: Valid Options Available for Design Example Generation

Device	Supported JESD204B IP Core Configurations	Example Design Type	Generate Generic Example Design?	Example Design Files	HDL Format	Target Development Kit
Stratix V, Arria V, Cyclone V	No	None	No	—	—	—
	No	None	Generic RTL	Simulation	Verilog, VHDL	—
	No	None	Generic RTL	Synthesis	Verilog (2)	—
	Yes	RTL	—	Simulation	Verilog, VHDL	—
	Yes	RTL	—	Synthesis	Verilog (2)	—

(2) For synthesis flow, only the Verilog HDL format is available.

Device	Supported JESD204B IP Core Configurations	Example Design Type	Generate Generic Example Design?	Example Design Files	HDL Format	Target Development Kit
Arria 10	No	None	No	—	—	—
	No	None	Generic RTL	Simulation	Verilog, VHDL	—
	No	None	Generic RTL	Synthesis	Verilog (2)	—
	No	None	Generic Nios	Synthesis (3)	Verilog (2)	• None • Arria 10 (4)
	Yes	RTL	—	Simulation	Verilog, VHDL	—
	Yes	RTL	—	Synthesis	Verilog (2)	—
	Yes	Nios	—	Synthesis (3)	Verilog (2)	• None • Arria 10 (4)

Generic Design Example

If the JESD204B IP parameters that you select does not match any design example that is available, there is an option for you to generate a generic design example.

A generic design example is a design that has pre-selected IP parameters that matches the list of supported IP parameters for the design example.

Note: The generated generic example design may have IP parameters that differ from the parameters of your IP core. Modify the generic example design according to your system specifications.

The table below lists the parameters in the generic design example.

Table 6: IP Parameter Settings for Generic Design Example

JESD204B IP Parameters	Design Example	
	Generic RTL State Machine Control	Generic Nios II Control
Devices Support	V series and Arria 10	Arria 10
L	2	2
M	2	2

(3) For Nios II control unit example design option, only synthesis filesets are available and only Verilog HDL format is supported.

(4) For Nios II control unit example design option, you can choose not to target your design to any development kit or choose to target the Arria 10 GX FPGA Development Kit.

JESD204B IP Parameters	Design Example	
	Generic RTL State Machine Control	Generic Nios II Control
F	2	2
K	16	16
S	1	1
Wrapper Options	Both Base and Phy	Both Base and Phy
Data Path	Duplex	Duplex
JESD204B Subclass	1	1
Data Rate	6144	6144
PCS Option	Enabled Hard PCS	Enabled Hard PCS
PLL Type	CMU	CMU
Bonding Mode	Bonded	Non-bonded
Enable Transceiver Dynamic Reconfiguration	No	Yes
PLL/CDR Reference Clock Frequency	153.6	153.6
Enable Bit Reversal And Byte Reversal	No	No
N	16	16
N'	16	16
CS	0	0
CF	0	0
High Density User Data Format (HD)	0	0
Enable scramble (SCR)	Yes	Yes
Enable Error Code Correction (ECC_EN)	Yes	Yes

Presets

Standard presets allow instant entry of pre-selected parameter values in the **IP** and **Example Design** tabs. You can select the presets at the lower right window in the parameter editor.

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. There are two preset settings available in the library:

- RTL State Machine Control example design
- Nios II Control example design

Note: Selecting a preset will overwrite any pre-existing parameter selections for the IP core under the IP tab. Use the generic example design option instead if you want to retain your pre-selected IP core parameter selections.

Table 7: Preset Settings

JESD204B IP Parameters	Presets	
	RTL State Machine Control	Nios II Control
Devices Support	V series and Arria 10	Arria 10
L	2	2
M	2	2
F	2	2
K	16	16
S	1	1
Wrapper Options	Both Base and Phy	Both Base and Phy
Data Path	Duplex	Duplex
JESD204B Subclass	1	1
Data Rate	6144	6144
PCS Option	Enabled Hard PCS	Enabled Hard PCS
PLL Type	CMU	CMU
Bonding Mode	Bonded	Non-bonded
Enable Transceiver Dynamic Reconfiguration	Yes	Yes
PLL/CDR Reference Clock Frequency	153.6	153.6
Enable Bit Reversal And Byte Reversal	No	No
N	16	16
N’	16	16
CS	0	0
CF	0	0
High Density User Data Format (HD)	0	0
Enable scramble (SCR)	Yes	Yes
Enable Error Code Correction (ECC_EN)	Yes	Yes

Selecting and Generating the Design Example

You can access and generate the IP core design example through the IP Catalog parameter editor.

Follow the steps below to launch the design example GUI and generate the design example.

1. In the **IP Catalog** (**Tools** > **IP Catalog**), select the JESD204B IP core.
2. Specify an entity name and location for your custom IP variation. This name identifies the IP core variation files in your project. If prompted, also specify the target Altera device family. Click **OK**.
3. In the parameter editor, click on the **IP** tab and set the JESD204B IP core parameters as per your specifications. If you want to pre-fill the parameters with the set of parameter values that result in a valid example design, use the presets in the **Presets** tab. Refer to **Presets** on page 10 for more details.
4. In the parameter editor, click on the **Example Design** tab.
5. Under the **Available Example Designs** section, select the available designs. The options you can select are based on the design examples that are available.
   - **None**: No design example available that matches the IP parameters selected.
   - **RTL State Machine Control**: Design example has RTL state machine as control unit.
   - **Nios II Control**: Design example has Nios II processor as control unit. This option is available for Arria 10 devices only.
6. If the **Select Design** option under the **Available Example Designs** section displays **None**, the **Generate generic example design** selection appears. In the **Generate generic example design** list, select one of the options available to generate a generic design example.
   - **No**: No generic design example will be generated.
   - **Generic RTL State Machine Control**: Generic design example has RTL state machine as control unit.
   - **Generic Nios Control**: Generic design example has Nios II processor as control unit. This option is available for Arria 10 devices only.

   Note: The generic design example parameter selection may not match the parameters that you selected in the **IP** tab. You can modify the generated generic design example files to match your desired IP parameter settings.
7. Under the **Example Design Files** section, select the desired design example files. The options you can select are based on the design examples that are available.
   - **Simulation**: Generate simulation files.
   - **Synthesis**: Generate synthesis files.
8. Under the **Generated HDL Format for Simulation** section (only available if the **Simulation** option is checked), select the desired HDL format. The options you can select are based on the design examples that are available.
9. Under the **Generated HDL Format for Synthesis** section (only available if the **Synthesis** option is checked), select the desired HDL format. The options you can select are based on the design examples that are available.
10. Under the **Target Development Kit** section, select the development kit that the design example will target. The options you can select are based on the design example that are available.
    - **None**: Design example does not target any board. The target device is set to a default device and may not match your selected target device in the Quartus project.
    - **Arria 10 GX FPGA Development Kit**: Design example is targeted for Arria 10 GX FPGA development kit. This option is available for Arria 10 devices only. The target device is Arria 10 GX FPGA and may not match your selected target device in the Quartus project.

Note: The hardware example design targets an Arria 10 ES3 device. It cannot function correctly on an Arria 10 production device.

11. Click the **Generate Example Design** button on the top right corner to generate the design example based on your settings.
    a. If the selected design example in step 5 is **None**, and the generate generic design example selection in step 6 is **No**, an error message is displayed and no design example will be generated.
    b. If the selected design example in step 5 is **RTL State Machine Control** or **Nios II Control**, the relevant design example is generated with the JESD204B parameters matching the JESD204B IP parameter settings in the **IP** tab.
    c. If the selected design example in step 5 is **None**, and the generate generic design example selection from Step 6 is **Generic RTL State Machine Control** or **Generic Nios II Control**, the relevant generic design example is generated with the pre-set JESD204B parameters. You can then modify the JESD204B parameters directly in the generated design files to match your desired parameter settings.

The design example files are generated in the folder that you specified when you clicked on **Generate Example Design**. This is a self-contained design example folder that is in the same directory that contains the generated IP files. All the files necessary to compile and run the design example, including an independently generated JESD204B IP core module that is separate from the core module generated from the **IP** tab is stored in this folder and its sub directories.

Related Information

- **Generating and Simulating the Design Example** on page 68
- **Generating the Design Example For Compilation** on page 70
- **Compiling the Design Example for Synthesis** on page 87
- **Running the Software Control Flow** on page 92

Design Example with RTL State Machine Control Unit

This design example with RTL state machine control unit is the legacy design example that was first released in Quartus II version 13.1. The design example has the following key features:

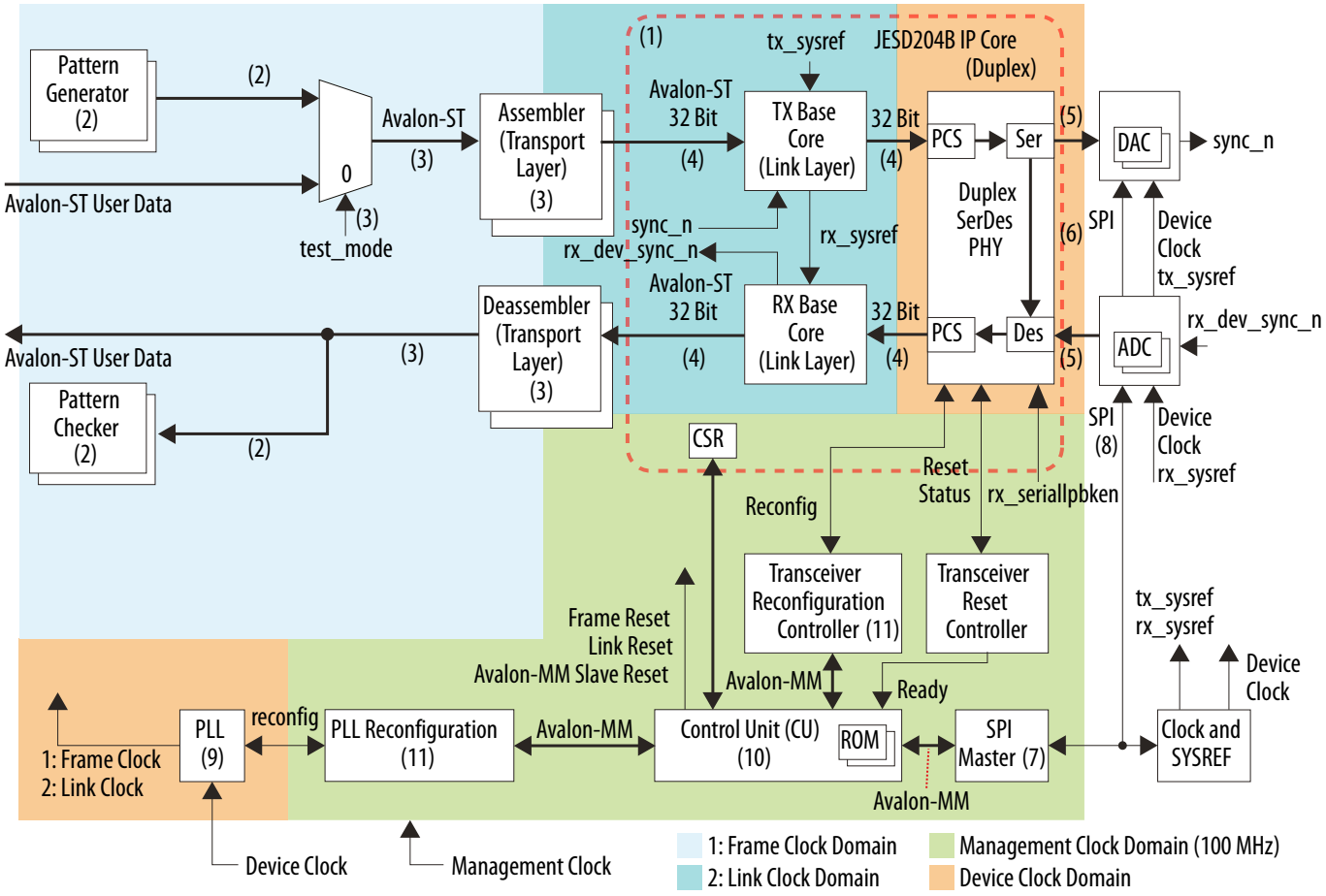
- Supports Arria V, Cyclone V, Stratix V, and Arria 10 devices.
- Purely hardware-implemented control path, no software control features.
- Lower FPGA core resource utilization compared to Nios II processor control unit design example.
- Available as a synthesizable design entity and a simulation model.

The design example entity consists of various components that interface with the JESD204B IP core to demonstrate the following features:

- single or multiple link configuration
- different LMF settings with scrambling and internal serial loopback enabled
- interoperability against diverse converter devices
- dynamic reconfiguration

Figure 4: RTL State Machine Control Unit Design Example Block Diagram

This figure illustrates the high level system architecture of the JESD204B IP core design example.



The list below describes the mechanism of the design example architecture (with reference to the note numbers in the design example block diagram).

1. For multiple links, the JESD204B IP core is instantiated multiple times. For example, in 2x112 (LMF) configuration, two cores are instantiated, where each core is configured at LMF=112.
2. The number of pattern generator or pattern checker instances is equivalent to the parameter value of LINK. The data bus width per instance is equivalent to the value of FRAMECLK_DIV*M*S*N.
3. The number of transport layer instances is equivalent to the parameter value of LINK. The legal value of LINK is 1 and 2. The data bus width per instance is equivalent to the value of FRAMECLK_DIV*M*S*N. The `test_mode` = 0 signal indicates a normal operation mode, where the assembler takes data from the Avalon-ST source. Otherwise, the assembler takes data from the pattern generator.
4. The Avalon-ST interface data bus is fixed at 32-bit. The number of 32-bit data bus is equal to the number of lanes (L).
5. The number of lanes per converter device (L).

6. You can enable internal serial loopback by setting the `rx_seriallpbken` input signal. You can dynamically toggle this input signal. When toggled to 1, the RX path takes the serial input from the TX path internally in the FPGA. When toggled to 0, the RX path takes the serial input from the external converter device. During internal serial loopback mode, the assembler takes input from the pattern generator.
7. A single serial port interface (SPI) master instance can control multiple SPI slaves. The SPI master is a 4-wire instance. If the SPI slave is a 3-wire instance, use a bidirectional I/O buffer in between the master and slave to interface the 4-wire master to 3-wire slave.
8. The SPI protocol interface. All slaves share the same data lines (MISO and MOSI, or DATAIO). Each slave has its own slave select or chip select line (`SS_n`).
9. The PLL takes the device clock from an external clock chip as the input reference. The PLL generates two output clocks (utilizing two output counters from a single VCO). Clock 1 is the frame clock for the transport layer, pattern generator, and pattern checker. Clock 2 is the link clock for the transport and link layer.
10. The control unit implements a memory initialization file (MIF) method for configuring the SPI. Each MIF corresponds to a separate external converter per device or clock chip. For example, in a system that interacts with both DAC and ADC, two MIFs are needed—one each for DAC and ADC.
11. The PLL reconfiguration and transceiver reconfiguration controller instances are only required for run time reconfiguration of the data rate.

Related Information

System Parameters on page 51
Shows illustrations of single and multiple JESD204B links.

Design Example Components

The RTL State Machine Control Unit design example for the JESD204B IP core consists of the following components:

- PLL
- PLL reconfiguration
- Transceiver reconfiguration controller
- Transceiver reset controller
- Pattern generator
- Pattern checker
- Assembler and deassembler (in the transport layer)
- SPI
- Control unit

The following sections describe in detail the function of each component.

PLL

The design example requires four different clock domains—device clock, management clock, frame clock, and link clock.

Typically, the device clock is generated from an external converter or a clock device while the management clock (AVS clock) is generated from an on-board 100 MHz oscillator.

For instance, if the JESD204B IP core is configured at data rate of 6.144 Gbps, transceiver reference clock frequency of 153.6 MHz, and number of octets per frame (F) = 2, the example below indicates the PLL clock frequencies:

- device clock = transceiver reference clock frequency = 153.6 MHz
- link clock = 6144 / 40 = 153.6 MHz
- frame clock = 153.6 x 32 / (8 x 2) = 307.2 MHz

Related Information

Clocking Scheme
More information about the JESD204B IP core clocks.

PLL Reconfiguration

The PLL reconfiguration utilizes the ALTERA_PLL_RECONFIG IP core to implement reconfiguration logic to facilitate dynamic real-time reconfiguration of PLLs in Altera devices. You can use this megafunction IP core to update the output clock frequency, PLL bandwidth, and phase shifts in real time, without reconfiguring the entire FPGA.

The design example uses the MIF approach to reconfigure the core PLL. The ALTERA_PLL_RECONFIG IP core has two parameter options—**Enable MIF Streaming** and **Path to MIF file**—for the MIF input. Turn on **Enable MIF Streaming** option and set the *core_pll.mif* as the value to **Path to MIF file** parameter.

The following PLL reconfiguration Avalon-MM operations occurs during data rate reconfiguration.

Table 8: PLL Reconfiguration Operation

Operation	Avalon-MM Interface Signal	Byte Address Offset (6bits)	Bit	Value
Arria V and Stratix V Devices				
Set MIF base address	pll_mgmt_*	0x01F	[8:0]	0x000 (maximum configuration) or 0x02E (downscale configuration)
Write to the START register to begin	pll_mgmt_*	0x02	[0:0]	0x01
Arria 10 Devices				
Start MIF streaming with MIF base address specified in data value	pll_mgmt_*	0x010	[31:0]	0x000 (maximum configuration) or 0x02E (downscale configuration) (5)

(5) The MIF base address is 9 bits (LSB). The remaining bits are reserved.

Related Information

AN 661: Implementing Fractional PLL Reconfiguration with Altera PLL and Altera PLL Reconfig Megafunctions
More information about the MIF streaming option.

Transceiver Reconfiguration Controller

The transceiver reconfiguration controller allows you to change the device transceiver settings at any time. Any portion of the transceiver can be selectively reconfigured. Each portion of the reconfiguration requires a read-modify-write operation (read first, then write), in such a way that it modifies only the appropriate bits in a register and not changing other bits.

In the design example, MIF approach is used to reconfigure the ATX PLL and transceiver channel in the JESD204 IP core via the Transceiver Reconfiguration Controller. The number of reconfiguration interface is determined by number of lanes (L) + number of TX_PLL (different number of TX_PLL for bonded and non-bonded mode). Since the MIF approach reconfiguration for transceiver only supports non-bonded mode, the number of TX_PLL is equal to number of lanes. The number of reconfiguration interface = 2 x number of lanes (L).

The transceiver reconfiguration controller interfaces:

- MIF Reconfiguration Avalon-MM master interface—connects to the MIF ROM.
- Transceiver Reconfiguration interface—connects to the JESD204B IP core, which eventually connects to the native PHY.
- Reconfiguration Management Avalon-MM slave interface—connects to the control unit.

Note: The transceiver reconfiguration controller is only used in Arria V and Stratix V devices. For Arria 10 devices, the control unit directly communicates with the transceiver in the JESD204B IP core through the `reconfig_avmm_*` interface signals.

The following transceiver reconfiguration controller Avalon-MM operations are involved during data rate reconfiguration.

Table 9: Transceiver Reconfiguration Controller Operation for Arria V and Stratix V Devices

Operation	Avalon-MM Interface Signal	Byte Address Offset (6bits)	Bit	Value
Write logical channel number	reconfig_mgmt_*	0x38	[9:0]	0
Write MIF mode	reconfig_mgmt_*	0x3A	[3:2]	2'b00
Write 0 to streamer offset register	reconfig_mgmt_*	0x3B	[15:0]	0
Write MIF base address to streamer data register	reconfig_mgmt_*	0X3C	[31:0]	*32'h1000
Initiate a write of all the above data	reconfig_mgmt_*	0x3A	[0]	1'b1
Write 1 to streamer offset register	reconfig_mgmt_*	0x3B	[15:0]	1
Write to streamer data register to set up MIF streaming	reconfig_mgmt_*	0x3C	[31:0]	3

Operation	Avalon-MM Interface Signal	Byte Address Offset (6bits)	Bit	Value
Initiate a write of all the above data to start streaming the MIF	reconfig_mgmt_*	0x3A	[0]	1'b1
Read the busy bit to determine when the write has completed	reconfig_mgmt_*	0x3A	[8]	1: Busy 0: Operation completed

Note: The above steps are repeated for the number of channels and followed by the number of TX_PLLs.

For Arria 10 devices, the only Avalon-MM operation is a direct write to the transceiver register through the `reconfig_avmm_*` interface at the JESD204B IP core. Every line in the MIF is `DPRIO_ADDR[25:16]+ BIT_MASK[15:8]+ DATA[7:0]`. The control unit maps the `DPRIO_ADDR` to `reconfig_avmm_address` and `BIT_MASK + DATA` to `reconfig_avmm_data`.

Related Information

- **Altera Transceiver PHY IP Core User Guide**
  More information about the transceiver reconfiguration controller.
- **Altera Arria 10 Transceiver PHY IP Core User Guide**

Transceiver Reset Controller

The transceiver reset controller uses the Altera's Transceiver PHY Reset Controller IP Core to ensure a reliable initialization of the transceiver. The reset controller has separate reset controls per channel to handle synchronization of reset inputs, hysteresis of PLL locked status, and automatic or manual reset recovery mode.

In this design example, the reset controller targets both the TX and RX channels. The **TX PLL**, **TX Channel**, and **RX Channel** parameters are programmable to accommodate single and multiple (2) JESD204B links.

Related Information

- **Altera Transceiver PHY IP Core User Guide**
  More information about the Transceiver PHY Reset Controller IP Core.
- **Arria V Device Handbook, Volume 2: Transceivers**
  More information about the device usage mode.

Pattern Generator

The pattern generator instantiates any supported generators and has an output multiplexer to select which generated pattern to forward to the transport layer based on the test mode during run time. Additionally, the pattern generator also supports run-time reconfiguration (downscale) on the number of converters per device (M) & samples per converter per frame (S).

The pattern generator can be a parallel PRBS, alternate checkerboard, or ramp wave generator. The data output bus width of the pattern generator is equivalent to the value of FRAMECLK_DIV × M × S × N.

The pattern generator includes a *REVERSE_DATA* parameter to control data arrangement at the output. The default value of this parameter is 0.

- 0—no data rearrangement at the output of the generator.
- 1—data rearrangement at the output of the generator.

For example, when M=2, S=1, N=16, F1/F2_FRAMECLK_DIV=1, the input or output data width equals to [31:0], with the following data arrangement:

0: {m1s0[31:16], m0s0[15:0]}
1: {m0s0[31:16], m1s0[15:0]}

Parallel PRBS Generator

PRBS generator circuits often consists of simple shift registers with feedback that serve as test sources for serial data links. The output sequence is not truly random but repeats after 2^X–1 bits, where X denotes the length of the shift register. Polynomial notation—which the polynomial order corresponds to the length of the shift register and the period of PRBS—provides a method of describing the sequence.

Alternate Checkerboard Generator

The alternate checkerboard generator circuit consists of simple flip registers that serve as test sources for serial data links.

The output sequence of subsequent N-bits sample is generated by inverting the previous N-bits (counting from LSB to MSB) of the same data pattern at that clock cycle. The first N-bits sample from LSB of the data pattern on next clock cycle is generated by inverting the last N-bits sample on the MSB of the data pattern on current clock cycle.

Ramp Wave Generator

The ramp wave generator circuit consists of a simple register and adders that serve as test sources for serial data links.

The output sequence of subsequent N-bits sample is an increment by one of the previous N-bits sample (counting from LSB to MSB) in the same data pattern at that clock cycle. The first N-bits sample from LSB of the data pattern on next clock cycle is generated by an increment by one of the last N-bits sample on the MSB of the data pattern on current clock cycle.

Pattern Checker

The pattern checker instantiates any supported checkers and support run time reconfiguration (downscale) of the number of converters per device (M) and samples per converter per frame (S).

The pattern checker can be either a parallel PRBS checker, alternate checkerboard checker, or ramp wave checker. The data input bus width of the pattern checker is equivalent to the value of FRAMECLK_DIV × M × S × N.

The pattern checker includes an *ERR_THRESHOLD* parameter to control the number of error tolerance allowed in the checker. The default value of this parameter is 1.

The pattern checker also includes a *REVERSE_DATA* parameter to control data arrangement at the input. The default value of this parameter is 0.

- 0—no data rearrangement at the input of the checker.
- 1—data rearrangement at the input of the checker.

Parallel PRBS Checker

The PRBS checker contains the same polynomial as in the PRBS generator. The polynomial is only updated when the enable signal is active, which indicates that the input data is valid. The feedback path is XOR'ed with the input data to do a comparison. The checker flags an error when it finds any single mismatch between polynomial data and input data.

Alternate Checkerboard Checker

The alternate checkerboard checker is implemented in the same way as in the alternate checkerboard generator. To do a comparison, an initial seed internally generates a set of expected data pattern result to XOR'ed with the input data. The seed is updated only when the enable signal is active, which indicates that the input data is valid. The checker flags an error when it finds any single mismatch between the expected data and input data.

Ramp Wave Checker

The ramp wave checker is implemented in the same way as in the ramp wave generator. To do a comparison, an initial seed internally generates a set of expected data pattern result to XOR'ed with the input data. The seed is updated only when the enable signal is active, which indicates that the input data is valid. The checker flags an error when it finds any single mismatch between the expected data and input data.

Transport Layer

The transport layer in the JESD204B IP core consists of an assembler at the TX path and a deassembler at the RX path.

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

- The 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.
- The 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.

Supported System Configuration

The transport layer supports static configurations where before compilation, you can modify the configurations using the IP core's parameter editor in the Quartus Prime software. To change to another configu-

ration, you have to recompile the design. The following list describes the supported configurations for the transport layer:

- Data rate (maximum) = 12.5 Gbps (`F1_FRAMECLK_DIV` = 4 and `F2_FRAMECLK_DIV` = 2)
- L = 1–8
- F = 1, 2, 4, 8
- N = 12, 13, 14, 15, 16
- N' = 16
- CS = 0–3
- CF = 0
- HD = 0 (for F=2, 4, 8), 1 (for F=1)

Dynamic Downscaling Of System Parameters (L, N, and F)

The Dynamic Downscaling of System Parameters (DDSP) feature enables you to dynamically downscale specific JESD204B system parameters through the CSR, without having to recompile the FPGA.

The transport layer supports dynamic downscaling of parameters L, F, and N only. The supported M and S parameters are determined by the L, F, and N' parameters. Some parameters (for example, CS and N') do not have this capability in the transport layer. If you needs to change any of these parameters, you must recompile the system.

You are advised to connect the power down channels to higher indexes and connect used channel at lower lanes. Otherwise, you have to reroute the physical-used channels to lower lanes externally when connecting the IP core to the transport layer. For example, when L = 4 and csr_l = 8'd1 (which means two lanes out of four lanes are active), with lane 1 and lane 3 being powered down, connection from the MAC to the transport layer for lane 0 remains. However, lane 1 is powered down while lane 2 is not powered down. Thus, lane 2 output from the MAC should be rerouted to lane 1 data input of the transport layer. The data port for those power-down channels will be tied off within the transport layer.

The 16-bit N' data for F = 1 is formed through the data from 2 lanes. Thus, F = 1 is not supported for odd number of lanes, for example, when LMF = 128. In this case, you can only reconfigure from F = 8 to F = 4 and F = 2 but not F = 1.

Relationship Between Frame Clock and Link Clock

The frame clock and link clock are synchronous.

The ratio of `link_clk` period to `frame_clk` period is given by this formula:

```
32 x L / (M x S x N')
```

Table 10: txframe_clk and rxframe_clk Frequency for Different F Parameter Settings

For a given f_{txlink} (txlink_clk frequency) and f_{rxlink} (rxlink_clk frequency), the $f_{txframe}$ (txframe_clk frequency) and $f_{rxframe}$ (rxframe_clk frequency) are derived from the formula listed in this table.

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`)

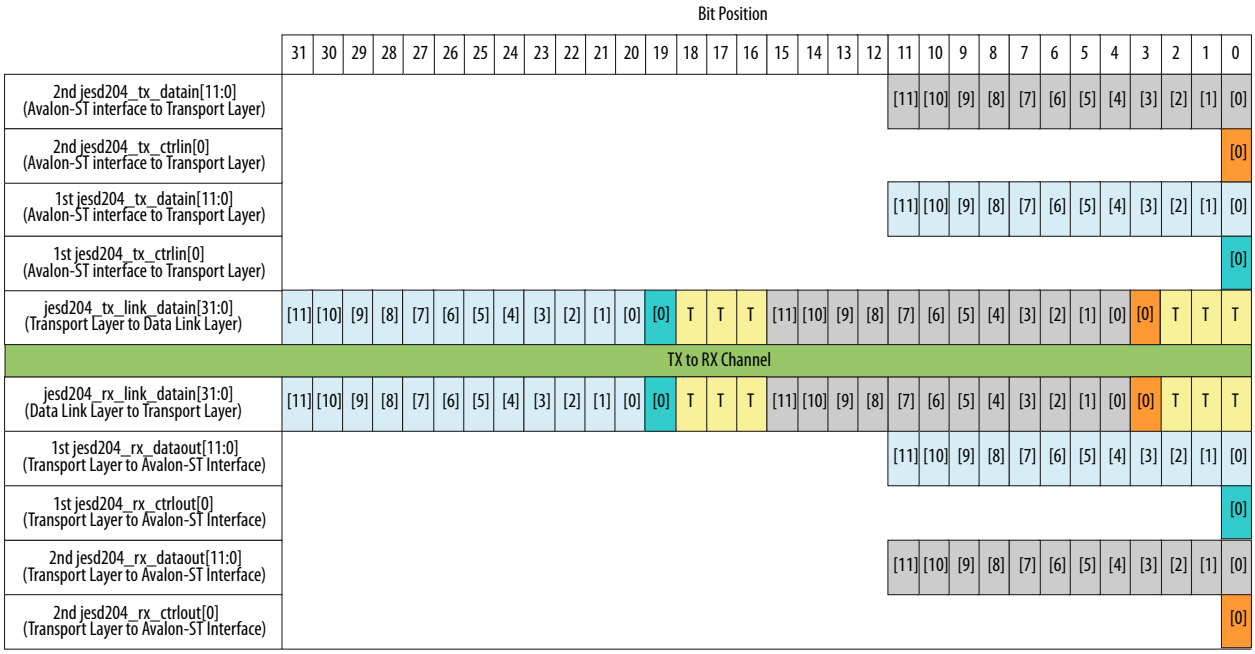
F Parameter	$f_{txframe}$ (txframe_clk frequency)	$f_{rxframe}$ (rxframe_clk frequency)
2	f_{txlink} x (2 / F2_FRAMECLK_DIV)	f_{rxlink} x (2 / F2_FRAMECLK_DIV)
4	f_{txlink}	f_{rxlink}
8	f_{txlink} / 2	f_{rxlink} / 2

Data Bit and Content Mapping Scheme

One major function of the transport layer is to arrange the data bits in a specific way between the Avalon-ST interface and the DLL in the JESD204B IP core.

Figure 5 shows the mapping scheme in the transport layer across various TX to RX interfaces for a specific system configuration.

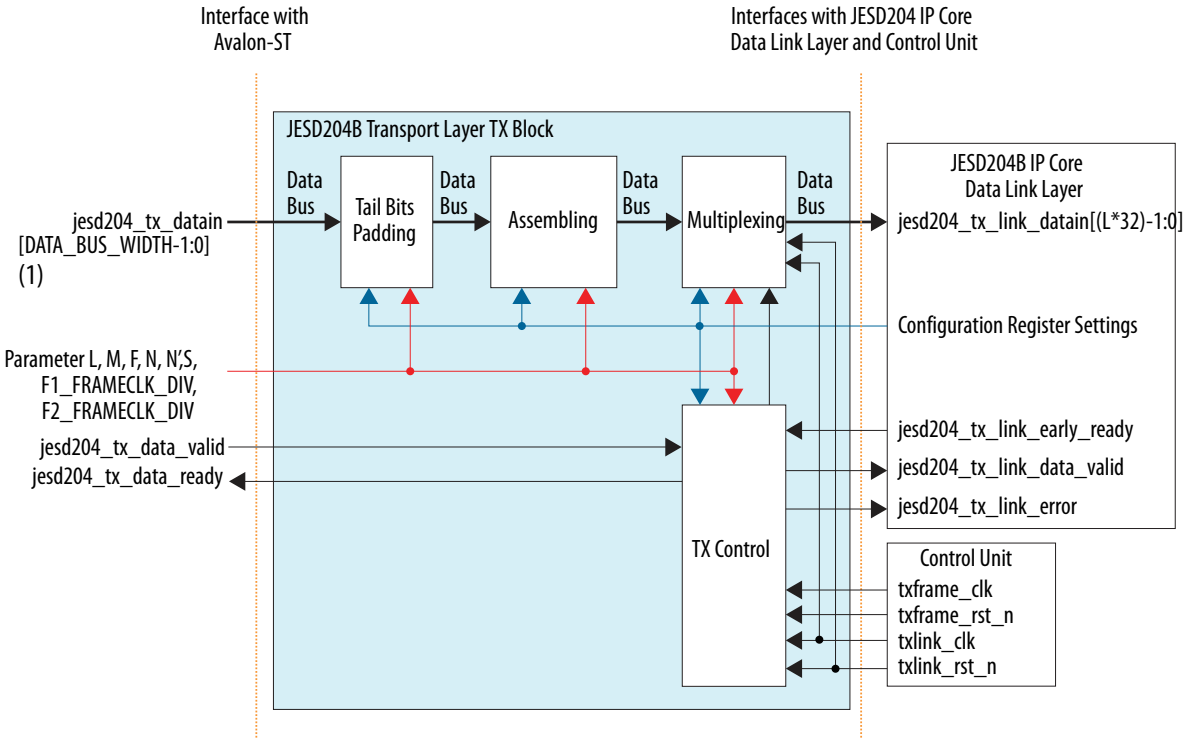
Figure 5: Mapping of Data Bit and Content Across Various Interfaces (LMF = 112, N = 12, N' = 16, S = 1, T represents the tail bits).



TX Path

The assembler in the TX path consists of the tail bits padding, assembling, and multiplexing blocks.

Figure 6: TX Path Assembler Block Diagram



Note:
1. The DATA_BUS_WIDTH value is the data input bus width size, which depends on the F and L parameter.
bus_width=M*S*N
F=(M*S*N_PRIME)/(8*L)
M*S=(8*F*L)/N_PRIME
bus_width=(8*F*L*N)/N_PRIME

- Tail bits padding block—pads incoming data (`jesd204_tx_datain`) with "0" if **N** < 16, so that the padded data is 16 bits per sample.
- Assembling block—arranges the resulting data bits in a specific way according to the mapping scheme (refer to **Figure 5**).
- Multiplexing block—sends the multiplexed data to the DLL interface, determined by certain control signals from the TX control block.

Table 11: Assembler Parameter Settings

Parameter	Description	Value
L	Number of lanes per converter device.	1–8
F	Number of octets per frame.	1, 2, 4, 8
CS	Number of control bits per conversion sample.	0–3
N	Number of conversion bits per converter.	12-16
N'	Number of transmitted bits per sample in the user data format.	16

Parameter	Description	Value
F1_FRAMECLK_DIV	Only applies to cases where F=1. The divider ratio on the `frame_clk`. The assembler always use the post-divided frame_clk (txframe_clk). (6)	1, 4
F2_FRAMECLK_DIV	Only applies to cases where F=2. The divider ratio on the `frame_clk`. The assembler always use the post-divided frame_clk (txframe_clk). (6)	1, 2
RECONFIG_EN	Enable reconfiguration support in the transport layer. Only downscaling reconfiguration is supported. Disable the reconfiguration to reduce the logic.	0, 1
DATA_BUS_WIDTH	The data input bus width size that depends on the F and L. bus_width = M*S*N F = (M*S*N_PRIME)/(8*L) M*S = (8*F*L)/N_PRIME Therefore the data bus width = (8*F*L*N)/N_PRIME	(8*F*L*N)/N_PRIME
CONTROL_BUS_WIDTH	The control input bus width size. The width depends on the CS parameter as well as the M and S parameters. When CS is 0, the control data is one bit wide (tie the signal to 0). If CS = 0, the bus width = 1. Otherwise, the bus width = (DATA_BUS_WIDTH/N*CS) while DATA_BUS_WIDTH/N = M*S	OUTPUT_BUS_WIDTH/N*CS

Table 12: Assembler Signals

Signal	Clock Domain	Direction	Description
Control Unit			
`txlink_clk`	—	Input	TX link clock signal. This clock is equal to the TX data rate divided by 40. This clock is synchronous to the `txframe_clk` signal.
`txframe_clk`	—	Input	TX frame clock used by the transport layer. The frequency is a function of parameters *F*, *F1_FRAMECLK_DIV*, *F2_FRAMECLK_DIV* and *txlink_clk*. This clock is synchronous to the `txlink_clk` signal.

(6) Refer to the **Table 15** to set the desired frame clock frequency with different **FRAMECLK_DIV** and **F** values.

Signal	Clock Domain	Direction	Description
txlink_rst_n	txlink_clk	Input	Reset for the TX link clock domain logic in the assembler. This reset is an active low signal and the deassertion is synchronous to the rising-edge of txlink_clk.
txframe_rst_n	txframe_clk	Input	Reset for the TX frame clock domain logic in the assembler. This reset is an active low signal and the deassertion is synchronous to the rising-edge of txframe_clk.
Signal	**Clock Domain**	**Direction**	**Description**
Between Avalon- ST and Transport Layer			
jesd204_tx_datain[(DATA_BUS_WIDTH)-1:0]	txframe_clk	Input	TX data from the Avalon-ST source interface. The source shall arrange the data in a specific order, as illustrated in the cases listed in **TX Path Data Remapping** section
jesd204_tx_controlin[(CONTROL_BUS_WIDTH)-1:0]	txframe_clk	Input	TX control data from the Avalon-ST source interface. The source shall arrange the data in a specific order, as illustrated in the cases listed in **TX Path Data Remapping** section
jesd204_tx_data_valid	txframe_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
jesd204_tx_data_ready	txframe_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
Signal	**Clock Domain**	**Direction**	**Description**
Between Transport Layer and DLL			

Signal	Clock Domain	Direction	Description
jesd204_tx_link_datain[(L*32)-1:0]	txlink_clk	Output	Indicates transmitted data from the transport layer to the DLL at txlink_clk clock rate, where four octets are packed into a 32-bit data width per lane. The data format is big endian. The table below illustrates the data mapping for L = 4: jesd204_tx_link_datain [x:y] / Lane [31:0] / 0 [63:32] / 1 [95:64] / 2 [127:96] / 3 Connect this signal to the TX DLL jesd204_tx_link_data[] input pin.
jesd204_tx_link_data_valid	txlink_clk	Output	Indicates whether the jesd204_tx_link_datain[] is valid or invalid. • 0—jesd204_tx_link_datain[] is invalid • 1—jesd204_tx_link_datain[] is valid Connect this signal to the TX DLL jesd204_tx_link_valid input pin.
jesd204_tx_link_early_ready (7)	txlink_clk	Input	Indicates that the DLL requires valid data at the subsequent implementation-specific duration. Connect this signal to the TX DLL jesd204_tx_frame_ready output pin.
jesd204_tx_link_error	txlink_clk	Output	Indicates an error at the Avalon-ST source interface. Specifically, this signal is asserted when jesd204_tx_data_valid = "0" while jesd204_tx_data_ready = "1". The DLL subsequently reports this error to the CSR block. Connect this signal to the TX DLL jesd204_tx_frame_error input pin.
Signal	Clock Domain	Direction	Description
CSR in DLL			

(7) If a JESD device of No Multiple-Converter Device Alignment, Single-Lane (NMCDA-SL) class is deployed, Altera recommends that you tie this input signal to "1".

Signal	Clock Domain	Direction	Description
csr_l[4:0] (8)	mgmt_clk	Input	Indicates the number of active lanes for the link. This 5-bit bus represents the **L** value in zero-based binary format. For example, if **L** = 1, the csr_l[4:0] = "00000". This design example supports the following values: • 00000 • 00001 • 00011 • 00111 Any programmed value beyond the supported range may result in undeterminable behavior in the transport layer. You must ensure that the csr_l[4:0] value always matches the system parameter **L** value when it is in static configuration. Runtime reconfiguration supports L fallback. For static configuration, set the maximum L and reconfigure csr_l[] to a smaller value during runtime. This transport layer only supports higher index channels to be powered down. To interleave the de-commision channels, you need to modify the interface connection from the DLL to transport layer. Connect this signal to the TX DLL csr_l[] output pin.
csr_f[7:0] (8)	mgmt_clk	Input	Indicates the number of octets per frame. This 8-bit bus represents the **F** value in zero-based binary format. For example, if **F** = 2, the csr_f[7:0] = "00000001". This design example supports the following values: • 00000000 • 00000001 • 00000011 • 00000111 Any programmed value beyond the supported range may result in undeterminable behavior in the transport layer. Ensure that the csr_f[7:0] value always matches the system parameter **F** value when it is in static configuration. Connect this signal to the TX DLL csr_f[] output pin.

(8) This signal should be static and valid before the deassertion of the link_rst_n and frame_rst_n signals.

Signal	Clock Domain	Direction	Description
csr_n[4:0] (8)	mgmt_clk	Input	Indicates the converter resolution. This 5-bit bus represents the **N** value in zero-based binary format. For example, if **N** = 16, the csr_n[4:0] = "01111". This design example supports the following values: • 01011 • 01100 • 01101 • 01110 • 01111 Any programmed value beyond the supported range may result in undeterminable behavior in the transport layer. You must ensure that the csr_n[4:0] value always match the system parameter **N** value. Connect this signal to the TX DLL csr_n[] output pin.

TX Path Operation

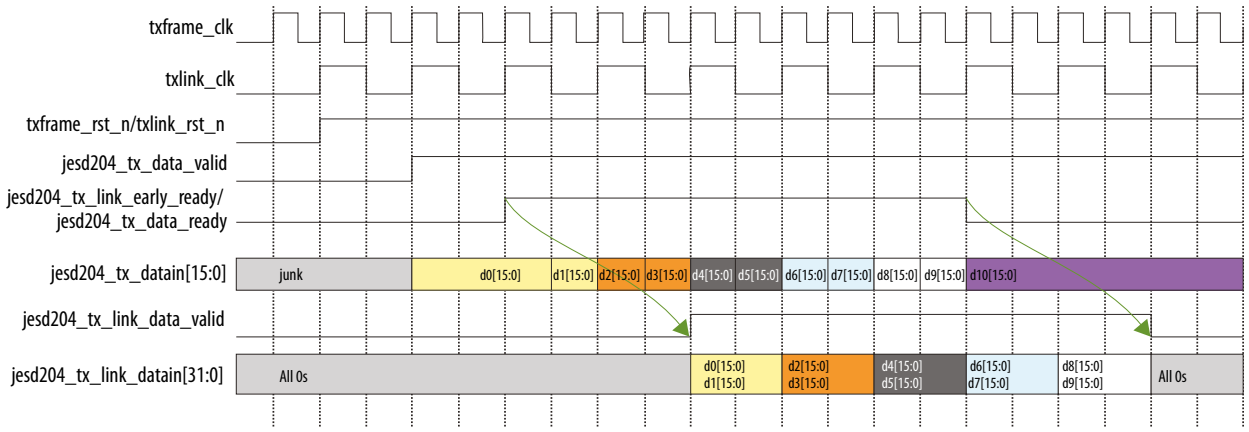
The data transfer protocol between the Avalon-ST interface and the TX path transport layer is data transfer with backpressure, where ready_latency = 0.

Figure 7: TX Operation Behavior

This figure shows the data transmission for a system configuration of **LMF** = 112, **N** = **N'** = 16, **S** = 1.

Operation:

- Upon the deassertion of the `txframe_rst_n` signal, the `jesd204_tx_link_early_ready` signal from the DLL to the transport layer is asserted some time later, which activates the transport layer to start sampling the `jesd204_tx_datain[15:0]` signal from the Avalon-ST interface.
- Each sampled 16-bit data is first written in a FIFO with a depth of four.
- Once the FIFO accumulates 32-bit data, the data is streamed to the DLL accordingly through the `jesd204_tx_link_datain[31:0]` signal.
- Finally, the `jesd204_tx_link_early_ready` and `jesd204_tx_data_ready` signals deassert because the DLL has entered code group synchronization state in this scenario.



TX Data Transmission

This section explains the data transmission behavior when there is a valid TX data out from the TL to DLL.

Upon the deassertion of `txframe_rst_n` signal, the link's `jesd204_tx_link_early_ready` signal equals to "1". This setting activates the TL to start sampling `jesd204_tx_datain` signal from the Avalon-ST interface and transmits sampled data (`jesd204_tx_link_datain`) to the TX link. The TX link only captures valid data from the TL when the `jesd204_tx_link_ready` signal equals to "1" (in user data phase). This means all the data transmitted from the TL before `jesd204_tx_link_ready` signal equals to "1" are ignored.

Figure 8: TX Data Transmission

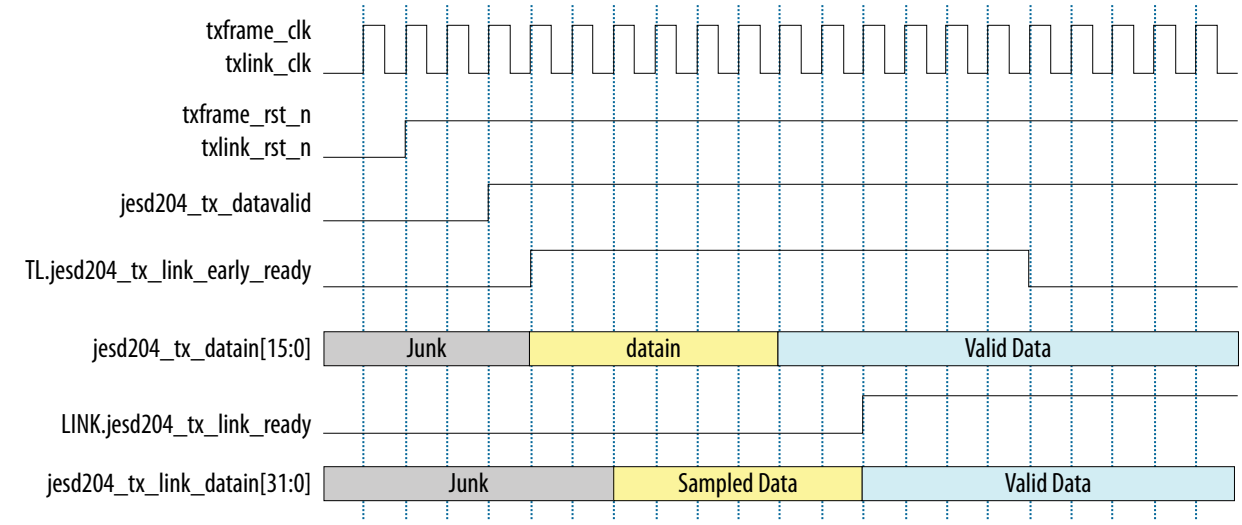
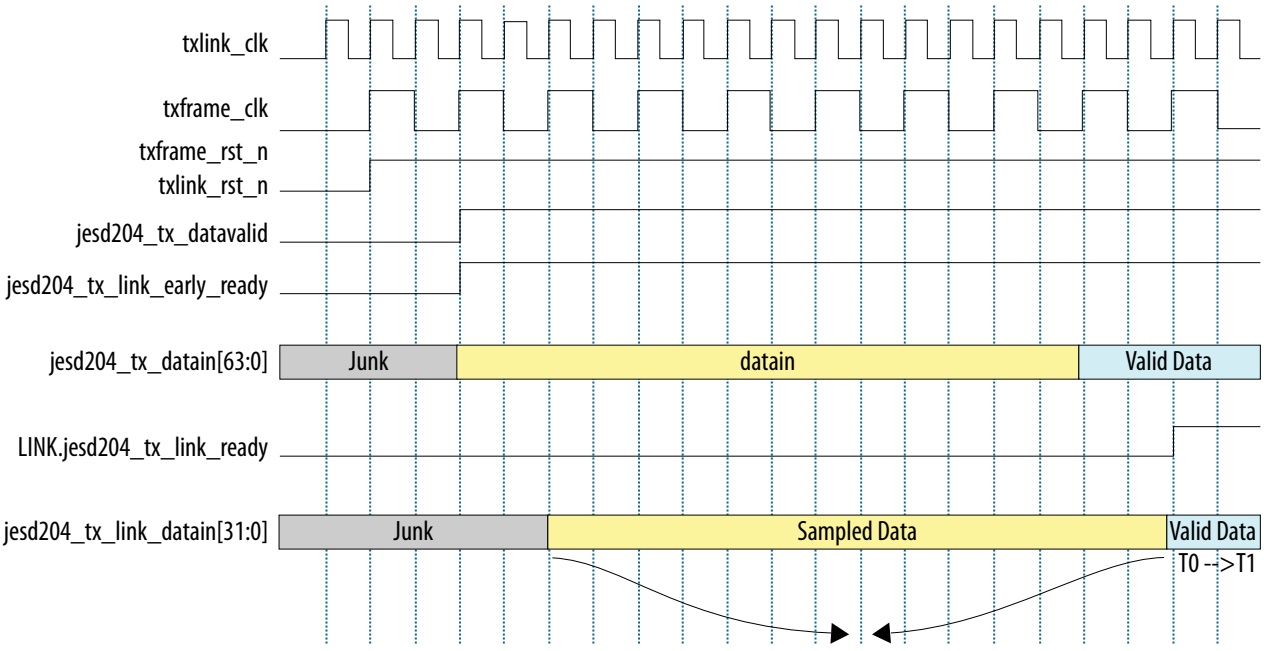


Figure 9: TX Data Transmission (For F = 8)



When F = 8, the data latency for jesd204_tx_link_datain should always be in an even latency link_clk count to ensure that the first valid data captured by the TX link is T0 data followed by T1 data.

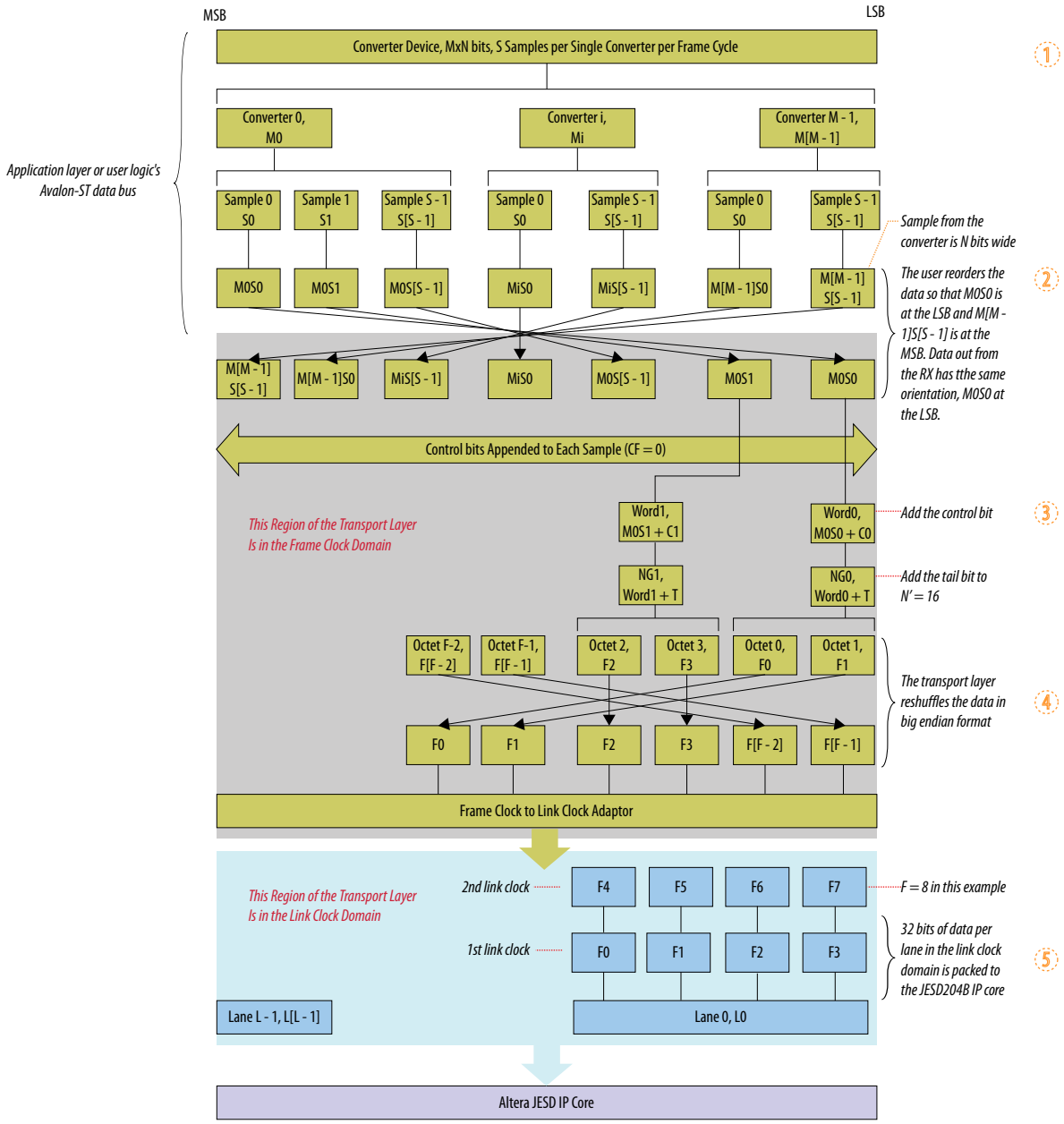
TX Path Data Remapping

The JESD204B IP core implements the data transfer in big endian format.

Figure below illustrates the converter sample to transceiver lane mapping operation in the transport layer. Each converter sample has N bits, M converters per ADC/DAC device, and S samples per converter (M) per frame clock cycle. The transport layer operates at full rate or FRAMECLK_DIV=1.

1. The application layer or user logic data path interfaces directly with the transport layer through the Avalon-ST data bus if the application layer operates in frame clock domain. If the application layer operates at a different clock domain than the frame clock domain, add a FIFO for the clock domain crossing.
2. You have to reorder the samples so that sample 0 of converter 0 is located at LSB of the Avalon-ST data bus, followed by sample 1 of converter 0 (if S>1) or sample 0 of converter 1 (if S=1). The most significant bits (MSB) of the Avalon-ST bus has a sample of S -1 of converter M-1. For example, if S=4 and M=4, the most significant bits will be occupied by sample 3 of converter 3.
3. In this example, there is no control word because CF=0. Control bits are added if CS>1. Depending on the value of CS and N, the number of tail bits added is N'-N-CS. For example, N'=16, N=12 and CS=2, the number of tail bits added to form a nibble group (NG) is 2.
4. The JESD204B IP core implements the data transfer in big endian format. Data is reshuffled in big endian format before crossing to the link clock domain through an adaptor.
5. The data is arranged so that the L0 is always on the right (LSB) in the data bus interfacing with the JESD204B IP core. In big endian implementation, the oldest data (F0) is placed at the MSB in L0. 32-bits or 4 octets of data are transferred to the IP core in one link clock cycle. For example of F=8, 2 link clock cycles are needed to transfer all 8 octets to the IP core.

Figure 10: User Data Format that Feeds into the Transport Layer and Output to the Link Layer



The following tables show examples of data mapping for L=4, F=1, 2, 4, 8 and M*S=2, 4, 8, 16. The configurations that the transport layer support are not limited to these examples.

Table 13: Data Mapping for F=1, L=4

F = 1				
Supported M and S	M*S=2 for F=1, L=4 F=1 supports either (case1: M=1, S=2) or (case2: M=2, S=1)			
F1_FRAMCLK_DIV=1 (9)	1st frameclk	jesd204_tx_datain[31:0] = {F8F12, F0F4}	Case1: M=1, S=2	M0S0=F0F4, M0S1=F8F12
			Case2: M=2, S=1	M0S0=F0F4, M1S0=F8F12
	2nd frameclk	jesd204_tx_datain[31:0] = {F9F13, F1F5}	Case1: M=1, S=2	M0S0=F1F5, M0S1=F9F13
			Case2: M=2, S=1	M0S0=F1F5, M1S0=F9F13
	3rd frameclk	jesd204_tx_datain[31:0] = {F10F114, F2F6}	Case1: M=1, S=2	M0S0=F2F6, M0S1=F10F14
			Case2: M=2, S=1	M0S0=F2F6, M1S0=F10F14
	4th frameclk	jesd204_tx_datain[31:0] = {F11F15, F3F7}	Case1: M=1, S=2	M0S0=F3F7, M0S1=F11F15
			Case2: M=2, S=1	M0S0=F3F7, M1S0=F11F15
F1_FRAMCLK_DIV=4 (10)	jesd204_tx_datain[127:0] = {{F11F15, F3F7},{F10F114, F2F6},{F9F13, F1F5},{F8F12, F0F4}}			
Lane	L3	L2	L1	L0
Data Out	{F12, F13, F14, F15}	{F8, F9, F10, F11}	{F4, F5, F6, F7}	{F0, F1, F2, F3}

Table 14: Data Mapping for F=2, L=4

F = 2	
Supported M and S	M*S=4 for F=2, L=4 F=2 supports either (case1: M=1, S=4), (case2: M=2, S=2) or (case3: M=4, S=1)

(9) The effective frame clock in the Transport Layer is 4x of the link clock.
(10) The effective frame clock in the Transport Layer is same as the link clock.

F = 2				
F2_FRAMCLK_DIV=1	1st frameclk	jesd204_tx_datain[63:0] = {F12F13, F8F9,F4F5, F0F1}	Case1: M=1, S=4	M0S0=F0F1, M0S1=F4F5, M0S2=F8F9, M0S3=F12F13 at
			Case2: M=2, S=2	M0S0=F0F1, M0S1=F4F5, M1S0=F8F9, M1S1=F12F13
			Case3: M=4, S=1	M0S0=F0F1, M1S0=F4F5, M2S0=F8F9, M3S0=F12F13
	2nd frameclk	jesd204_tx_datain[63:0] = {F14F15, F10F11,F6F7, F2F3}	Case1: M=1, S=4	M0S0=F2F3, M0S1=F6F7, M0S2=F10F11, M0S3=F14F15
			Case2: M=2, S=2	M0S0=F2F3, M0S1=F6F7, M1S0=F10F11, M1S1=F14F15
			Case3: M=4, S=1	M0S0=F2F3, M1S0=F6F7, M2S0=F10F11, M3S0=F14F15
F2_FRAMCLK_DIV=2	jesd204_tx_datain[127:0] = {{F14F15, F10F11,F6F7, F2F3}, {F12F13, F8F9,F4F5, F0F1}}			
Lane	L3	L2	L1	L0
Data Out	{F12, F13, F14, F15}	{F8, F9, F10, F11}	{F4, F5, F6, F7}	{F0, F1, F2, F3}

Table 15: Data Mapping for F=4, L=4

F = 4				
Supported M and S	M*S=8 for F=4, L=4 F=4 supports either (case1: M=1, S=8), (case2: M=2, S=4), (case3: M=4, S=2) or (case4: M=8, S=1)			
F=4	jesd204_tx_datain[127:0] = {F14F15,F12F13, F10F11, F8F9,F6F7,F4F5, F2F3,F0F1}	Case1: M=1, S=8	{M0S7, M0S6, M0S5, M0S4, M0S3, M0S2, M0S1, M0S0}	
		Case2: M=2, S=4	{M1S3, M1S2, M1S1, M1S0, M0S3, M0S2, M0S1, M0S0}	
		Case3: M=4, S=2	{M3S1, M3S0, M2S1, M2S0, M1S1, M1S0, M0S1, M0S0}	
		Case4: M=8, S=1	{M7S0, M6S0, M5S0, M4S0, M3S0, M2S0, M1S0, M0S0}	
Lane	L3	L2	L1	L0

F = 4				
Data Out	{F12, F13, F14, F15}	{F8, F9, F10, F11}	{F4, F5, F6, F7}	{F0, F1, F2, F3}

Table 16: Data Mapping for F=8, L=4

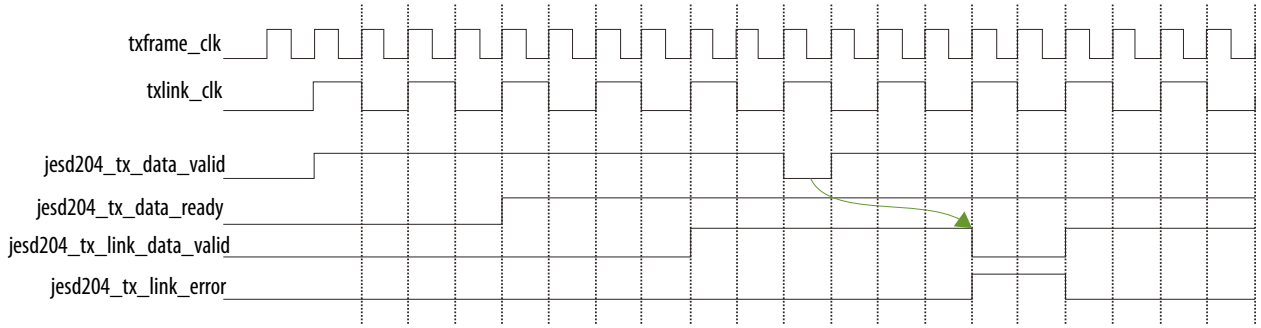
F = 8				
Supported M and S	M*S=16 for F=8, L=4 F=8 supports either (case1: M=1, S=16), (case2: M=2, S=8), (case3: M=4, S=4), (case4: M=8, S=2) or (case5: M=16, S=1)			
F=8	jesd204_tx_datain[255:0] = {{F3031, F28F29,F26F27, F24F25}, {F22F23, F20F21,F18F19, F16F17}, {F14F15, F12F13,F10F11, F8F9}, {F6F7,F4F5, F2F3,F0F1}}	Case1: M=1, S=16	{M0S15, M0S14, M0S13, M0S12, M0S11, M0S10, M0S9, M0S8, M0S7, M0S6, M0S5, M0S4, M0S3, M0S2, M0S1, M0S0}	
Lane	L3	L2	L1	L0
Data Out at linkclk T0	{F24, F25, F26, F27}	{F16, F17, F18, F19}	{F8, F9, F10, F11}	{F0, F1, F2, F3}
Data Out at linkclk T1	{F28, F29, F30, F31}	{F20, F21, F22, F23}	{F12, F13, F14, F15}	{F4, F5, F6, F7}

TX Error Reporting

For TX path error reporting, the transport layer expects a valid stream of TX data from the Avalon-ST interface (indicated by `jesd204_tx_data_valid` signal = 1) as long as the `jesd204_tx_data_ready` remains asserted. If the `jesd204_tx_data_valid` signal unexpectedly deasserts during this stage, the transport layer reports an error to the DLL by asserting the `jesd204_tx_link_error` signal and deasserting the `jesd204_tx_link_data_valid` signal accordingly, as shown in the timing diagram below.

Figure 11: TX Error Reporting

The `jesd204_tx_data_valid` signal deasserts for one frame_clk and cannot be sampled by the link_clk.



TX Latency

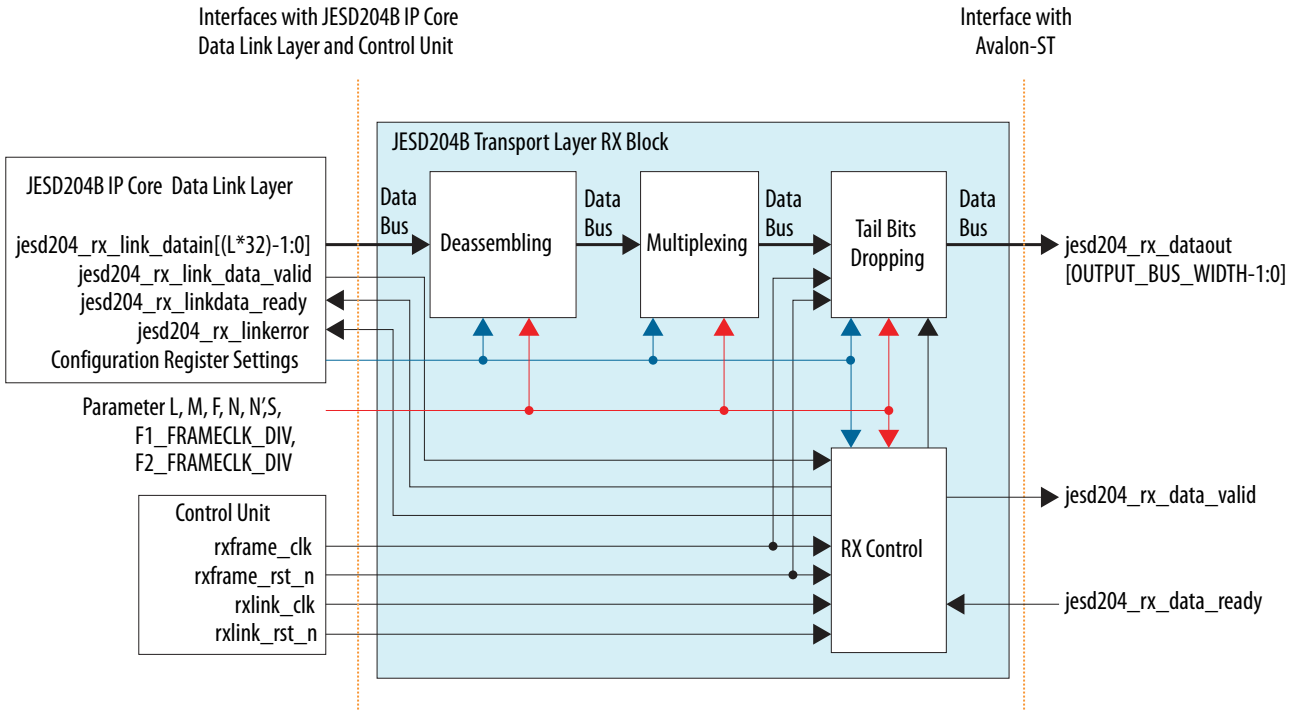
Table 17: TX Latency Associated with Different F and FRAMECLK_DIV Settings.

F	FRAMECLK_DIV	TX Latency
1	1	3 `txframe_clk` period. • Maximum 5 `txframe_clk` period for byte 3 • Minimum 2 `txframe_clk` period for byte 0
1	4	1 `txframe_clk` period
2	1	3 `txframe_clk` period. • Maximum 4 `txframe_clk` period for byte 2 and byte 3 • Minimum 3 `txframe_clk` period for byte 0 and byte 1
2	2	1 `txframe_clk` period
4	—	1 `txframe_clk` period
8	—	1 `txframe_clk` period

RX Path

The deassembler in the RX path consists of the tail bits dropping, deassembling, and multiplexing blocks.

Figure 12: RX Path Assembler Block Diagram



- Tail bit dropping block—drops padded tail bits in the incoming data (`jesd204_rx_link_datain`).
- Deassembling block—rearranges the resulting data bits in a specific way according to the mapping scheme (refer to **Figure 5**).
- Multiplexing block—sends the multiplexed data to the Avalon-ST interface, determined by certain control signals from the RX control block.

Table 18: Deassembler Parameter Settings

Parameter	Description	Value
L	Number of lanes per converter device.	1–8
F	Number of octets per frame.	1, 2, 4, 8
CS	Number of control bits per conversion sample.	0–3
N	Number of conversion bits per converter.	12-16
N'	Number of transmitted bits per sample in the user data format.	16
F1_FRAMECLK_DIV	Only applies to cases where F=1. The divider ratio on the `frame_clk`. The deassembler always uses the post-divided frame_clk (rxframe_clk). (11)	1, 4

(11) Refer to the **Table 15** to set the desired frame clock frequency with different **FRAMECLK_DIV** and **F** parameter values.

Parameter	Description	Value
F2_FRAMECLK_DIV	Only applies to cases where F=2. The divider ratio on the `frame_clk`. The deassembler always uses the post-divided frame_clk (rxframe_clk). (11)	1, 2
RECONFIG_EN	Enable reconfiguration support in the transport layer. Only downscaling reconfiguration is supported. Disable the reconfiguration to reduce the logic.	0, 1
OUTPUT_BUS_WIDTH	The data output bus width size that depends on the F and L. bus_width = M*S*N F = (M*S*N_PRIME)/(8*L) M*S = (8*F*L)/N_PRIME Therefore the output bus width = (8*F*L*N)/N_PRIME	(8*F*L*N)/N_PRIME
CONTROL_BUS_WIDTH	The control output bus width size. The width depends on the CS parameter as well as the M and S parameters. When CS is 0, the control data is one bit wide (tie the signal to 0). If CS = 0, the bus width = 1. Otherwise, the bus width = (OUTPUT_BUS_WIDTH/N*CS) while OUTPUT_BUS_WIDTH/N = M*S	OUTPUT_BUS_WIDTH/N*CS

Table 19: Deassembler Signals

Signal	Clock Domain	Direction	Description
Control Unit			
`rxlink_clk`	—	Input	RX link clock signal. This clock is equal to the RX data rate divided by 40. This clock is synchronous to the `rxframe_clk` signal.
`rxframe_clk`	—	Input	RX frame clock used by the deassembler. The frequency is a function of parameters *F*, *F1_FRAMECLK_DIV*, *F2_FRAMECLK_DIV* and *rxlink_clk*. This clock is synchronous to the `rxlink_clk` signal.
`rxlink_rst_n`	`rxlink_clk`	Input	Reset for the RX link clock domain logic in the deassembler. This reset is an active low signal and the deassertion is synchronous to the rising-edge of `rxlink_clk`.

Signal	Clock Domain	Direction	Description
rxframe_rst_n	rxframe_clk	Input	Reset for the RX frame clock domain logic in the deassembler. This reset is an active low signal and the deassertion is synchronous to the rising-edge of rxframe_clk.
Signal	**Clock Domain**	**Direction**	**Description**
Between Avalon- ST and Transport Layer			
jesd204_rx_dataout[(OUTPUT_BUS_WIDTH)-1:0]	rxframe_clk	Output	RX data to the Avalon-ST source interface. The transport layer arranges the data in a specific order, as illustrated in the cases listed in **RX Path Data Remapping** section.
jesd204_rx_controlout[CONTROL_BUS_WIDTH -1:0]	rxframe_clk	Output	RX control data to the Avalon-ST source interface. The transport layer arranges the data in a specific order, as illustrated in the cases listed in **RX Path Data Remapping** section.
jesd204_rx_data_valid	rxframe_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
jesd204_rx_data_ready	rxframe_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
Signal	**Clock Domain**	**Direction**	**Description**
Between Transport Layer and DLL			

<table>
<tr><th>Signal</th><th>Clock Domain</th><th>Direction</th><th>Description</th></tr>
<tr><td>jesd204_rx_link_datain[(L*32)-1:0]</td><td>rxlink_clk</td><td>Input</td><td>Indicates received data from the DLL to the transport layer, where four octets are packed into a 32-bit data width per lane. The data format is big endian. The table below illustrates the data mapping for L = 4:
<table><tr><th>jesd204_rx_link_datain [x:y]</th><th>Lane</th></tr><tr><td>[31:0]</td><td>0</td></tr><tr><td>[63:32]</td><td>1</td></tr><tr><td>[95:64]</td><td>2</td></tr><tr><td>[127:96]</td><td>3</td></tr></table>
Connect this signal to the RX DLL jesd204_rx_link_data[] output pin.</td></tr>
<tr><td>jesd204_rx_link_data_valid</td><td>rxlink_clk</td><td>Input</td><td>Indicates whether the jesd204_rx_link_datain[] is valid or invalid.
• 0—jesd204_rx_link_datain[] is invalid
• 1—jesd204_rx_link_datain[] is valid
Connect this signal to the RX DLL jesd204_rx_link_valid output pin.</td></tr>
<tr><td>jesd204_rx_link_data_ready</td><td>rxframe_clk</td><td>Output</td><td>Indicates that the transport layer is ready to sample jesd204_rx_link_datain[].
• 0—transport layer is not ready to sample jesd204_rx_link_datain[]
• 1—transport layer starts sampling jesd204_rx_link_datain[] at the next clock cycle.
Connect this signal to the RX DLL jesd204_rx_link_ready input pin.</td></tr>
<tr><td>jesd204_rx_link_error</td><td>rxlink_clk</td><td>Output</td><td>Indicates an empty data stream due to invalid data. This signal is asserted high to indicate an error at the Avalon-ST sink interface (for example, when jesd204_rx_data_valid = "1" while jesd204_rx_data_ready = "0"). The DLL subsequently reports this error to the CSR block.
Connect this signal to the RX DLL jesd204_rx_frame_error input pin.</td></tr>
</table>

Signal	Clock Domain	Direction	Description
CSR in DLL			
csr_l[4:0] (12)	mgmt_clk	Input	Indicates the number of active lanes for the link. This 5-bit bus represents the **L** value in zero-based binary format. For example, if **L** = 1, the csr_l[4:0] = "00000". This design example supports the following values: • 00000 • 00001 • 00011 • 00111 Any programmed value beyond the supported range may result in undeterminable behavior in the transport layer. You must ensure that the csr_l[4:0] value always match the system parameter **L** value. Runtime reconfiguration supports L fallback. For static configuration, set the maximum L and reconfigure csr_l[] to a smaller value during runtime. This transport layer only supports higher index channels to be powered down. To interleave the de-commision channels, you need to modify the interface connection from the DLL to transport layer. Connect this signal to the RX DLLcsr_l[] output pin.

(12) This signal should be static and valid before the deassertion of the link_rst_n and frame_rst_n signals.

Signal	Clock Domain	Direction	Description
csr_f[7:0] (12)	mgmt_clk	Input	Indicates the number of octets per frame. This 8-bit bus represents the **F** value in zero-based binary format. For example, if **F** = 2, the csr_f[7:0] = "00000001". This design example supports the following values: • 00000000 • 00000001 • 00000011 • 00000111 Any programmed value beyond the supported range may result in undeterminable behavior in the transport layer. You must ensure that the csr_f[7:0] value always match the system parameter **F** value. Connect this signal to the RX DLL csr_f[] output pin.
csr_n[4:0] (12)	mgmt_clk	Input	Indicates the converter resolution. This 5-bit bus represents the **N** value in zero-based binary format. For example, if **N** = 16, the csr_n[4:0] = "01111". This design example supports the following values: • 01011 • 01100 • 01101 • 01110 • 01111 Any programmed value beyond the supported range may result in undeterminable behavior in the transport layer. You must ensure that the csr_n[4:0] value always match the system parameter **N** value. Connect this signal to the RX DLL csr_n[] output pin.

RX Path Operation

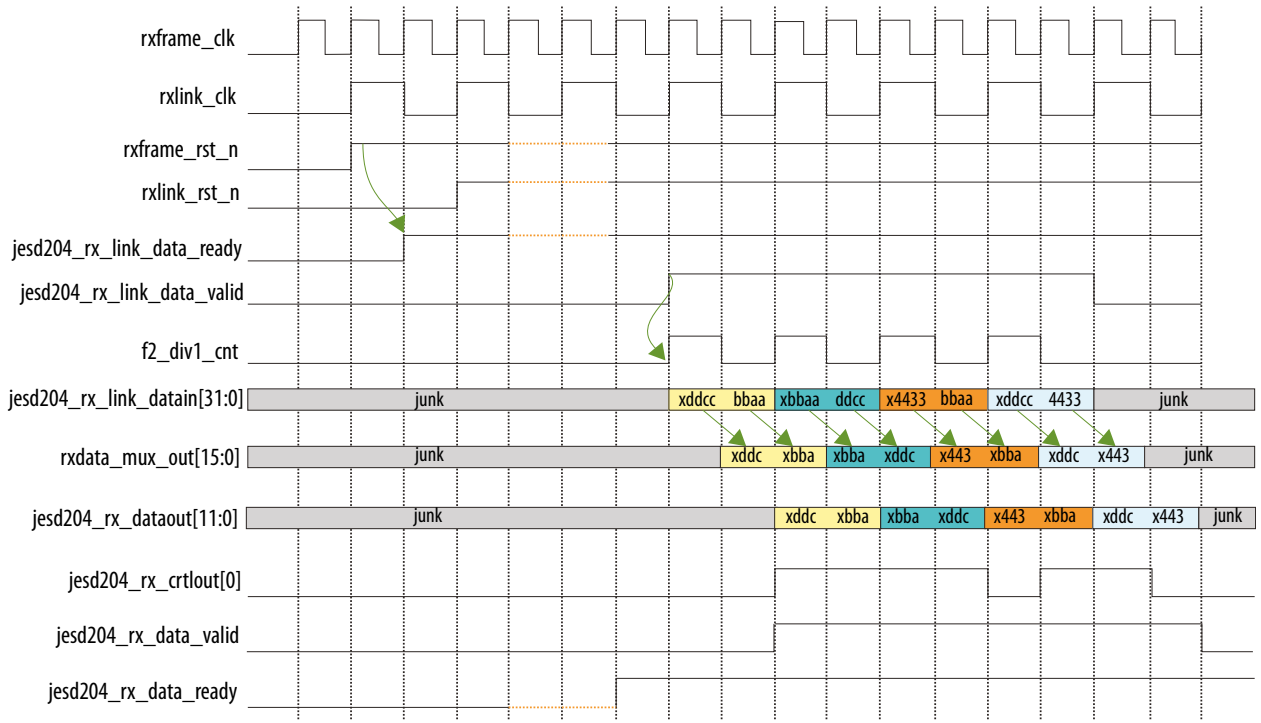
The data transfer protocol between the Avalon-ST interface and the RX path transport layer is data transfer without backpressure. Therefore, the sink shall always be ready to sample the incoming data whenever data at the source is valid.

Figure 13: RX Operation Behavior

This figure shows the data transmission for a system configuration of **LMF** = 112, **N** = 12, **N'** = 16, **S** =1.

Operation:

- Upon the deassertion of the `rxframe_rst_n` signal, the `jesd204_rx_link_data_ready` signal from the deassembler to the DLL is asserted at the next `rxframe_clk`.
- Subsequently, the DLL asserts the `jesd204_rx_link_data_valid` signal for the deassembler to activate the `f2_div1_cnt` signal logic and to start sampling the `jesd204_rx_link_datain[31:0]` signal. (13)
- At the following `rxframe_clk`, the `jesd204_rx_data_valid` is asserted along with the multiplexed `jesd204_rx_dataout[11:0]` signal to stream data to the Avalon-ST interface.
- Finally, the DLL deasserts the `jesd204_rx_link_data_valid` signal when there is no more valid data.
- The deassembler deactivates the `f2_div1_cnt` signal logic accordingly, and deasserts the `jesd204_rx_data_valid` at the next `rxframe_clk`.



RX Data Reception

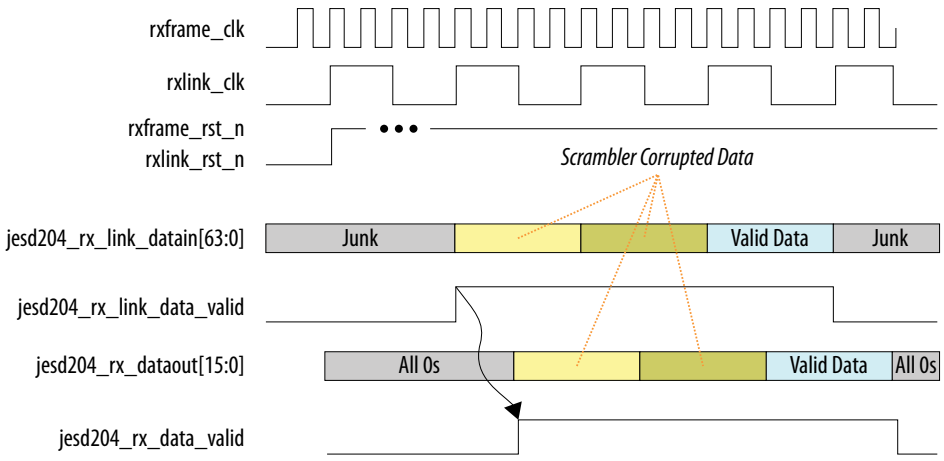
This section explains when there is a valid RX data out from the DLL to the TL to with scrambler enabled.

The MAC layer process the `jesd204_rx_dataout` signal once the TL asserts the `jesd204_rx_data_valid` signal. However, there are some data that should be discarded by the upper layer when the you enable the scrambler. This is because the initial unknown seed value within the scrambler can corrupt the very first eight octets, which is the data for the first two link clock cycles. The data can be translated to the frame

(13) The `f2_div1_cnt` signal is internally generated in the RX control block to correctly stream data to the Avalon-ST interface.

clock cycle depending on the *F* and *FRAMECLK_DIV* parameters selected based on the frame clock to link clock relationship.

Figure 14: RX Data Reception



Related Information

Relationship Between Frame Clock and Link Clock on page 21

RX Path Data Remapping

The JESD204B IP core implements the data transfer in big endian format.

The RX path data remapping is the reverse of TX path data remapping. Refer to **Figure 10**for the RX transport layer remapping operation.

The following tables show examples of data mapping for L=4, F=1, 2, 4, 8 and M*S=2, 4, 8, 16. The configurations that the transport layer support are not limited to these examples.

Table 20: Data Mapping for F=1, L=4

F = 1				
Lane	L3	L2	L1	L0
Data In	{F12, F13, F14, F15}	{F8, F9, F10, F11}	{F4, F5, F6, F7}	{F0, F1, F2, F3}
Supported M and S	M*S=2 for F=1, L=4 F=1 supports either (case1: M=1, S=2) or (case2: M=2, S=1) Assuming N=16, M0S0=jesd204_rx_dataout[15:0], M0S1/M1S0= jesd204_rx_dataout[31:16]			

F = 1				
F1_FRAMCLK_DIV=1	1st frameclk	cnt=0 : jesd204_rx_dataout[31:0] = {F8F12, F0F4}	Case1: M=1, S=2	M0S0=F0F4, M0S1=F8F12
			Case2: M=2, S=1	M0S0=F0F4, M1S0=F8F12
	2nd frameclk	cnt=1: jesd204_rx_dataout[31:0] = {F9F13, F1F5}	Case1: M=1, S=2	M0S0=F1F5, M0S1=F9F13
			Case2: M=2, S=1	M0S0=F1F5, M1S0=F9F13
	3rd frameclk	cnt=2: jesd204_rx_dataout[31:0] = {F10F114, F2F6}	Case1: M=1, S=2	M0S0=F2F6, M0S1=F10F14
			Case2: M=2, S=1	M0S0=F2F6, M1S0=F10F14
	4th frameclk	cnt=3: jesd204_rx_dataout[31:0] = {F11F15, F3F7}	Case1: M=1, S=2	M0S0=F3F7, M0S1=F11F15
			Case2: M=2, S=1	M0S0=F3F7, M1S0=F11F15
F1_FRAMCLK_DIV=4	jesd204_rx_dataout[127:0] = {{F11F15, F3F7},{F10F114, F2F6},{F9F13, F1F5},{F8F12, F0F4}}			

Table 21: Data Mapping for F=2, L=4

F = 2				
Lane	L3	L2	L1	L0
Data In	{F12, F13, F14, F15}	{F8, F9, F10, F11}	{F4, F5, F6, F7}	{F0, F1, F2, F3}
Supported M and S	M*S=4 for F=2, L=4 F=2 supports either (case1: M=1, S=4), (case2: M=2, S=2) or (case3: M=4, S=1)			

F = 2				
F2_FRAMCLK_DIV=1	1st frameclk	cnt=0: jesd204_rx_dataout[63:0] = {F12F13, F8F9,F4F5, F0F1}	Case1: M=1, S=4	M0S0=F0F1, M0S1=F4F5, M0S2=F8F9, M0S3=F12F13
			Case2: M=2, S=2	M0S0=F0F1, M0S1=F4F5, M1S0=F8F9, M1S1=F12F13
			Case3: M=4, S=1	M0S0=F0F1, M1S0=F4F5, M2S0=F8F9, M3S0=F12F13
	2nd frameclk	cnt=1: jesd204_rx_dataout[63:0] = {F14F15, F10F11,F6F7, F2F3}	Case1: M=1, S=4	M0S0=F2F3, M0S1=F6F7, M0S2=F10F11, M0S3=F14F15
			Case2: M=2, S=2	M0S0=F2F3, M0S1=F6F7, M1S0=F10F11, M1S1=F14F15
			Case3: M=4, S=1	M0S0=F2F3, M1S0=F6F7, M2S0=F10F11, M3S0=F14F15
F2_FRAMCLK_DIV=2	jesd204_rx_dataout[127:0] = {{F14F15, F10F11,F6F7, F2F3}, {F12F13, F8F9,F4F5, F0F1}}			

Table 22: Data Mapping for F=4, L=4

F = 4				
Lane	L3	L2	L1	L0
Data In	{F12, F13, F14, F15}	{F8, F9, F10, F11}	{F4, F5, F6, F7}	{F0, F1, F2, F3}
Supported M and S	M*S=8 for F=4, L=4 F=4 supports either (case1: M=1, S=8), (case2: M=2, S=4), (case3: M=4, S=2) or (case4: M=8, S=1)			
F=4	jesd204_rx_dataout[127:0] = {F14F15, F12F13,F10F11, F8F9,F6F7,F4F5, F2F3,F0F1}	Case1: M=1, S=8	{M0S7, M0S6, M0S5, M0S4, M0S3, M0S2, M0S1, M0S0}	
		Case2: M=2, S=4	{M1S3, M1S2, M1S1, M1S0, M0S3, M0S2, M0S1, M0S0}	
		Case3: M=4, S=2	{M3S1, M3S0, M2S1, M2S0, M1S1, M1S0, M0S1, M0S0}	
		Case4: M=8, S=1	{M7S0, M6S0, M5S0, M4S0, M3S0, M2S0, M1S0, M0S0}	

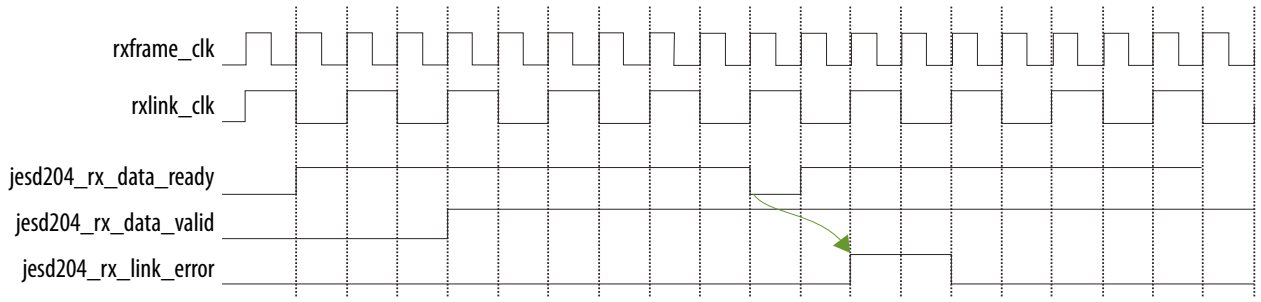
Table 23: Data Mapping for F=8, L=4

F = 8				
Lane	L3	L2	L1	L0
Data In linkclk T0	{F24, F25, F26, F27}	{F16, F17, F18, F19}	{F8, F9, F10, F11}	{F0, F1, F2, F3}
Data In linkclk T1	{F28, F29, F30, F31}	{F20, F21, F22, F23}	{F12, F13, F14, F15}	{F4, F5, F6, F7}
Supported M and S	M*S=16 for F=8, L=4 F=8 supports either (case1: M=1, S=16), (case2: M=2, S=8), (case3: M=4, S=4), (case4: M=8, S=2) or (case5: M=16, S=1)			
F=8	jesd204_rx_dataout[255:0] = {{F3031, F28F29,F26F27, F24F25},{F22F23, F20F21,F18F19, F16F17},{F14F15, F12F13, F10F11,F8F9}, {F6F7,F4F5, F2F3,F0F1}}	Case1: M=1, S=16	{M0S15, M0S14, M0S13, M0S12, M0S11, M0S10, M0S9, M0S8, M0S7, M0S6, M0S5, M0S4, M0S3, M0S2, M0S1, M0S0}	

RX Error Reporting

For RX path error reporting, the transport layer expects the AL to always be ready to sample the RX data (as indicated by the `jesd204_rx_data_ready` signal equal to "1") as long as the `jesd204_rx_data_valid` remains asserted. If the `jesd204_rx_data_ready` signal unexpectedly deasserts, the transport layer reports the error to the DLL by asserting the `jesd204_rx_link_error` signal, as shown in the timing diagram below.

Figure 15: RX Error Reporting



RX Latency

The RX latency is defined as the time needed to fully transfer a 32-bit data in a lane (`jesd204_rx_link_datain*`) to the Avalon-ST interface (`jesd204_rx_dataout*`) when the `jesd204_rx_link_data_valid` signal equals to "1".

Table 24: RX Latency Associated with Different F and FRAMECLK_DIV Settings.

F	FRAMECLK_DIV	RX Latency
1	1	• Maximum 5 `rxframe_clk` period for byte 3 • Minimum 2 `rxframe_clk` period for byte 0
1	4	2 `rxframe_clk` period
2	1	• Maximum 3 `rxframe_clk` period for byte 2 and byte 3 • Minimum 2 `rxframe_clk` period for byte 0 and byte 1
2	2	2 `txframe_clk` period
4	—	2 `txframe_clk` period
8	—	2 `txframe_clk` period

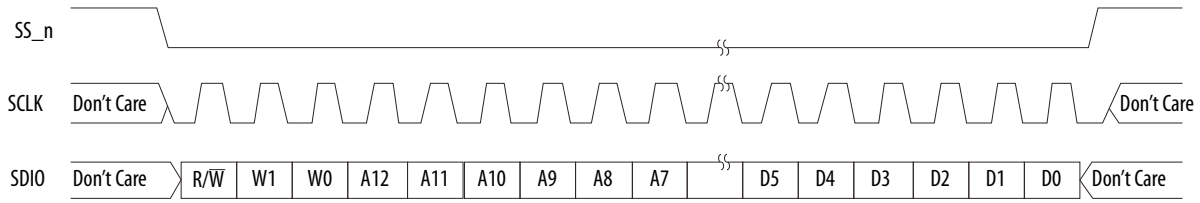
Serial Port Interface (SPI)

An external converter device with a SPI allows you to configure the converter for specific functions or operations through a structured register space provided inside the converter device. The SPI gives flexibility and customization, depending on the application. Addresses are accessed via the serial port and can be written to or read from the port. The memory is organized into bytes that can be further divided into fields.

The SPI communicates using two data lines, a control line, and a synchronization clock. A single SPI master can work with multiple slaves. The SPI core logic is synchronous to the clock input provided by the Avalon-MM interface. When configured as a master, the core divides the Avalon-MM clock to generate the `SCLK` output.

Figure 16: Serial Port Interface (24-bit) Timing Diagram

Figure shows the timing diagram of a 24-bit SPI transaction required by a typical external converter device.

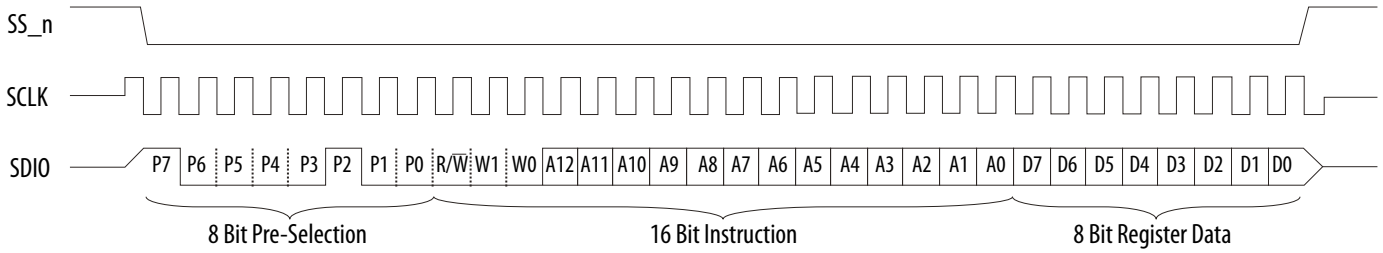


The first 16 bits are instruction data. The first bit in the stream is the read or write indicator bit. This bit goes high to indicate a read request. W1 and W0 represent the number of data bytes to transfer for either a

read or write process. For implementation simplicity, W1 and W0 are always set at 0 in this design example. The subsequent 13 bits represent the starting address of the data sent. The last 8 bits are register data.

For a 32-bit SPI transaction, each SPI programming cycle needs to be preceded with a preselection byte. The preselection byte is typically used to forward the SPI command to the right destination. **figure** shows the timing diagram of a 32-bit SPI transaction.

Figure 17: Serial Port Interface (32-bit) Timing Diagram



In this design example, the SPI core is configured as a 4-wire master protocol to control three independent SPI slaves—ADC, DAC, and clock devices. The width of the receive and transmit registers are configured at 32 bits. Data is sent in MSB-first mode in compliance with the converter device default power up mode. The SPI clock (`sclk`) rate is configured at a frequency of the SPI input clock rate divided by 5. If the SPI input clock rate is 100 MHz (in the `mgmt_clock` domain), the `sclk` rate is 20 MHz. If the external converter device's SPI interface is a 3-wire protocol without both MOSI (master output, slave input) and MISO (master input, slave output) lines but with a single DATAIO pin, you can use the ALTIOBUF Megafunction IP core (configured with bidirectional buffer) with the SPI master to convert the MOSI and MISO lines to a single DATAIO pin. The DATAIO pin can be dynamically reconfigured as MOSI by asserting the output enable (`oe`) signal or as MISO by deasserting the `oe` signal. For implementation simplicity, you can directly connect the master MOSI pin to the slave DATAIO pin if read transactions are not required.

Related Information

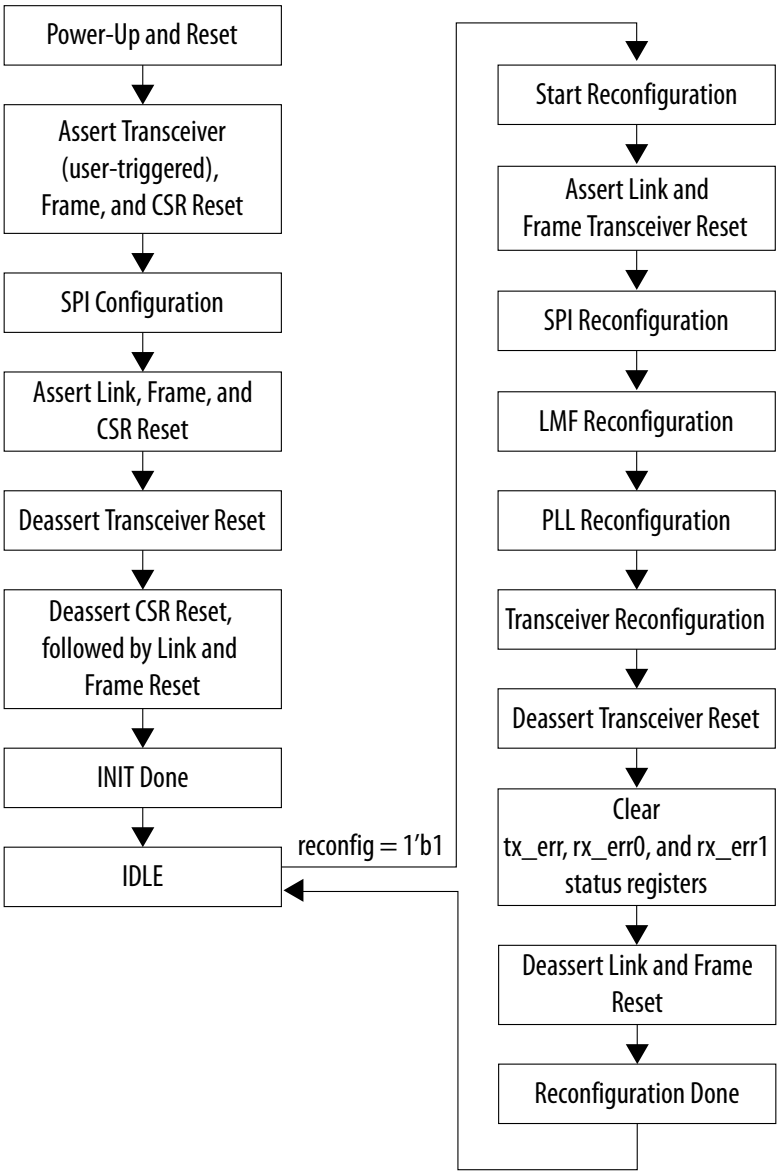
I/O Buffer (ALTIOBUF) Megafunction User Guide
More information about configuring the ALTIOBUF Megafunction IP Core.

Control Unit

The control unit has access to the CSR interface of the JESD204B IP core duplex base core, PLL reconfiguration, transceiver reconfiguration controller, and SPI master. The control unit also serves as a clock and reset unit (CRU) for the design example.

The control unit replaces the software-based Nios II processor to perform device configuration and initialization on the JESD204B duplex base core. This configuration and initialization process includes the transceivers, transport layer, pattern generator and checker, external converters (ADC/DAC), and clock devices over the SPI interface.

Figure 18: Control Unit Process Flow



Memory Block (ROM)

The control unit is a finite state machine (FSM) that works with multiple memory blocks (ROMs).

Each ROM holds the configuration data required to configure the external converter or clock devices for each SPI slave. A memory initialization file (MIF) contains the initial values for each address in the memory. Each memory block requires a separate file. You can create the MIF using the text editor tool in the Quartus Prime software.

Figure 19: Example of MIF Format and Content

```
-- MIF content for ADC

WIDTH=24;        -- the size of data in bits
DEPTH=8;         -- the size of memory in words
ADDRESS_RADIX=UNS;  -- the radix for address values
DATA_RADIX=BIN;    -- the radix for data values

CONTENT BEGIN
  0  :  000000000101111100010101;  -- write 0x15 to link control 1 register 0x5F to disable the lane
  1  :  000000000101111001000100;  -- write 0x44 to quick config register  0x5E for L=4, M=4
  2  :  000000000110010011000000;  -- write 0xC0 to DID register 0x64
  3  :  000000000110111000000011;  -- write 0x03 to parameter SCR/L register 0x6E to disable scrambler
  4  :  000000000111000000001111;  -- write 0x0F to parameter K register 0x70 for K=16 in base IP core
  5  :  000000000000110100000100;  -- write 0x04 to test mode register 0x0D for checkerboard test pattern
  6  :  000000000101111100010100;  -- write 0x14 to link control 1 register 0x5F to enable the lane
  7  :  111111111111111111111111;  -- indicates end of mif or end of programming sequence
END;
```

The initial values for each address and sequence is defined based on the requirement of the external converter and clock devices. The example above is based on 24-bit SPI write-only programming.

The last word must not be a valid data and must be set to all 1's to indicate the end of the MIF or programming sequence. This is because each converter device may have a different number of programmable registers and hence involves a different number of MIF words. In this design example, three ROMs are used by default for each external ADC, DAC, and clock devices. If either one of the device is not used, a single word MIF with all 1's can be created.

Note: The MIFs in this design example is an example for a particular converter device. You must define the MIF content based on the requirement of the external converter devices.

Finite State Machine (FSM)

The steps below describe the FSM flow:

1. Initialize the SPI:
   a. Perform a read transaction from the ROM on per word basis and write to the SPI master for SPI write transaction to the external SPI slave.
   b. Perform a read transaction from the next ROM and perform the same SPI write transaction to next SPI slave.
2. Initialize the JESD204B IP base core, transport layer, pattern generator, and pattern checker upon successful initialization of the transceiver.

System Parameters

Table 25: System Parameter Settings

This table lists the parameters exposed at the system level.

Parameter	Value [14]	Default	Description
LINK	1, 2	1	Number of JESD204B link. One link represent one JESD204B instance.
L	1, 2, 4, 8	2	Number of lanes per converter device.
M	1, 2, 4, 8	2	Number of converters per device.
F	1, 2, 4, 8	2	Number of octets per frame.
S	1, 2	1	Number of transmitted samples per converter per frame.
N	12–16	16	Number of conversion bits per converter.
N'	16	16	Number of transmitted bits per sample in the user data format.
F1_FRAMECLK_DIV	1, 4	4	The divider ratio on `frame_clk` when F = 1. The transport layer uses the post-divided `frame_clk`.
F2_FRAMECLK_DIV	1, 2	2	The divider ratio on `frame_clk` when F = 2. The transport layer uses the post-divided `frame_clk`.
POLYNOMIAL_LENGTH	7, 9, 15, 23, 31	7	Defines the polynomial length for the PRBS pattern generator and checker, which is also the equivalent number of stages for the shift register. • If PRBS-7 is required, set this parameter to 7. • If PRBS-9 is required, set this parameter to 9. • If PRBS-15 is required, set this parameter to 15. • If PRBS-23 is required, set this parameter to 23. • If PRBS-31 is required, set this parameter to 31. This parameter value must not be larger than N, which is the output data width of the PRBS pattern generator or converter resolution. If an N of 12-14 is required, PRBS-7 and PRBS-9 are the only feasible options. If an N of 15-16 is required, PRBS-7, PRBS-9, and PRBS-15 are the only feasible options.
FEEDBACK_TAP	6, 5, 14, 18, 28	6	Defines the feedback tap for the PRBS pattern generator and checker. This is an intermediate stage that is XOR-ed with the last stage to generate to next PRBS bit. • If PRBS-7 is required, set this parameter to 6. • If PRBS-9 is required, set this parameter to 5. • If PRBS-15 is required, set this parameter to 14. • If PRBS-23 is required, set this parameter to 18. • If PRBS-31 is required, set this parameter to 28.

Table below lists the configuration that this design example supports. However, the design example generated by the Qsys system is always fixed at a data rate of 6144 Mbps and a limited set of configuration as shown in the table below. If your setting in the Qsys parameter editor does not match one of the LMF

[14] Values supported or demonstrated by this design example.

and bonded mode parameter values in **Table**, the design example is generated with the default values of LMF = 124.

Table 26: Static and Dynamic Reconfiguration Parameter Values Supported

Mode	Link	L	M	F	Reference Clock	Frame Clock	Link Clock	F1_ FRAMECLK_ DIV	F2_ FRAMECLK_ DIV
Static									
Bonded/Non-bonded	1	1	1	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	1	1	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	1	2	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	1	2	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	1	4	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	2	1	1	153.6	153.6	153.6	4	
Bonded/Non-bonded	1	2	1	2	153.6	153.6	153.6		2
Bonded/Non-bonded	1	2	1	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	2	1	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	2	2	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	2	2	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	2	4	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	2	4	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	2	8	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	4	1	1	153.6	153.6	153.6	4	
Bonded/Non-bonded	1	4	1	2	153.6	153.6	153.6		2
Bonded/Non-bonded	1	4	1	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	4	2	1	153.6	153.6	153.6	4	
Bonded/Non-bonded	1	4	2	2	153.6	153.6	153.6		2
Bonded/Non-bonded	1	4	2	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	4	2	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	4	4	2	153.6	153.6	153.6		2
Bonded/Non-bonded	1	4	4	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	4	4	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	4	8	4	153.6	153.6	153.6	1	
Bonded/Non-bonded	1	4	8	8	153.6	76.8	153.6	1	

Mode	Link	L	M	F	Reference Clock	Frame Clock	Link Clock	F1_ FRAMECLK_ DIV	F2_ FRAMECLK_ DIV
Bonded/Non-bonded	1	4	16	8	153.6	76.8	153.6	1	
Bonded/Non-bonded	1	8	1	1	307.2	153.6	153.6	4	
Bonded/Non-bonded	1	8	1	2	307.2	153.6	153.6		2
Bonded/Non-bonded	1	8	2	1	307.2	153.6	153.6	4	
Bonded/Non-bonded	1	8	2	2	307.2	153.6	153.6		2
Bonded/Non-bonded	1	8	2	4	307.2	153.6	153.6	1	
Bonded/Non-bonded	1	8	4	1	307.2	153.6	153.6	4	
Bonded/Non-bonded	1	8	4	2	307.2	153.6	153.6		2
Bonded/Non-bonded	1	8	4	4	307.2	153.6	153.6	1	
Bonded/Non-bonded	1	8	4	8	307.2	76.8	153.6	1	
Bonded/Non-bonded	1	8	8	2	307.2	153.6	153.6		2
Bonded/Non-bonded	1	8	8	4	307.2	153.6	153.6	1	
Bonded/Non-bonded	1	8	8	8	307.2	76.8	153.6	1	
Bonded/Non-bonded	1	8	16	4	307.2	153.6	153.6	1	
Bonded/Non-bonded	1	8	16	8	307.2	76.8	153.6	1	
Bonded/Non-bonded	1	8	32	8	307.2	76.8	153.6	1	
Bonded/Non-bonded	2	1	1	2	153.6	153.6	153.6		2
Bonded/Non-bonded	2	2	2	2	153.6	153.6	153.6		2
Dynamic Reconfiguration									
Non-bonded	2	2	2	2	153.6	153.6	153.6		2

The following figures show the datapath of single and multiple JESD204B links.

Figure 20: Datapath of A Single JESD204B Link

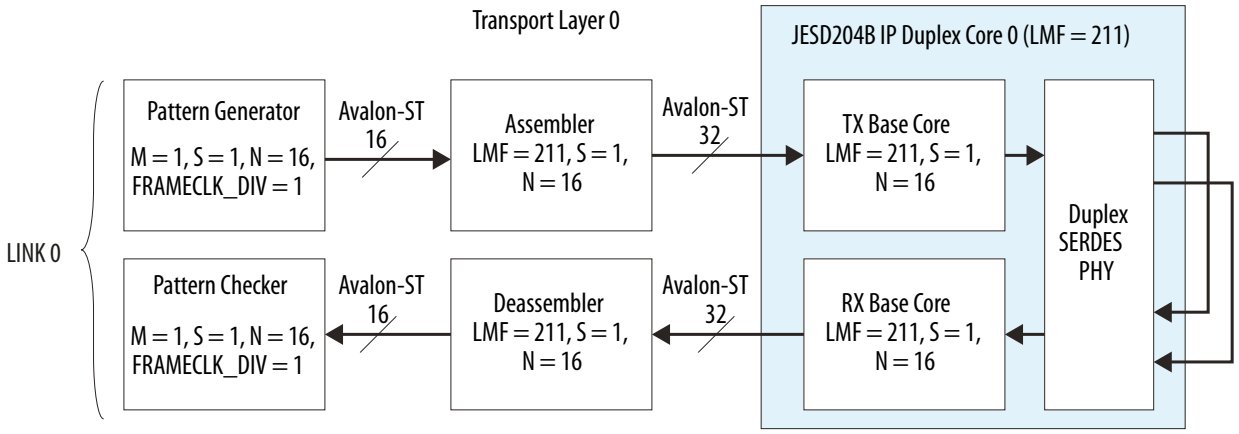
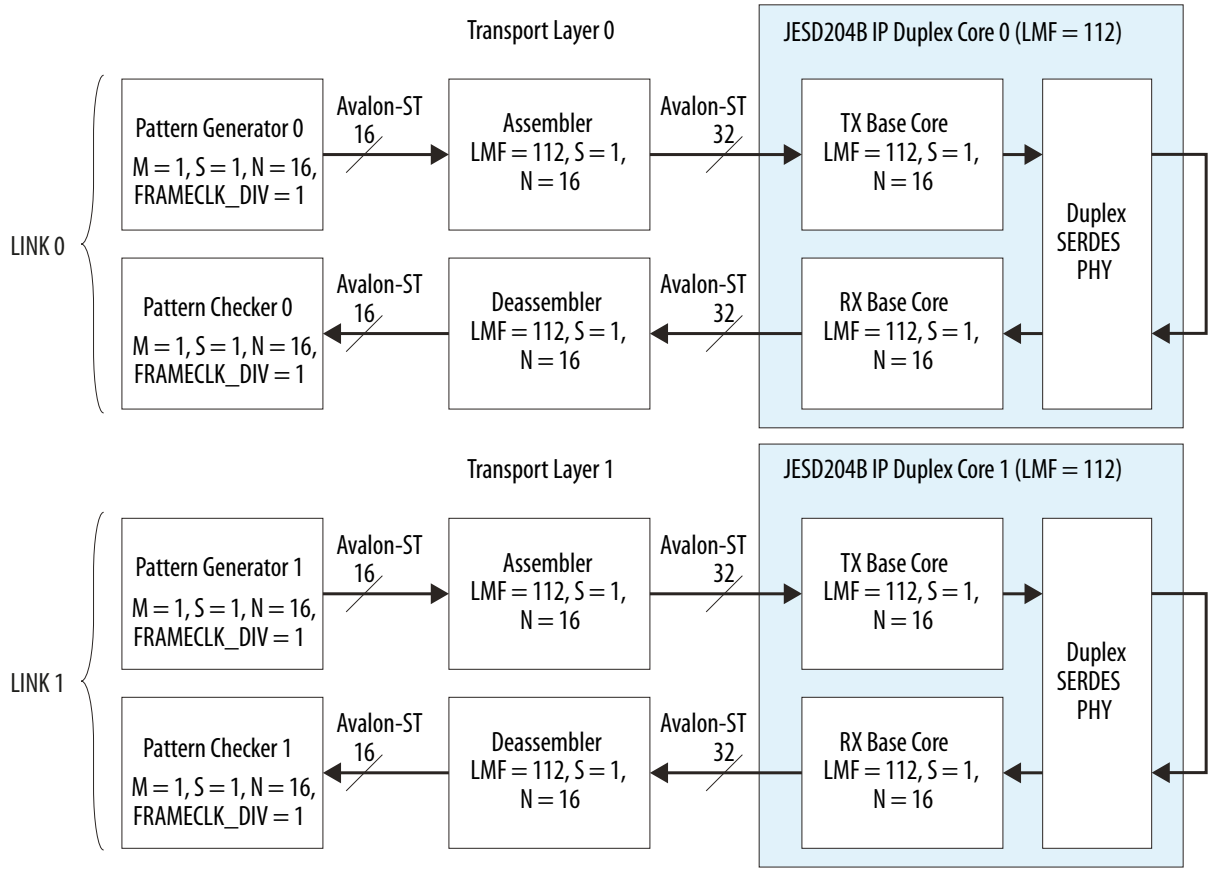


Figure 21: Datapath of Multiple JESD204B Links



Run-Time Reconfiguration

The JESD204B IP core supports run-time reconfiguration for the LMF and data rate settings. The design example only demonstrates the following set of configuration.

To generate the design example with run-time reconfiguration enabled, the LMF and bonding mode parameters must match the default value listed in the table below.

Table 27: Run-time Reconfiguration Demonstrated By The Design Example

Parameter	Default	Run-time Reconfiguration
LMF	222	112
FRAMECLK_DIV	2	2
Data Rate	6144 Mbps	3072 Mbps
Link Clock	153.6 MHz	76.8 MHz
Frame Clock	153.6 MHz	76.8 MHz
Bonding Mode	Non-bonded	Non-bonded

System Interface Signals

Table 28: Interface Signals

Signal	Clock Domain	Direction	Description
Clocks and Resets			
`device_clk`	—	Input	Device clock signal from the external converter or clock device.
`mgmt_clk`	—	Input	Management clock signal from the on-board 100 MHz oscillator.
`frame_clk`	—	Output	Internally generated clock. The Avalon-ST user data input must be synchronized to this clock domain for normal operation mode.
`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**
JESD204B			
`tx_sysref[LINK-1:0]`	`link_clk`	Input	TX *SYSREF* signal for JESD204B Subclass 1 implementation.

Signal	Clock Domain	Direction	Description
sync_n[LINK-1:0]	link_clk	Input	Indicates a TX SYNC_N from the receiver. This is an active low signal and is asserted 0 to indicate a synchronization request or error reporting.
mdev_sync_n[LINK-1:0]	link_clk	Input	Indicates a multidevice synchronization request at the TX path. Synchronize signal combination should be done externally and then input to the JESD204B IP core through this signal. In a single link instance where multidevice synchronization is not needed, you need to tie this signal to the dev_sync_n signal.
alldev_lane_aligned	link_clk	Input	Aligns all lanes for this device at the RX path. For multidevice synchronization, multiplex all the dev_lane_aligned signals before connecting to this signal pin. For single device support, connect the dev_lane_aligned signal back to this signal.
rx_sysref[LINK-1:0]	link_clk	Input	RX *SYSREF* signal for JESD204B Subclass 1 implementation.
tx_dev_sync_n[LINK-1:0]	link_clk	Output	Indicates a clean synchronization request at the TX path. This is an active low signal and is asserted 0 to indicate a synchronization request. The SYNC_N signal error reporting is masked out of this signal. This signal is also asserted during software-initiated synchronization.
dev_lane_aligned[LINK-1:0]	link_clk	Output	Indicates that all lanes for this device are aligned at the RX path.
rx_dev_sync_n[LINK-1:0]	link_clk	Output	Indicates a SYNC_N to the transmitter. This is an active low signal and is asserted 0 to indicate a synchronization request. Instead of reporting the link error through this signal, the JESD204B IP core uses the jesd204_rx_int signal to indicate an interrupt.
Signal	**Clock Domain**	**Direction**	**Description**
SPI			
miso	sclk	Input	Output data from a slave to the input of the master.

Signal	Clock Domain	Direction	Description
mosi	sclk	Output	Output data from the master to the inputs of the slaves.
sclk	mgmt_clk	Output	Clock driven by the master to slaves, to synchronize the data bits.
ss_n[2:0]	sclk	Output	Active low select signal driven by the master to individual slaves, to select the target slave. Defaults to 3 bits.
Signal	**Clock Domain**	**Direction**	**Description**
Serial Data and Control			
rx_serial_data[LINK*L-1:0]	—	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.
rx_seriallpbken[LINK*L-1:0]	—	Input	Assert this signal to enable internal serial loopback in the duplex transceiver.
Signal	**Clock Domain**	**Direction**	**Description**
User Request Control			
reconfig	mgmt_clk	Input	Active high reconfiguration request. Set this signal to static 0 during compile time if run time reconfiguration is not required.
runtime_lmf	mgmt_clk	Input	Reconfigure the LMF value at run-time. This value must be stable prior to assertion of the reconfig signal. • 0—Downscale to the LMF value stored in MIF file. • 1— Upscale back to maximum LMF value. Assuming at compile time, the LMF configuration is 222, set this signal to 0 to scale down the LMF configuration to 112. Set this signal to 1 to scale up the LMF configuration back to 222.

Signal	Clock Domain	Direction	Description
runtime_datarate	mgmt_clk	Input	Reconfigure the data rate at run-time. This value must be stable prior to assertion of reconfig signal. • 0— Downscale to data rate setting stored in PLL, PHY, and clock MIF. • 1— Upscale back to maximum data rate setting stored in PLL, PHY, and clock MIF. Assuming the compile time data rate is 3.072 Gbps, set this signal to 0 to scale down the data rate to 1.536 Gbps. Set this signal to 1 to scale up the data rate back to 3.072 Gbps.
cu_busy	mgmt_clk	Output	Assert high to indicate that the control unit is busy. All reconfiguration input will be ignored when this signal is high.
Signal	**Clock Domain**	**Direction**	**Description**
Avalon- ST User Data			

Signal	Clock Domain	Direction	Description
avst_usr_din[(FRAMECLK_DIV*LINK*M*S*N)-1:0]	frame_clk	Input	TX data from the Avalon-ST source interface. The source arranges the data in a specific order, as illustrated in the cases below: Case 1: If F1/F2_FRAMECLK_DIV =1, LINK = 1, M = 1, S =1, N = 16: • avst_usr_din[15:0] Case 2: If F1/F2_FRAMECLK_DIV =1, LINK = 1, M = 2 (denoted by m0 and m1), S =1, N = 16: • avst_usr_din[15:0] = m0[15:0] • avst_usr_din[31:16] = m1[15:0] Case 3: If F1/F2_FRAMECLK_DIV =1, LINK = 2 (denoted by link0 and link1), M = 1, S =1, N = 16: • avst_usr_din[15:0] = link0 • avst_usr_din[31:16] = link1 Case 4: If F1/F2_FRAMECLK_DIV =1, LINK = 2 (denoted by link0 and link1), M = 2 (denoted by m0 and m1), S =1, N = 16: • avst_usr_din[15:0] = link0, m0[15:0] • avst_usr_din[31:16] = link0, m1[15:0] • avst_usr_din[47:32] = link1, m0[15:0] • avst_usr_din[63:48] = link1, m1[15:0]
avst_usr_din_valid	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	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

Signal	Clock Domain	Direction	Description
avst_usr_dout[(FRAMECLK_DIV*LINK*M*S*N)-1:0]	frame_clk	Output	RX data to the Avalon-ST sink interface. The transport layer arranges the data in a specific order, as illustrated in the cases below: Case 1: If F1/F2_FRAMECLK_DIV =1, LINK = 1, M = 1, S =1, N = 16: • avst_usr_dout[15:0] Case 2: If F1/F2_FRAMECLK_DIV =1, LINK = 1, M = 2 (denoted by m0 and m1), S =1, N = 16: • avst_usr_dout[15:0] = m0[15:0] • avst_usr_dout[31:16] = m1[15:0] Case 3: If F1/F2_FRAMECLK_DIV =1, LINK = 2 (denoted by link0 and link1), M = 1, S =1, N = 16: • avst_usr_dout[15:0] = link0 • avst_usr_dout[31:16] = link1 Case 4: If F1/F2_FRAMECLK_DIV =1, LINK = 2 (denoted by link0 and link1), M = 2 (denoted by m0 and m1), S =1, N = 16: • avst_usr_dout[15:0] = link0, m0[15:0] • avst_usr_dout[31:16] = link0, m1[15:0] • avst_usr_dout[47:32] = link1, m0[15:0] • avst_usr_dout[63:48] = link1, m1[15:0]
avst_usr_dout_valid	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	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

Signal	Clock Domain	Direction	Description
test_mode[3:0]	frame_clk	Input	Specifies the operation mode. • 0000—Normal mode. The design example takes data from the Avalon-ST source. • 1000—Test mode. The design example generates alternate checkerboard data pattern. • 1001—Test mode. The design example generates ramp wave data pattern. • 1010—Test mode. The design example generates the PRBS data pattern. • Others—Reserved
Signal	**Clock Domain**	**Direction**	**Description**
Status			
rx_is_lockedtodata [LINK*L-1:0]	device_clk	Output	Asserted to indicate that the RX CDR PLL is locked to the RX data and the RX CDR has changed from LTR to LTD mode.
data_error [LINK-1:0]	frame_clk	Output	Asserted to indicate that the pattern checker has found a mismatch in the received data and the expected data. One error signal per pattern checker.
jesd204_tx_int[LINK-1:0]	link_clk	Output	Interrupt pin for the JESD204B IP core (TX). The interrupt signal is asserted when an error condition or synchronization request is detected.
jesd204_rx_int[LINK-1:0]	link_clk	Output	Interrupt pin for the JESD204B IP core (RX). The interrupt signal is asserted when an error condition or synchronization request is detected.

Example Feature: Dynamic Reconfiguration

The JESD204B IP core design example demonstrates dynamic (run-time) reconfiguration of either the LMF or data rate, at any one time.

Dynamic Reconfiguration Operation

The dynamic reconfiguration feature implements various reconfiguration controller modules such as PLL reconfiguration, Transceiver Reconfiguration Controller, SPI master, and JESD204B IP core Avalon-MM slave. These modules connect to the control unit through the Avalon-MM interface. You can control the reconfiguration using the `reconfig`, `runtime_lmf`, and `runtime_datarate` input ports exposed at control unit interface.

Figure 22: Dynamic Reconfiguration Block Diagram (For 28 nm Device Families—Stratix V and Arria V)

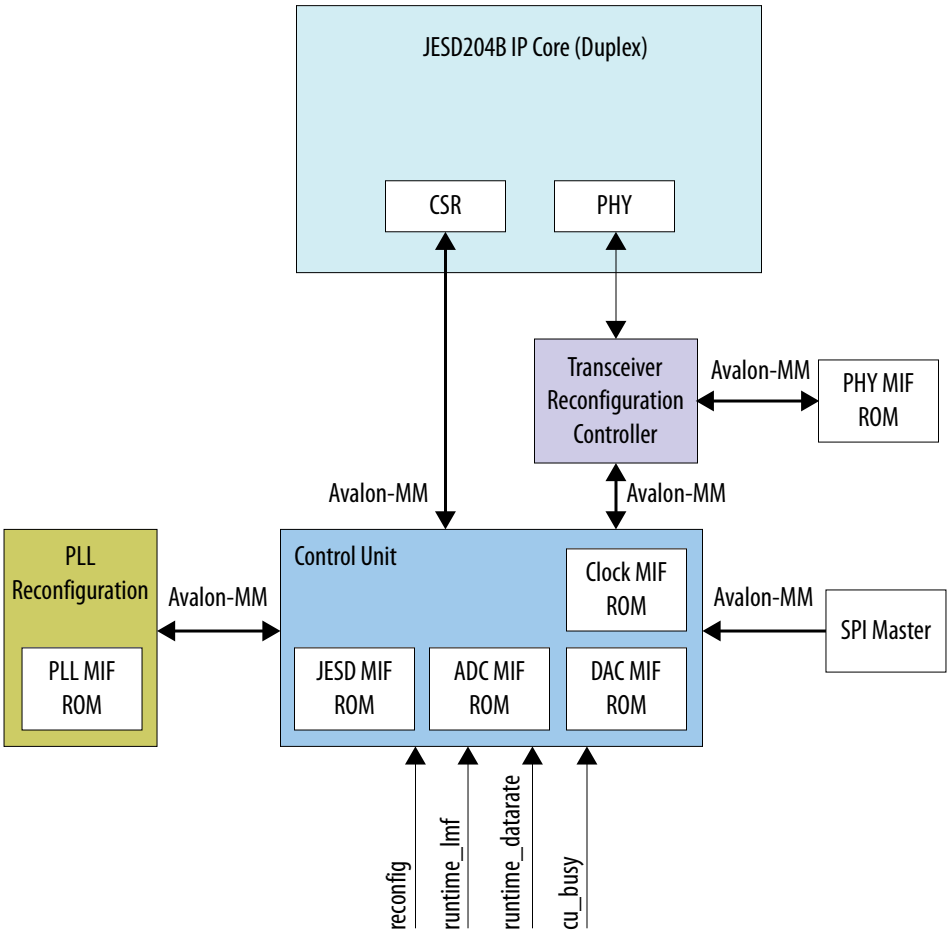
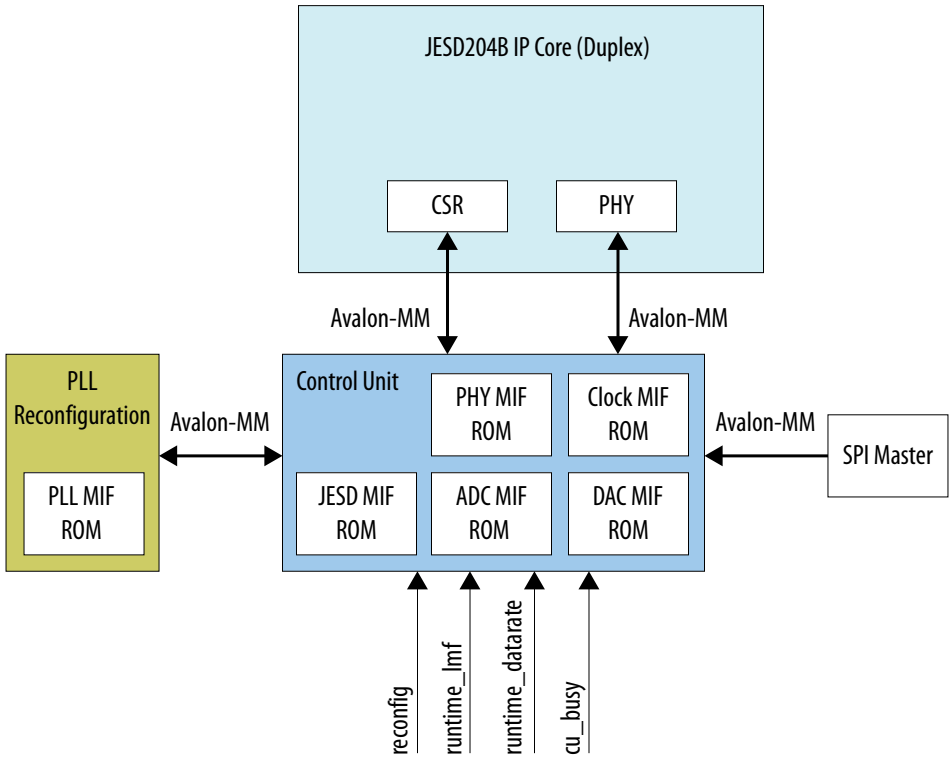


Figure 23: Dynamic Reconfiguration Block Diagram (For 20 nm Device Families—Arria 10)



The MIF ROM content for maximum and downscale configuration:

- PLL MIF ROM—contains the PLL counter, charge pump, and bandwidth setting.
- JESD MIF ROM—contains the LMF information.
- PHY MIF ROM—contains the transceiver channel and PLL setting.
- ADC MIF ROM—contains the ADC converter setting.
- DAC MIF ROM—contains the DAC converter setting.
- CLK MIF ROM—contains the device clock setting.

MIF ROM

You need to generate two MIF files for each reconfigurable IP core as shown in **Figure 23** or **Figure 24**, and merge them into a single MIF file for each IP core. The following section shows the MIF file format.

Core PLL

The MIF format is fixed by the PLL. You need to generate two PLLs with maximum and downscale setting to get these two MIF files. Then, merge the files into one (*core_pll.mif*). Only the PLL with maximum configuration is used in final compilation.

```
Maximum Configuration MIF
WIDTH=32;
DEPTH=92;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;
```

```
CONTENT BEGIN
    0   :   00000000000000000000000000111110; -- START OF MIF
    1   :   00000000000000000000000000000100;
    2   :   00000000000000000000000100000001;
    3   :   00000000000000000000000000000011;
.
.
.
 42  :   00000000000000000000000000000010;
    43  :   00000000000000000000000000001000;
    44  :   00000000000000000000000001000000;
    45  :   00000000000000000000000000111111; -- END OF MIF



Downscale Configuration MIF

 46  :   00000000000000000000000000111110; -- START OF MIF
    47  :   00000000000000000000000000000100;
    48  :   00000000000000000000000100000001;
    49  :   00000000000000000000000000000011;
.
.
.
 88  :   00000000000000000000000000000010;
    89  :   00000000000000000000000000001000;
    90  :   00000000000000000000000001000000;
    91  :   00000000000000000000000000111111; -- END OF MIF
END;
```

PHY (Stratix V and Arria V)

The MIF format is fixed by the PHY. You need to generate two JESD204B IP cores with maximum and downscale setting. Then, compile each of the setting to get a total of four MIF files (two for TX PLL and two for channel MIF). Then, merge the files into one (*phy.mif*). Only the JESD204B IP cores with maximum configuration is used in final compilation.

```
Maximum TX PLL Configuration MIF
WIDTH=16;
DEPTH=186;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
    0    :   0000000000100001; -- Start of MIF opcode (TX_PLL, 6144Mbps)
    1    :   0000000000100010;
.
.
.
 10   :   0011000000000000;
    11   :   0000000000011111; -- End of MIF opcode



Maximum Channel Configuration MIF

 12   :   0000000000100001; -- Start of MIF opcode (Channel, 6144Mbps)
    13   :   0000000000000010;
.
.
.
```

```
 [88..91]  :   0000000000000000;
    92   :   0000000000011111; -- End of MIF opcode


Downscale TX PLL Configuration MIF

 93   :   0000000000100001; -- Start of MIF opcode (TX_PLL, 3072Mbps)
    94   :   0000000000100010;
.
.
.
 103  :   0011000000000000;
    104  :   0000000000011111; -- End of MIF opcode


Downscale Channel Configuration MIF

 105  :   0000000000100001; -- Start of MIF opcode (Channel, 3072Mbps)
    106  :   0000000000000010;
.
.
.
 [181..184]  :   0000000000000000;
    185  :   0000000000011111; -- End of MIF opcode
END;
```

PHY (Arria 10)

The MIF format is fixed by the PHY. You need to generate two JESD204B IP cores with maximum and downscale setting. Then, compile each of the setting to get a total of four MIF files (two for TX PLL and two for channel MIF). Then, merge the files into two (*xcvr_atx_pll_combined.mif* and *xcvr_cdr_combined.mif*). Only the JESD204B IP cores with maximum configuration is used in final compilation.

xcvr_atx_pll_combined.mif

```
Maximum Configuration MIF

CONTENT BEGIN
    00    :   102FF71; -- Start of MIF
    01    :   103BF01;
    02    :   1047F04;
    03    :   1054700;
.
.
.
 10   :   11AFF00;
    11   :   11CE020;
    12   :   11DE020;
    13   :   3FFFFFF; -- End of MIF


Downscale Channel Configuration MIF

    14    :   102FF71; -- Start of MIF
    15    :   103BF01;
    16    :   1047F04;
    17    :   1054700;
.
.
.
 24   :   11AFF00;
    25   :   11CE020;
    26   :   11DE020;
```

```
    27   :   3FFFFFF; -- End of MIF
END;
```

xcvr_cdr_combined.mif

```
Maximum Configuration MIF

CONTENT BEGIN
    00    :   006DF02; -- Start of MIF
    01    :   007FF09;
    02    :   008FF04;
    03    :   00AFF01;
.
.
.
 76   :   173FF31;
    77   :   1741F0C;
    78   :   1753F13;
    79   :   3FFFFFF; -- End of MIF


Downscale Channel Configuration MIF

    7A    :   006DF02; -- Start of MIF
    7B    :   007FF09;
    7C    :   008FF04;
    7D    :   00AFF01;
.
.
.
 F0   :   173FF31;
    F1   :   1741F0C;
    F2   :   1753F13;
    F3   :   3FFFFFF; -- End of MIF
END;
```

JESD

The current JESD MIF contains only the LMF information. You need to manually code the MIF content in the following format.

```
Maximum Configuration MIF
WIDTH=16;
DEPTH=16;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
    0    :   0000000000000001; -- L            (maximum config)
    1    :   0000000000000001; -- M
    2    :   0000000000000001; -- F
.
.
.
 3    :   1111111111111111; -- End of MIF
 [4..7]  :   0000000000000000;


Downscale Configuration MIF

 8    :   0000000000000000; -- L            (downscale config)
 9    :   0000000000000000; -- M
 10   :   0000000000000001; -- F
.
```

```
.
.
 11   :   1111111111111111; -- End of MIF
 [12..15]  :   0000000000000000;
END;
```

ADC/DAC/CLK

The content for ADC/DAC/CLK MIF is vendor-specific. The general format for the MIF is as shown below, with each section terminated by all 1's.

```
Maximum Configuration MIF
WIDTH=32;
DEPTH=128;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
    0   :   10000100000000000001000001111100;  -- (Maximum Config)
    1   :   10000100000000000001010000000101;
    2   :   10000100000000000001011000000101;
    3   :   10000100000000000001110000000010;
.
.
.
 28  :   10000001000000001111111100000001;
    29  :   10000001000000000101111100010100;
    30  :   11111111111111111111111111111111;  -- End of MIF
    [31..63] :   00000000000000000000000000000000;


Downscale Configuration MIF

 64  :   10000100000000000001000001111100;   -- (downscale config)
 65  :   10000100000000000001010000000101;
 66  :   10000100000000000001011000000101;
 67  :   10000100000000000001110000000010;
.
.
.
 92  :   10000001000000001111111100000001;
 93  :   10000001000000000101111100010100;
 94  :   11111111111111111111111111111111;  -- End of MIF
 95..127]  :   00000000000000000000000000000000;
END;
```

Generating and Simulating the Design Example

To use the JESD204B IP core design example testbench, follow these steps:

1. Generate the design example simulation testbench. Refer to **Generating the Design Example Simulation Model** on page 69
2. Simulate the design example using simulator-specific scripts. Refer to **Simulating the JESD204B IP Core Design Example** on page 69

Related Information

Selecting and Generating the Design Example on page 11

Generating the Design Example Simulation Model

After generating the IP core, generate the design example simulation testbench using the script (*gen_ed_sim_verilog.tcl* or *gen_ed_sim_vhdl*) located in the <*example_design_directory*>`/ed_sim` directory.

Note: For more information about the JESD204B design example testbench, refer to the `README_DESIGN_EXAMPLE.txt` file located in the <*example_design_directory*>`/ed_sim` folder.

To run the Tcl script using the Quartus Prime sofware, follow these steps:

1. Launch the Quartus Prime software.
2. On the View menu, click **Utility Windows** and select **Tcl Console**.
3. In the **Tcl Console**, type `cd <example_design_directory>/ed_sim` to go to the specified directory.
4. Type `source gen_ed_sim_verilog.tcl` (Verilog) or `source gen_ed_sim_vhdl.tcl` (VHDL) to generate the simulation files.

To run the Tcl script using the command line, follow these steps:

1. Obtain the Quartus Prime software resource.
2. Type `cd <example_design_directory>/ed_sim` to go to the specified directory.
3. Type `quartus_sh -t gen_ed_sim_verilog.tcl` (Verilog) or `quartus_sh -t gen_ed_sim_vhdl.tcl` (VHDL) to generate the simulation files.

Simulating the JESD204B IP Core Design Example

By default, the Quartus Prime software generates simulator-specific scripts containing commands to compile, elaborate, and simulate Altera IP models and simulation model library files. You can copy the commands into your simulation testbench script, or edit these files to add commands for compiling, elaborating, and simulating your design and testbench.

To simulate the design using the ModelSim-Altera SE/AE simulator, follow these steps:

1. Start the ModelSim-Altera simulator.
2. On the File menu, click **Change Directory** > **Select** <*example_design_directory*>**/ed_sim/testbench/mentor**.
3. On the File menu, click **Load** > **Macro file**. Select **run_tb_top.tcl**. This file compiles the design and runs the simulation automatically, providing a pass/fail indication on completion.

To simulate the design using the VCS MX simulator (in Linux), follow these steps:

1. Start the VCS MX simulator.
2. On the File menu, click **Change Directory** > **Select** <*example_design_directory*>**/ed_sim/testbench/synopsys/vcsmx**.
3. Run **run_tb_top.sh**. This file compiles the design and runs the simulation automatically, providing a pass/fail indication on completion.

To simulate the design using the Aldec Riviera-PRO simulator, follow these steps:

1. Start the Aldec Riviera-PRO simulator.
2. On the File menu, click **Change Directory** > **Select** <*example_design_directory*>**/ed_sim/testbench/aldec**.
3. On the Tools menu, click **Execute Macro**. Select **run_tb_top.tcl**. This file compiles the design and runs the simulation automatically, providing a pass/fail indication on completion.

Note: VHDL is not supported in Aldec Riviera (for Arria 10 devices only) simulator.

Generating the Design Example For Compilation

Use the *gen_quartus_synth.tcl* script to generate the JESD204B design example for compilation.

Note: If you use the Tcl console in the Quartus Prime software to generate the *gen_quartus_synth.tcl* script, close all Quartus Prime project before you start generating the script.

To run the Tcl script using the Quartus Prime sofware, follow these steps:

1. Launch the Quartus Prime software.
2. On the View menu, click **Utility Windows** and select **Tcl Console**.
3. In the **Tcl Console**, type `cd <example_design_directory>/ed_synth` to go to the specified directory.
4. Type `source gen_quartus_synth.tcl` to generate the JESD204B design example for compilation.

To run the Tcl script using the command line, follow these steps:

1. Obtain the Quartus Prime software resource.
2. Type `cd <example_design_directory>/ed_synth` to go to the specified directory.
3. Type `quartus_sh -t gen_ed_quartus_synth.tcl` to generate the JESD204B design example for compilation.

Related Information

Selecting and Generating the Design Example on page 11

Compiling the JESD204B IP Core Design Example

You can use the generated **.qip** file to include relevant files into your project. Generate the Quartus Prime synthesis compilation files by running the script (*gen_quartus_synth.tcl*) located in the <*example_design_directory*>`/ed_synth/` directory.

Note: If you use the Tcl console in the Quartus Prime software to generate the *gen_quartus_synth.tcl* script, close all Quartus Prime project before you start generating.

To compile your design using the Quartus Prime software , follow these steps:

1. Launch the Quartus Prime software.
2. On the File menu, click **Open Project** > **Select** <*example_design_directory*>`/ed_synth/example_design/`.
3. Select **jesd204b_ed.qpf**. (15)
4. On the Processing menu, click **Start Compilation**.

   At the end of the compilation, the Quartus Prime software provides a pass/fail indication.

(15) This is the default quartus project file that the Quartus Prime software automatically generates. You can edit this file and the **.qsf** file according to your design preference.

Design Example with Nios II Processor Control Unit

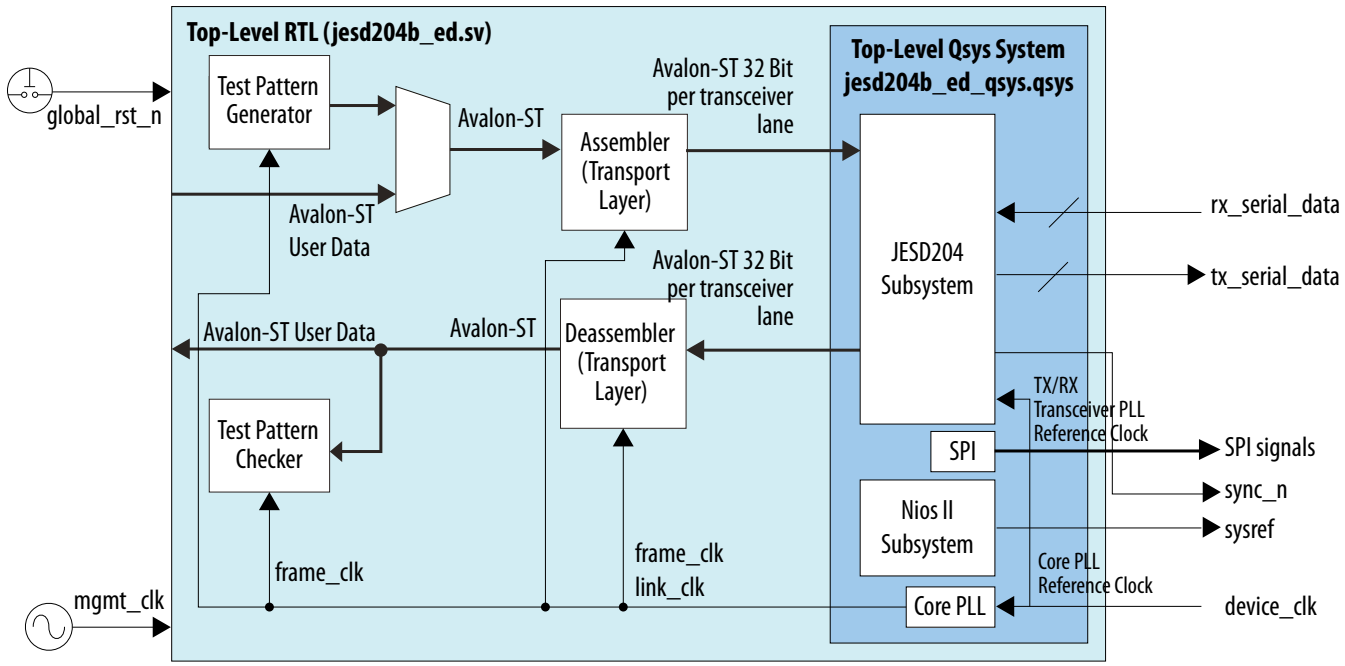
This design example with Nios II processor control unit provides an option if you need a software control flow for your JESD204B system.

Generate this design by selecting the **Nios II Control** option in the Example Designs tab of the parameter editor. You can also generate a generic design by selecting the **Generic Nios II Control** option in the **Generate generic example design** selection. This design example has the following key features:

- Supports Arria 10 devices only.
- C-based software control flow implemented on a Nios II soft core processor.
- Available as synthesizable design entity only.

Note: No simulation model is provided for this design. If you need a design example that has a simulation model, use the RTL State Machine Control design example.

Figure 24: Nios II Control Unit Design Example Block Diagram



Design Example Components

The Nios II processor control unit design example for the JESD204B IP core consists of the following components:

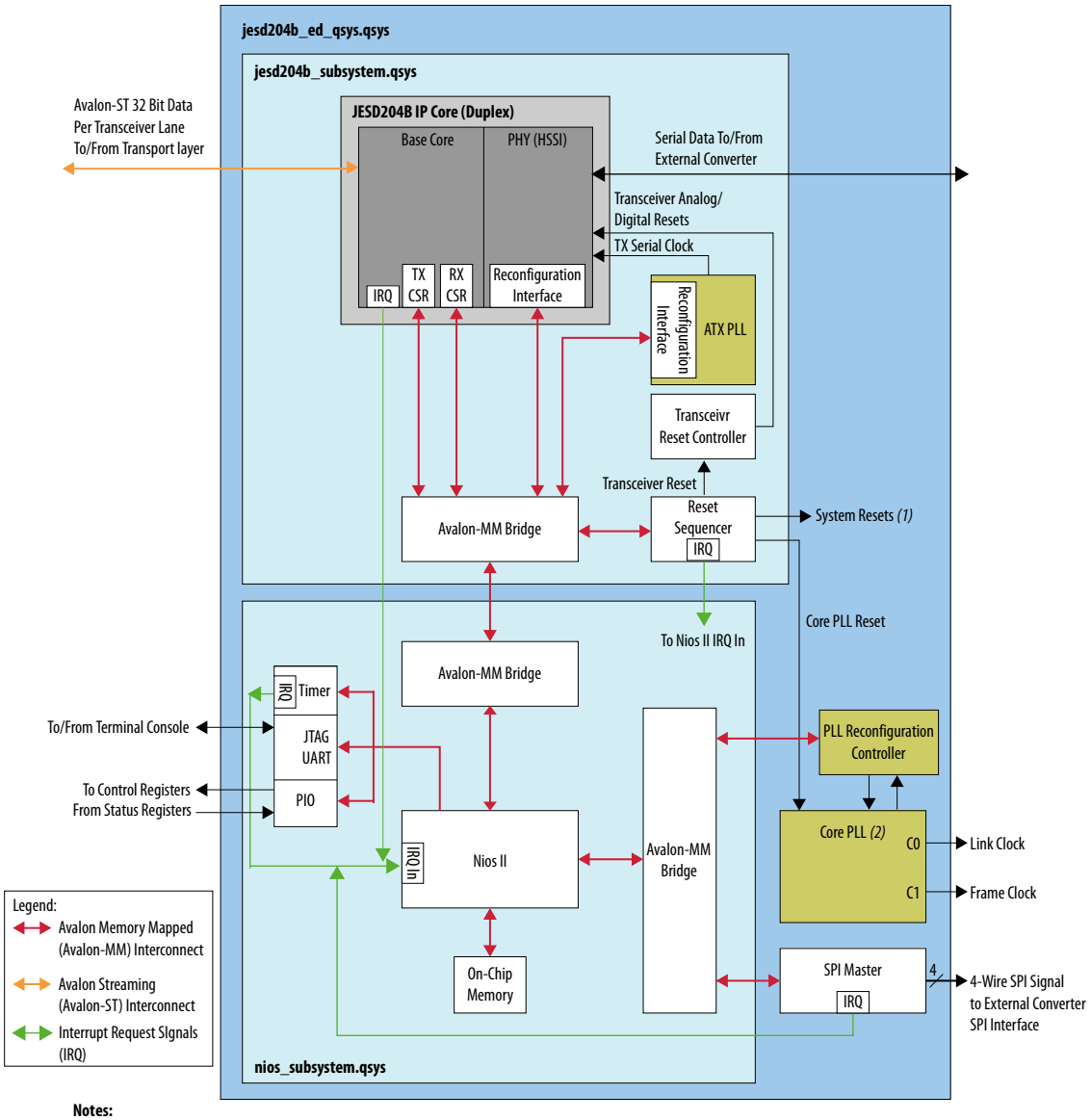
- Qsys system
  - JESD204B subsystem
  - Nios II subsystem
  - Core PLL
  - PLL reconfiguration controller
  - Serial Port Interface (SPI) – master module
- Test pattern generator
- Test pattern checker
- Assembler and deassembler (in the transport layer)

The following sections describe in detail the function of each component.

Qsys System Component

The Qsys system instantiates both the JESD204B IP core data path and the Nios II subsystem control path.

Figure 25: Qsys System



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. The PLL takes the device clock from an external clock chip as the input reference. The PLL generates two output clocks (using two output counters from a single VCO). The clock from output counter C0 is the link clock for the transport and link layer. The clock from output counter C1 is the frame clock for the transport layer, pattern generator, and pattern checker.

The top level Qsys system, *jesd204b_ed_qsys.qsys*, instantiates the following modules:

- JESD204B subsystem
- Nios II subsystem
- Core PLL
- PLL reconfiguration controller
- SPI master

The main data path flows through the JESD204B subsystem. In the example design, the JESD204B IP core is configured in duplex mode with both TX and RX data paths. On the TX data path, user data flows from the transport layer through the JESD204B IP core base module via a 32-bit per transceiver lane Avalon Streaming (Avalon-ST) interface and out as serial data to the external converters via the JESD204B IP core PHY module. On the RX data path, serial data flows from the external converters (or from the TX data path, in internal serial loopback mode) to the JESD204B IP core PHY module and out from the JESD204B IP core base module to the transport layer via a 32-bit per transceiver lane Avalon-ST interface.

The control path is centered on the Nios II processor in the Nios II subsystem and connects to various peripherals via the Avalon Memory-Mapped (Avalon-MM) interface. A secondary control path from the SPI master module links out to the SPI configuration interface of external converters via a 4-wire SPI interconnect. The configuration of the external converters is done by writing configuration data from the Nios II processor to the SPI master module. The SPI master module handles the serial transfer of data to the SPI interface on the converter end via the 4-wire SPI interconnect.

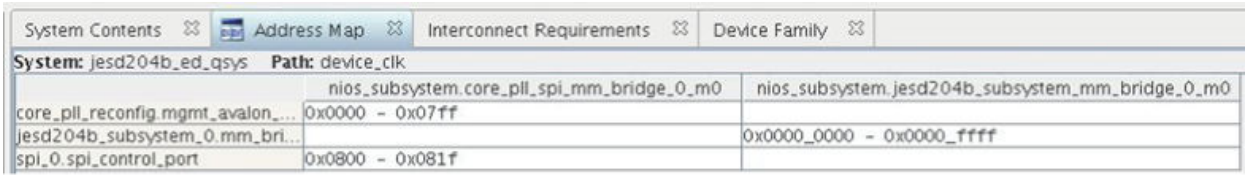
The core PLL generates the link clock and frame clock for the system. During a data rate dynamic reconfiguration process, the core PLL is dynamically reconfigurable at run time via the PLL reconfiguration controller.

To view the top level Qsys system in Qsys, follow these steps:

1. Launch the Quartus Prime software.
2. On the File menu, click Open.
3. Browse and select the `jesd204b_ed_qsys.qsys` file located in the project directory.
4. Click Open to view the Qsys system.

You can access the address mapping of the submodules in the top level Qsys project by clicking on the **Address Map** tab in the Qsys window.

Figure 26: Address Map View in Qsys



The Qsys system supports multi-link scenarios (up to 16 links) using the existing address map. To add more links to the system, add more *jesd204b_subsystem.qsys* modules to the project, connect them to the *jesd204b_subsystem* Avalon-MM bridge, and adjust the address map accordingly. Bits 16-19 of the *nios_subsystem-to-jesd204b_subsystem* Avalon-MM bridge are reserved to support multi-links.

JESD204B Subsystem in Qsys

The JESD204B subsystem Qsys project, *jesd204b_subsystem.qsys*, instantiates the following modules:

- JESD204B IP core (altera_jesd204) configured in duplex, non-bonded mode (with TX and RX datapaths)
- Reset sequencer (altera_reset_sequencer)
- Transceiver PHY reset controller (altera_xcvr_reset_control)
- ATX PLL (altera_xcvr_atx_pll_a10)
- Avalon-MM bridge (altera_avalon_mm_bridge)

The grouping of modules into a single Qsys subsystem project facilitates easy implementation of multi-link capabilities. For every link that you implement, a *jesd204b_subsystem.qsys* project is instantiated in the top level Qsys project and assigned an address as described in the Address Map section of the Qsys System section. Each link can be reset and dynamically reconfigured independently.

JESD204B IP Core

The generated example design is a self-contained system with its own JESD204B IP core. This IP core is separate from the IP core that is generated from the IP tab. The example design JESD204B IP core is configured in duplex mode (with TX and RX data paths) and has the IP parameter settings as set when you generate the example design. The JESD204B IP base core and PHY layer connect to the Nios II processor 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 (including data rate reconfiguration)

The Nios II processor writes to the JESD204B IP core CSR during a dynamic reconfiguration operation. By default, the software does not contain any dynamic reconfiguration features but you can use the Qsys system to implement such feature in the software.

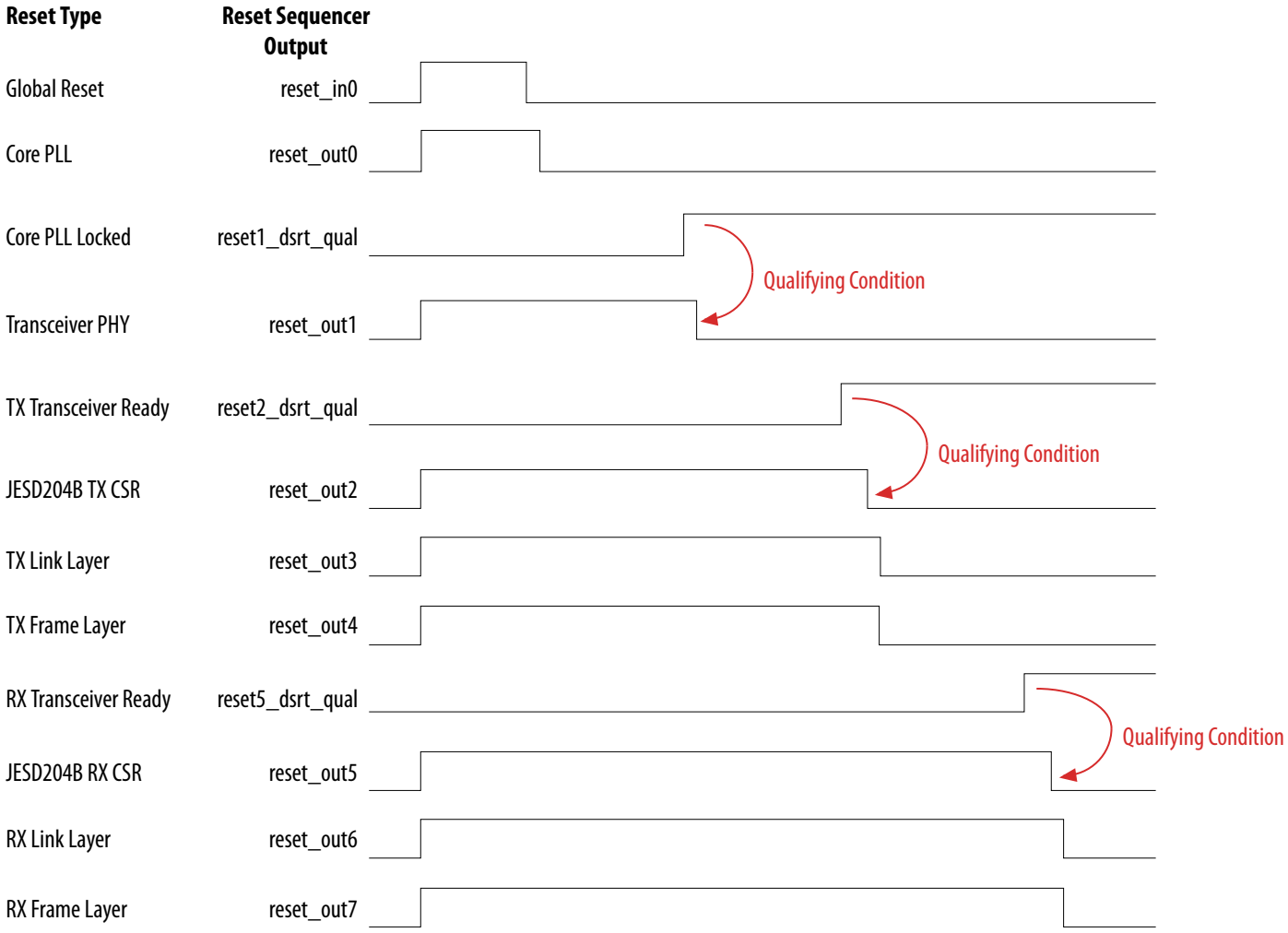
Reset Sequencer

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

- Core PLL reset—resets the core PLL
- Transceiver reset—resets the JESD204B IP core PHY module
- TX/RX JESD204B IP core CSR reset—resets the TX/RX JESD204B IP core CSRs
- TX/RX link reset—resets the TX/RX JESD204B IP core base module and transport layer
- 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_rst_n` input pin in the top level design. The Nios II processor executes a soft reset by issuing the reset command to the Avalon-MM interface of the reset sequencer. When you assert a hard reset or issue the relevant reset command via the Nios II processor, 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 27: Reset Sequence



Transceiver PHY Reset Controller

The transceiver PHY reset controller is a standard Qsys 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.

ATX PLL

The ATX PLL is a standard Qsys component in the IP Catalog standard library. This module supplies a low-jitter serial clock to the Arria 10 transceiver PHY module. The reference clock input to the ATX PLL is the *device_clk*. The ATX PLL has an Avalon-MM interface that connects to the Nios II processor via the Avalon-MM interconnect and can receive configuration instructions from the Nios II processor. By default, the software does not contain any dynamic reconfiguration features but you can use the Qsys system to implement such feature in the software.

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

You can access the address map of the submodules in the JESD204B subsystem by clicking on the **Address Map** tab in the Qsys window.

Table 29: JESD204B Subsystem Address Map

This table lists the memory allocation address map.

Avalon-MM Peripheral	Address Map
JESD204B IP core transceiver reconfiguration interface	0x0000 – 0x3FFF
ATX PLL (up to 4 modules per link)	0x8000 – 0x8FFF (Module 0) 0x9000 – 0x9FFF (Module 1) 0xA000 – 0xAFFF (Module 2) 0xB000 – 0xBFFF (Module 3)
JESD204B IP core CSR – TX	0xC000 – 0xC3FF
JESD204B IP core CSR – RX	0xD000 – 0xD3FF
Reset sequencer	0xE000 – 0xE0FF

Related Information

- **Quartus Prime Handbook Volume 1: Design and Synthesis**
  For detailed description of the reset sequencer module.
- **Arria 10 Transceiver PHY User Guide**
  For detailed description of the transceiver PHY reset controller and ATX PLL.

Nios II Subsystem in Qsys

The Nios II subsystem Qsys project, *nios_subsystem.qsys*[16], instantiates the following peripherals:

- Nios II processor (altera_nios2_gen2)
- On-chip memory (altera_avalon_onchip_memory2)—provides both instruction and data memory space
- Timer (altera_avalon_timer)—provides a general timer function for the software
- JTAG UART (altera_avalon_jtag_uart)—serves as the main communications portal between the user and the Nios II processor via the terminal console in Nios II SBT for Eclipse tool
- Avalon-MM bridges (altera_avalon_mm_bridge)—two Avalon-MM bridge modules; one to interface to the JESD204B subsystem and the other to interface to the Qsys components (core PLL reconfiguration controller and SPI master modules) in the top level Qsys project.
- PIO (altera_avalon_pio)—provides general input/output (I/O) access from the Nios II processor to the HDL components in the FPGA via two sets of 32-bit registers:
  - `io_status`—status registers input from the HDL components to the Nios II processor
  - `io_control`—control registers output from the Nios II processor to the HDL components

The tables below describe the signal connectivity for the `io_status` and `io_control` registers.

Table 30: Signal Connectivity for `io_status` Registers

Bit	Signal
0	Core PLL locked
1	TX transceiver ready (Link 0)
2	RX transceiver ready (Link 0)
3	Test pattern checker data error (Link 0)
4–31	TX transceiver ready, RX transceiver ready, and test pattern checker data error signals for subsequent links, if present.

Table 31: Signal Connectivity for `io_control` Registers

Bit	Signal
0	RX serial loopback enable for lane 0 (Link 0)
1	RX serial loopback enable for lane 1 (Link 0)
2	RX serial loopback enable for lane 2 (Link 0)
3	RX serial loopback enable for lane 3 (Link 0)
4–30	RX serial loopback enable for subsequent links, if present.
31	Sysref (Not implemented in software)

(16) Note that Qsys does not allow multiple Qsys systems with the same name in the same project. If you have a Qsys system with the same name, please change the name of your Qsys system.

You can access the address map of the submodules in the Nios II subsystem by clicking on the **Address Map** tab in the Qsys window.

Core PLL

The core PLL is an Arria 10 I/O PLL (altera_iopll) module that generates the clocks for the FPGA core fabric.

The core PLL uses the `device_clk` as its reference clock to generate two derivative clocks from a single VCO:

- Link clock – from output C0
- Frame clock – from output C1

Table 32: Clocks

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 F=1 and F=2, the resulting frame clock value can easily exceed the capability of the core PLL to generate and close timing. The top level RTL file (`jesd204b_ed.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 example design, the *F1_FRAMECLK_DIV* is set to 4 and *F2_FRAMECLK_DIV* is set to 2. For example, the actual frame clock for a serial data rate of 6.144 Gbps and F = 1 is:

(6144/(10 × 1)) / *F1_FRAMECLK_DIV* = 614.4 / 4 = 153.6 MHz

PLL Reconfiguration Controller

The PLL reconfiguration controller (*altera_pll_reconfig*) facilitates dynamic real-time reconfiguration of the core PLL.

You can use this IP core to update the output clock frequency, PLL bandwidth, and phase shifts in real time, without reconfiguring the entire FPGA. The PLL reconfiguration controller connects to the Nios II processor via the Avalon-MM interconnect. The Nios II processor sends dynamic reconfiguration instructions to the controller during a dynamic data rate reconfiguration operation. By default, the software does not contain any dynamic reconfiguration features but you can use the Qsys system to implement such feature in the software.

Related Information

- **AN 728: I/O PLL Reconfiguration and Dynamic Phase Shift for Arria 10 Devices**
  More details on implementing dynamic reconfiguration of the core PLL

- **AN 729: Implementing JESD204B IP Core System Reference Design with Nios II Processor As Control Unit**
  An example of implementing a full-featured software control flow with various user commands in a JESD204B system that incorporates a Nios II processor
- **JESD204B Reference Design**
  Available design examples in Altera Design Store.

SPI Master

The SPI master module (*altera_avalon_spi*) is a 4-wire, 24-bit width interface.

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 module is connects to the Nios II processor via the Avalon-MM interconnect. By default, the software does not contain any external converter configuration features but you can use the Qsys system to implement the feature in the software.

For more details on the SPI master module, refer to chapter 5.

Transport Layer

The transport layer in the JESD204B IP core 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 implemented in the top level RTL file, not in the Qsys project.

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

- The 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.
- The 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 may 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. This means that any dynamic reconfiguration operation that affects those values for the JESD204B IP core will affect the transport layer in the same way. By default, the software does not contain any dynamic reconfiguration features but you can use the Qsys system to implement such feature in the software.

For more details on the implementation of the transport layer in RTL and customization options, refer to chapter 5.

Test Pattern Generator

The test pattern generator generates one of three patterns; parallel PRBS, alternate checkerboard, or ramp wave, and sends it along to the transport layer during test mode.

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 certain 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 will affect 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. By default, the software does not contain any dynamic reconfiguration features but you can use the Qsys system to implement such feature in the software.

Note: The test pattern generator is implemented in the top level RTL file, not in the Qsys project.

Test Pattern Checker

The test pattern checker is implemented in the top level RTL file, not in the Qsys project. The test pattern checker checks one of three patterns; parallel PRBS, alternate checkerboard, or ramp wave from the transport layer during test mode. 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 certain 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 will affect 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. By default, the software does not contain any dynamic reconfiguration features but you can use the Qsys system to implement such feature in the software.

System Clocking

The main reference clock for the design data path is the `device_clk`, which is supplied from an external source. The `device_clk` is the reference clock for the core PLL and the TX/RX transceiver reference clocks. The core PLL generates the `link_clk` and `frame_clk` from the `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. The external source supplies a clock called the `mgmt_clk` to clock the control path of the design (the Nios II subsystem and any modules connected to the Nios II via the Avalon-MM bus interconnect).

Table 33: System Clocking for the Design Example

Clocks	Description	Source	Modules Clocked
`device_clk`	Reference clock for the data path	—	Core PLL, ATX PLL, RX transceiver PLL
`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	—	Nios II subsystem and any modules connected to Nios II via Avalon-MM bus interconnect

Nios II Processor Design Example Files

The design example is stored in the <*your project*>/ed_nios file directory. For the design example with Nios II processor control unit, only the synthesis flow is available; simulation flow is not available.

Table 34: Design Example Files

This table lists the important folders and files in the **ed_nios** file directory.

File Type	File/Folder	Description
Quartus project files	jesd204b_ed.qpf	Quartus project file.
	jesd204b_ed.qsf	Quartus settings file.
	output_files	Folder containing output files from Quartus compilation (for example, reports or sof)
Verilog HDL design files	jesd204b_ed.sv	Top level HDL.
	jesd204b_ed.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 RTL.
	switch_debouncer.v	Switch debouncer RTL.
Qsys Projects	jesd204b_ed_ qsys.qsys	Top level Qsys system project.
	jesd204b_ subsystem.qsys	JESD204B subsystem (refer to related information)
	se_outbuf_1bit.qsys	Output buffer module.
	se_outbuf_1bit	Folder containing the output buffer module.
	nios_subsystem.qsys	Nios II subsystem (refer to related information)
	jesd204b_ed_qsys	Folder containing generated HDL files from jesd204b_ed_ qsys.qsys.
	*.sopcinfo	Files containing system information for software project building (refer to related information) .
Software files	software	Folder containing all software-related files (detailed description in the *Software File Directory* table).

There are two folders for the software files:

- jesd204_nios_ed—contains all user source and header files.
- jesd204_nios_ed_bsp—board support package (BSP) that contains system files.

Table 35: Software File Directory

File Type	File	Description
Header files (in jesd204_nios_ed folder)	altera_jesd204_qsys_regs.h	Offsets, masks, and bit position definitions for peripherals in Qsys system that do not have standard access libraries. This includes the following peripherals: • JESD204B TX and RX CSR • Reset sequencer • PIO control • PIO status • Core PLL reconfiguration
	main.h	General user parameter definitions.
	functions.h	Contains function prototype definitions of sub-functions in main.c.
	macros.h	Contains function prototype definitions of macro functions in macros.c.
Source files (in jesd204_nios_ed folder)	main.c	Main C program. Also contain sub functions.
	macros.c	JESD204B Qsys system device access macros.
System files (in jesd204_nios_ed_bsp folder)	system.h	BSP-generated header file containing Qsys system-specific parameters such as: • Peripheral base addresses • Interrupt controller IDs • IRQ priorities **Attention:** Do not edit this auto-generated header file.

Related Information

- **JESD204B Subsystem in Qsys** on page 75
- **Nios II Subsystem in Qsys** on page 78
- **Running the Software Control Flow** on page 92

Nios II Processor Design Example System Parameters

The top level HDL file (`jesd204b_ed.sv`) includes system parameters that define the configuration of the example design as a whole. You can change the values in the HDL file after generation to customize to your desired configuration but the values must fall within the supported value ranges.

Table 36: System Parameter

Parameter	Value [17]	Description
LINK	1, 2	Number of JESD204B link. One link represent one JESD204B instance.
L	1, 2, 4, 8	Number of lanes per converter device.
M	1, 2, 4, 8	Number of converters per device.
F	1, 2, 4, 8	Number of octets per frame.
S	1, 2	Number of transmitted samples per converter per frame.
N	12–16	Number of conversion bits per converter.
N'	16	Number of transmitted bits per sample.
CS	0–3	Number of JESD204B control bits per conversion sample.
F1_FRAMECLK_DIV	1, 4	The divider ratio for `frame_clk` when F = 1 (refer to **Core PLL** on page 79 section.
F2_FRAMECLK_DIV	1, 2	The divider ratio for `frame_clk` when F = 2 (refer to **Core PLL** on page 79 section..
POLYNOMIAL_LENGTH	7, 9, 15, 23, 31	Defines the polynomial length for the PRBS pattern generator and checker, which is also the equivalent number of stages for the shift register. • If PRBS-7 is required, set this parameter to 7. • If PRBS-9 is required, set this parameter to 9. • If PRBS-15 is required, set this parameter to 15. • If PRBS-23 is required, set this parameter to 23. • If PRBS-31 is required, set this parameter to 31. This parameter value must not be larger than N, which is the output data width of the PRBS pattern generator or converter resolution. If an N of 12-14 is required, PRBS-7 and PRBS-9 are the only feasible options. If an N of 15-16 is required, PRBS-7, PRBS-9, and PRBS-15 are the only feasible options.
FEEDBACK_TAP	6, 5, 14, 18, 28	Defines the feedback tap for the PRBS pattern generator and checker. This is an intermediate stage that is XOR-ed with the last stage to generate to next PRBS bit. • If PRBS-7 is required, set this parameter to 6. • If PRBS-9 is required, set this parameter to 5. • If PRBS-15 is required, set this parameter to 14. • If PRBS-23 is required, set this parameter to 18. • If PRBS-31 is required, set this parameter to 28.

[17] Values supported or demonstrated by this design example.

Nios II Processor Design Example System Interface Signals

Table 37: System Interface Signals

Signal	Clock Domain	Direction	Description
Clocks and Resets			
device_clk	—	Input	Reference clock for JESD204B data path.
mgmt_clk	—	Input	Reference clock for Nios II processor control path and all peripherals connected via Avalon-MM interconnect.
sma_clkout	—	Output	Clock output to SMA connector on board (for test only).
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.
Signal	**Clock Domain**	**Direction**	**Description**
Avalon- ST User Data			

Signal	Clock Domain	Direction	Description
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: • 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

Signal	Clock Domain	Direction	Description
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	spi_SCLK	Input	Output data from a slave to the input of the master.
spi_MOSI	spi_SCLK	Output	Output data from the master to the inputs of the slaves.
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.

Compiling the Design Example for Synthesis

After generating the design example, all the necessary files for synthesis are stored in the *<your project>*/ed_nios directory.

To compile the design using the Quartus Prime software, follow these steps:

1. Launch the Quartus Prime software.
2. On the File menu, click **Open Project**.
3. Navigate to your project directory and select the Quartus project file (jesd204b_ed.qpf). Click **Open**.
   The Quartus project is now open in the **Project Navigator** window. If required, you can modify the HDL files and Qsys projects to customize the design configurations to your specifications.
4. On the Processing menu, select **Start Compilation** to compile the HDL.
   The Quartus Prime software compiles the design and indicates the compilation status in the **Tasks** window.

Related Information

Selecting and Generating the Design Example on page 11

Implementing the Design on the Development Kit

The **Target Development Kit** option in the parameter editor window gives you the option to target the example design to a development kit. For the Nios II processor design example, you can target the design to the Arria 10 GX FPGA Development Kit.

Note: The hardware example design targets an Arria 10 ES3 device (10AX115S3F45E2SGE3). It cannot function correctly on an Arria 10 production device.

When you select this target, the Quartus Settings File (`jesd204b_ed.qsf`) is updated with the following changes:

1. The target device is set to match the Arria 10 device on the Arria 10 GX FPGA development kit. The device part number is 10AX115S3F45E2SGE3.
2. Pin assignments are added for selected signals listed in the Pin Assignments section.

Pin Assignments

The serial data (tx_serial_data [n] and rx_serial_data[n]) pin assignments are shown for the maximum number of transceiver lanes supported by the design example, which is L = 8. For other configurations where L < 8, a subset of the serial data pin assignments are selected.

Table 38: Top Level Interface Ports and its Corresponding Pin Assignments

The interface ports of the top level HDL file (`jesd204b_ed.sv`) with its corresponding FPGA pin assignments on the Arria 10 FPGA development board are listed in the table below.

Interface Port Name	FPGA Pin Number	I/O Standard	Direction	Board Source/Destination
global_rst_n	T12	1.8 V	Input	User PB0 push-button
device_clk	AN8	LVDS	Input	On-board Si5338 programmable oscillator (Output: CLK1B)
device_clk (n)	AN7	LVDS	Input	On-board Si5338 programmable oscillator (Output: CLK1A)
mgmt_clk	AR36	LVDS	Input	On-board Si570 programmable oscillator (Output: 100 Mhz)
mgmt_clk(n)	AR37	LVDS	Input	On-board Si570 programmable oscillator (Output: 100 Mhz)
sma_clkout	E24	1.8 V	Output	SMA clock out
spi_MISO	AR22	1.8 V	Input	FMC Port A connector
spi_MOSI	AR11	1.8 V	Output	FMC Port A connector
spi_SCLK	AT10	1.8 V	Output	FMC Port A connector
spi_SS_n[0]	AV14	1.8 V	Output	FMC Port A connector
tx_serial_data[7]	AW3	High Speed Differential I/O	Output	FMC Port A connector

Interface Port Name	FPGA Pin Number	I/O Standard	Direction	Board Source/Destination
tx_serial_data[7](n)	AW4	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[6]	AY1	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[6](n)	AY2	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[5]	BA3	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[5](n)	BA4	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[4]	BB1	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[4](n)	BB2	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[3]	BC3	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[3](n)	BC4	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[2]	BB5	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[2](n)	BB6	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[1]	BD5	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[1](n)	BD6	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[0]	BC7	High Speed Differential I/O	Output	FMC Port A connector
tx_serial_data[0](n)	BC8	High Speed Differential I/O	Output	FMC Port A connector
rx_serial_data[7]	AM5	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[7](n)	AM6	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[6]	AN3	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[6](n)	AN4	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[5]	AP5	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[5](n)	AP6	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[4]	AT5	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[4](n)	AT6	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[3]	AV5	High Speed Differential I/O	Input	FMC Port A connector

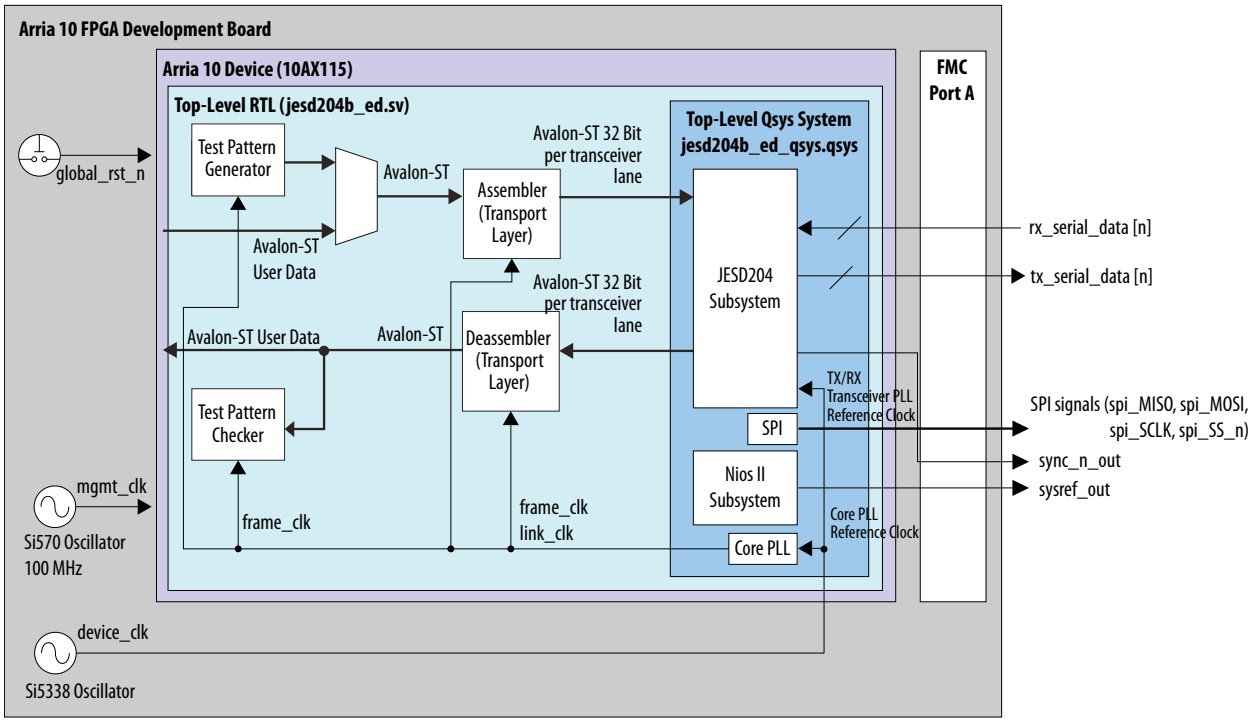
Interface Port Name	FPGA Pin Number	I/O Standard	Direction	Board Source/Destination
rx_serial_data[3] (n)	AV6	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[2]	AY5	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[2] (n)	AY6	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[1]	BA7	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[1] (n)	BA8	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[0]	AW7	High Speed Differential I/O	Input	FMC Port A connector
rx_serial_data[0] (n)	AW8	High Speed Differential I/O	Input	FMC Port A connector
sysref_out	AV11	LVDS	Output	FMC Port A connector
sysref_out (n)	AW11	LVDS	Output	FMC Port A connector
sync_n_out	AN20	LVDS	Output	FMC Port A connector
sync_n_out (n)	AP19	LVDS	Output	FMC Port A connector

Hardware Setup

The Arria 10 GX FPGA development board with a 10AX115 device features two FPGA mezzanine card (FMC) connectors for you to interoperate with external converters.

Figure 28: Block Diagram of the JESD204B Reference Design with Nios II Processor

This figure illustrates the example design block diagram implemented on the Arria 10 GX FPGA development board.



The JESD204B serial data, control, and configuration signal pins are assigned to FMC port A connector. To interoperate with your converter device, ensure that the pin assignments for the JESD204B serial data, control, and configuration ports are set correctly according your converter specifications. The global reset pin (`global_rst_n`) connects to the user PB0 push-button on the board. The control plane clock (`mgmt_clk`) is sourced from the on-board Si570 programmable oscillator. The Si570 clock output routes through a Si53301 clock buffer that allows you to select between the Si570 clock output and SMA input. Follow the instructions in the Programming the Device section to set the board settings correctly.

The example design is configured in internal serial loopback mode. Therefore, the JESD204B data path reference clock (`device_clk`) is sourced from an on-board clock source, the Si5338 programmable oscillator. In general, when interoperating with an external converter, the `device_clk` is sourced from the converter through the FMC connector. If this suits your configuration, modify the `device_clk` pin assignments accordingly to assign the pin to the FMC connector.

Programming the Device

Follow the steps below to setup and program the device on the development board.

1. Connect the mini-USB programming cable from your workstation to the mini-USB connector (J3) on the development board.
2. Connect the power adapter shipped with the development board to the power supply jack (J13).
3. Turn on the power for the development board.
4. To configure the clock frequency for the `mgmt_clk` that is sourced from the on-board Si570 programmable oscillator (X3), use the Altera Clock Control GUI that is included with the development kit. Select the Si570 (X3) tab, enter the required target frequency in the Target frequency (MHz) box and click **Set New Frequency**. The correct target frequency value is 100 MHz.
   You must set the CLK_SEL port of the Si53301 clock buffer module (U42) on the board to LOW to select the Si570 clock output. To set the CLK_SEL signal to LOW, set switch 1 of DIPSWITCH5 (SW6) to the ON position. Refer to the Arria 10 GX FPGA development kit schematic and related documentation for more details.
5. To configure the clock frequency for the `device_clk` that is sourced from the on-board Si5338 programmable oscillator, use the Altera Clock Control GUI that is included with the development kit. Select the Si5338 (U14) tab, enter the required target frequency in the CLK1 box and click **Set New Freq**. The correct target frequency value depends on the L parameter selected for the example design. Table below lists the required values for the `device_clk`, with the serial data rate set as 6.144 Gbps.

L Parameter	Target Frequency (MHz)
8	307.2
Others	153.6

6. Compile the design as described in the Compiling Example Design For Synthesis section.
7. In the Tools menu, click **Programmer**.
8. In the Programmer window, click **Add File**.
9. In the Select Programming File window, navigate to *<your project>*`/ed_nios/output_files` and select the SOF programming file (jesd204b_ed.sof). Click **Open**.
10. Verify that all the hardware setup options are set correctly to your system configurations.
11. Click **Start** to program the file into the board device.

After programming the Arria 10 FPGA device on the development board, the system needs to be initialized via software before the link is fully active. Follow the steps in the Executing the Software C Code section to complete the link initialization process.

Attention: Do not skip this step. The JESD204B link will not function correctly without software link initialization.

Related Information

Executing the Software C Code on page 94

Running the Software Control Flow

The key feature of the design example with Nios II processor control unit is the ability to control certain aspects of JESD204B system using a C-based, software control flow.

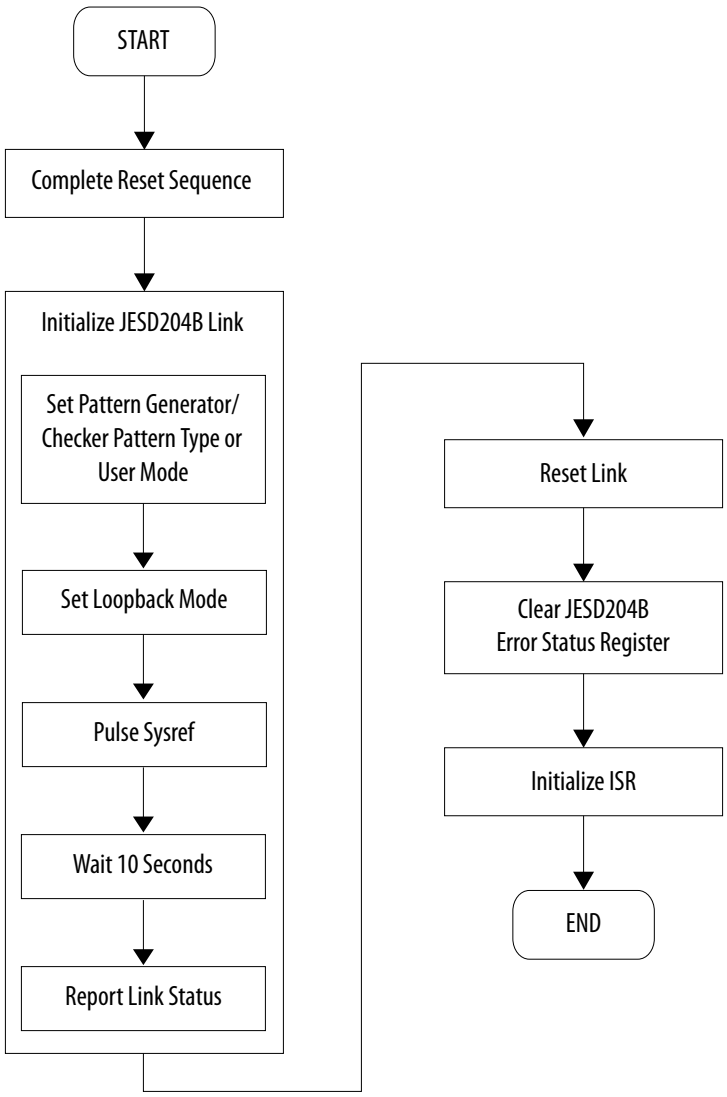
The software control flow allows you to perform the following tasks:

- 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.

Figure 29: Software C Code Execution Flow

Figure illustrates the main C code execution flow



The JESD204B link initialization performs the following tasks:

- Set the pattern type or user mode for the pattern generator or 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.

Related Information

Selecting and Generating the Design Example on page 11

Executing the Software C Code

To execute the software code and initialize the JESD204B link:

1. Program the device on the board with the FPGA programming file as described in the **Programming the Device** on page 92 section.
2. In the Quartus Prime software, navigate to the Tools menu and select **Nios II Software Build Tools for Eclipse**.
3. In the Select a workspace dialog box, navigate to the software workspace, <*your project*>`/ed_nios/software` and click **OK**.
4. Create a new Nios II application and board support package (BSP) from the template. On the File menu, navigate to New and click **Nios II Application and BSP From Template**.
5. In the Nios II Application and BSP From Template window, enter the following information:
   - SOPC Information File Name: <*your project*>`/ed_nios/jesd204b_ed_qsys.sopcinfo`
   - Project name: **jesd204_nios_ed** (18)
   - User default location: Checked
   - Templates: Blank Project
6. Click **Next**. Verify that the default BSP name is `jesd204_nios_ed_bsp`, then click **Finish**. The Nios II application project (`jesd204_nios_ed`) and BSP (`jesd204_nios_ed_bsp`) appears in the Project Explorer window.

   Note: Whenever you modify and recompile the Quartus project, you must regenerate the BSP files. In the Project Explorer window, right-click the `jesd204_nios_ed_bsp` project, navigate to Nios II and click **Generate**. This regenerates the BSP files based on your most current compiled Quartus project settings.
7. Import the design example source (*.c) and header (*.h) files into the application directory. In the Project Explorer window, right click on the `jesd204_nios_ed` project and click **Import**.
8. In the Import window, select **General** > **File System** as the import source. Click **Next>**.
9. Browse to the <*your project*>`/ed_nios/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:

(18) You can choose any name for the project. This project name is given for reference purposes only.

- altera_jesd204_regs.h
- functions.h
- macros.h
- main.h
- macros.c
- main.c

Verify that the destination folder is `jesd204_nios_ed`. Click **Finish**. All the source and header files should be imported into the `jesd204_nios_ed` project directory.

10. Right-click the `jesd204_nios_ed_bsp` project, navigate to Nios II and click **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**.
11. Expand the `jesd204_nios_ed` application project in the Project Explorer window and verify that the folder contains all the source and header files.
12. To compile the C code, navigate to the Project menu and select **Build All**. The compiler now compiles the C code into executable code.
13. To download the executable code to the development board, navigate to the Run menu and select **Run Configurations**. In the Run Configurations window, double-click **Nios II Hardware** on the left panel. Check that all run configurations are correct, then click **Run** on the bottom right corner of the window.

The 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.

All `printf` statements in the C code print to this console window. Similarly, all user input functions like `getc`, `gets`, and `scanf` get user input from this console window. At the end of the initialization sequence, the code prints the JESD204B link status to the console. The following tables list the expected values of the link status register report.

Table 39: TX Status 0 Register Bits

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 40: RX Status 0 Register Bits

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

The code also reports the status of the pattern checker. Any pattern checker errors that occur during the initialization period is flagged in the console window.

Software Parameters

The software parameters defined in the main header file, `main.h`, controls various behaviors of the C code.

Table 41: 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. 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 Qsys.
LOOPBACK_INIT	1	Initial value of the loopback. Set to 1 for internal serial loopback mode, else set to 0.

Parameter	Default Value	Description
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.

Software Interrupt Service Routines (ISR)

One key feature of the Nios II processor control unit 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 II 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. The *PRINT_INTERRUPT_MESSAGES* parameter in the `main.h` header file controls the printing of interrupt error messages in the Nios II console window. Set the parameter to 1 (default) to print error messages, else set to 0. Refer to the Software Parameters section for more details. You can modify the ISRs in the C code to customize the interrupt handling response based on your system specifications.

Related Information

Software Parameters on page 96

Software Functions Description

The software C code provided 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.

Functions in main.c Source File

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

Table 42: 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.

Function Prototype	Description
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.
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.

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 43: 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 QSYS 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 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.

Function Prototype	Description
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.
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.

Function Prototype	Description
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*.
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.

Function Prototype	Description
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.
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.

Related Information

AN 729: Implementing JESD204B IP Core System Reference Design with Nios II Processor As Control Unit

For an example of a full-featured software C code that includes user command interface to execute various features such as dynamic reconfiguration or external SPI configuration.

Customizing the Design Example

Use the following guidelines if you want to customize the design example that you generate from the IP core's parameter editor.

Modifying the JESD204B IP Core Parameters Post-Generation

In the event that the design examples available does not suit your desired JESD204B IP parameter configuration, you can generate a generic example design and then manually modify the parameters to meet your specifications.

1. Generate a generic design example with Nios II control unit.
2. Open the top level Qsys system (`jesd204b_ed_qsys.qsys`) in Qsys.
3. In the Qsys window, navigate to the File menu and click **Open**. Select `jesd204b_subsystem.qsys` and click **Open**.
4. In the System Contents tab, double-click the jesd204b module. This brings up the parameter editor that shows the current parameter settings of the JESD204B IP core. This is the duplicate JESD204B IP core that is generated in the design example folder, not the JESD204B IP core that is generated from the IP tab of the parameter editor, which is stored in a different folder.
5. Modify the IP core parameters of the jesd204b module as per your system specifications. When you are done, navigate to the File menu and click **Save**.
6. In the File menu, click **Open**. Select the top level Qsys project, `jesd204b_ed_qsys.qsys` and click **Open**.
7. Verify that the `jesd204b_subsystem.qsys` project has been updated with the new IP parameters for the jesd204b module. Right-click the jesd204b_subsystem_0 module and select **Drill into subsystem**. The `jesd204b_subsystem.qsys` project opens in the Qsys window. In the System Contents tab, double-click on the jesd204b module and verify that the parameters in the parameters tab match the ones that you have updated earlier. When you are done verifying, click **Move to the top of hierarchy** to move back to the `jesd204b_ed_qsys.qsys` view.
8. Click **Generate HDL** to generate the HDL files needed for Quartus compilation.
9. After the HDL generation is completed, click **Finish** to save your Qsys settings and exit the Qsys window.
10. You have to manually change the system parameters in the top level RTL file to match the parameters that you set in the Qsys project, if applicable. Open the top level RTL file (`jesd204b_ed.sv`) in any text editor of your choice.
11. Modify the system parameters at the top of the file to match the new JESD204B IP core settings in the Qsys project, if applicable. Refer to the **Nios II Processor Design Example System Parameters** on page 83 for more details on the system parameters.
12. Save the file and compile the design in Quartus Prime software as per the instructions in the **Compiling the Design Example for Synthesis** on page 87.

Changing the Data Rate or Reference Clock Frequency

When changing the data rate or reference clock frequency, be aware of the relationships between the serial data rate, link clock, and frame clock as described in the Core PLL section and change the PLL output clock settings accordingly to meet the clock frequency requirements.

Also be aware of the F1_FRAMECLK_DIV and F2_FRAMECLK_DIV frame clock division factor parameters 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 the timing closure for the FPGA core fabric.

To change the serial data rate or reference clock frequency:

1. Open the `jesd204b_subsystem.qsys` project in the Qsys window.
2. Double-click the jesd204b module to bring up the parameters editor for the JESD204B IP core.
3. Change the **Data rate** and **PLL/CDR Reference Clock Frequency** values to meet your system requirements.
4. Double-click the xcvr_atx_pll_a10_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.
5. 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. Note that the PLL output frequency is half of the PLL output data rate as the clocking of the TX data is in DDR mode. Ensure that the data rate and PLL reference clock values match the parameters that were entered into the jesd204b module.
6. Navigate back to the top level `jesd204b_ed_qsys.qsys` hierarchy.
7. Double-click the core_pll module to bring up the parameters editor for the core PLL module.
8. 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 and xcvr_atx_pll_a10_0 modules. Also change the outclk0 group settings (which correspond to the link clock) and outclk1 group settings (which correspond to the frame clock) where necessary. Ensure that the `link_clk` and `frame_clk` values satisfy the frequency requirements as described in the Core PLL section.
9. Click **Generate HDL** to generate the design files needed for Quartus Prime compilation.
10. After the HDL generation is completed, click **Finish** to save your Qsys settings and exit the Qsys window.
11. If the frame clock settings (outclk1 of the core_pll module) are such that F1_FRAMECLK_DIV or F2_FRAMECLK_DIV values are 1, change the relevant system parameters in the top level design file, `jesd204b_ed.sv`.
12. Save the file and compile the design.

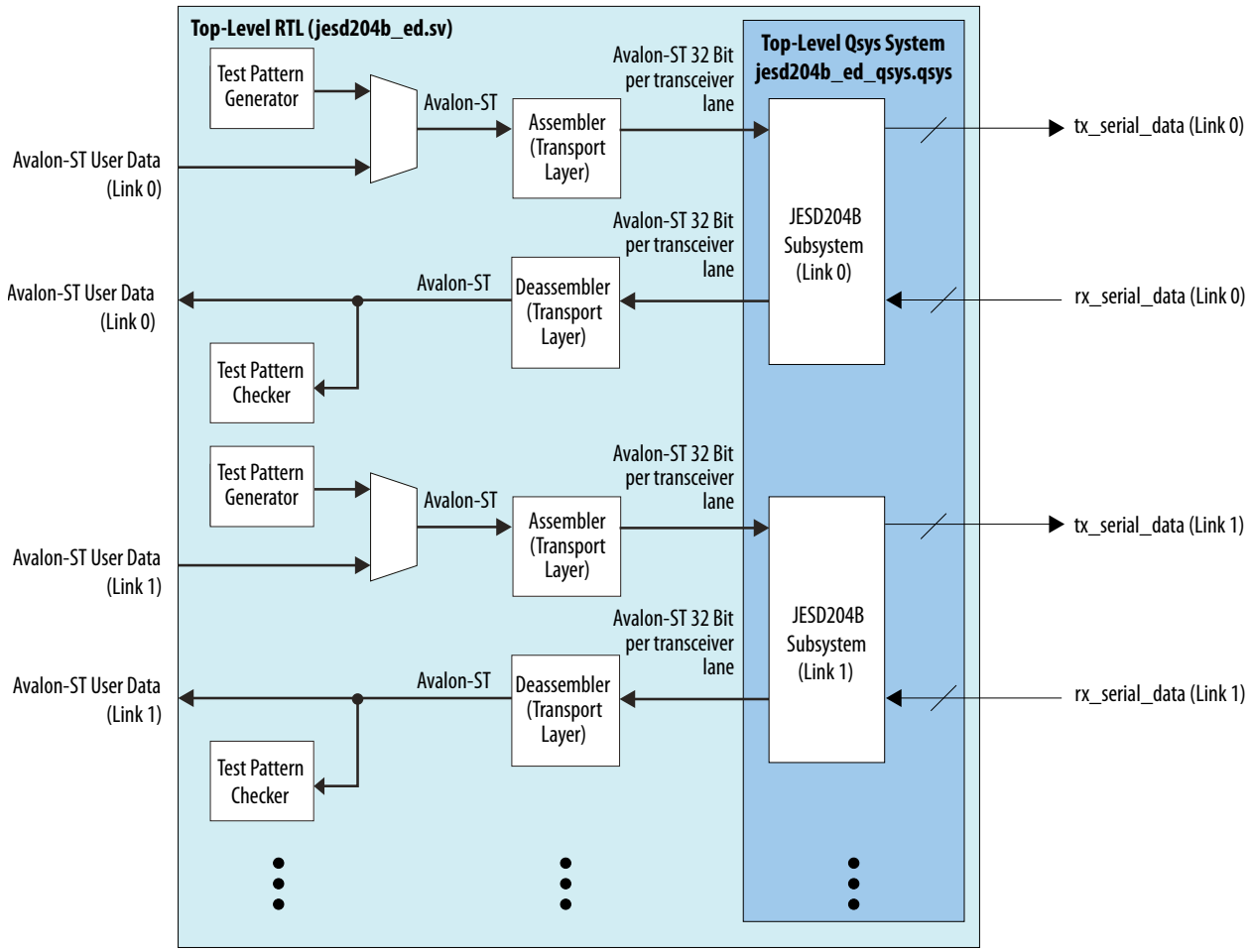
Implementing a Multi-Link Design

The design example Qsys projects, top level HDL file, and software C code are designed for easy implementation of a JESD204B multi-link use case. In the Qsys projects, each link in a JESD204B multi-link use case corresponds to a single instantiation of the **jesd204b_subsystem** module, which includes the JESD204B IP core and other supporting modules. This section assumes that each **jesd204b_subsystem** module in the multi-link design has identical parameter configurations.

In the top level HDL file, each link in a JESD204B multi-link use case corresponds to an instantiation of a transport layer TX and RX pair and a pattern generator and checker pair (assuming duplex data path configuration). The HDL file uses the Verilog `generate` statement using the system parameter `LINK` as an index variable to generate the requisite number of instances for the multi-link use case. This section

assumes that each transport layer TX and RX pair and pattern generator and checker pair in the multi-link design has identical parameter configurations. In the software C code, all software tasks are coded with multi-link capabilities. The *MAX_LINKS* software parameter in the **main.h** header file defines the number of links in the design. In a multi-link scenario, each software action performs an identical task on each link starting with link 0 and proceeding sequentially until the link indicated by the *MAX_LINKS* parameter.

Figure 30: Multi-Link Use Case (Data Path Only) Block Diagram



To implement a multi-link design, you need to perform these procedures:

1. Edit the Qsys project.
2. Edit the top level HDL file.
3. Edit the software C code.

The following sections describe these procedures in detail.

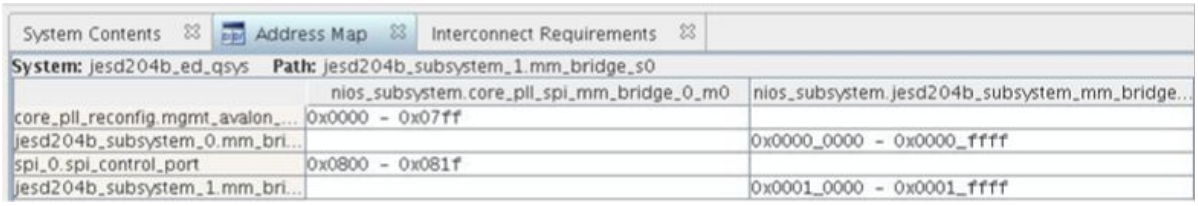
Editing the Qsys Project

1. Open the top level system, `jesd204b_ed_qsys.qsys`, in Qsys.
2. Each JESD204B link is represented by a single jesd204b_subsystem instantiation. To implement multi-links in Qsys, duplicate the jesd204b_subsystem instantiations. Right-click the jesd204b_subsystem_0 module and select **Duplicate**. This duplicates the jesd204b_subsystem_0 module to a new module called jesd204b_subsystem_1.
3. Connect the jesd204b_subsystem_1 ports as shown in the table below. Any ports not described in the table below should be exported. To export a port, click the **Double-click to export** label in the Export column of the System Contents tab.

Ports for jesd204b_subsystem_1 Module	Connection
device_clk	device_clk.clk
do_not_connect_reset_0	mgmt_clk.clk_reset
do_not_connect_reset_1	mgmt_clk.clk_reset
do_not_connect_reset_2	mgmt_clk.clk_reset
frame_clk	frame_clk.clk
jesd204b_jesd204_rx_int	nios_subsystem.nios2_d_irq
jesd204b_jesd204_tx_int	nios_subsystem.nios2_d_irq
link_clk	link_clk.clk
mgmt_clk	mgmt_clk.clk
mgmt_reset	reset_controller_0.reset_out
mm_bridge_s0	nios_subsystem.jesd204b_subsystem_mm_bridge_0_m0
reset_seq_irq	nios_subsystem.nios2_d_irq
reset_seq_pll_reset	*(Do not connect)*
reset_seq_reset_in0	reset_controller_0.reset_out

4. Adjust the interrupt priorities of the interrupt ports (`jesd204b_jesd204_rx_int`, `jesd204b_jesd204_tx_int`, and `reset_seq_irq`) in the new jesd204b_subsystem_1 module to meet your system specifications. Click and edit the priority number of the relevant ports in the IRQ column of the System Contents tab. The lower the priority number, the higher the priority.
5. Assign the address map of the jesd204b_subsystem_1 module in the Address Map tab. Bits 16-19 of the nios_subsystem-to-jesd204b_subsystem Avalon-MM bridge are reserved to support multi-links. Assign the address map according to the figure shown below. Bits 16-19 in the address map denotes the link indicator. For subsequent links, increment the link indicator accordingly. The system can support up to 16 links.

Figure 31: Multi-Link Address Map



6. Repeat steps 2 – 5 for subsequent links in your design.
7. Click **Generate HDL** to generate the design files needed for Quartus compilation.
8. After the HDL generation is completed, click **Finish** to save your Qsys settings and exit the Qsys window.

Editing the Top Level HDL File

1. Open the top level HDL file (`jesd204b_ed.sv`) in any text editor.
2. Modify the *LINK* system parameter to reflect the number of links in your design.
3. Replace the single-link jesd204b_ed_qsys instance with the multi-link instance generated earlier as shown in **Editing the Qsys Project** on page 108.
4. Reconnect all the ports that are similar between the single-link jesd204b_ed_qsys instance and the multi-link instance.
5. The ports that are new in the multi-link jesd204b_ed_qsys instance are associated with the jesd204b_subsystem_1 module. Connect the ports that have the jesd204b_subsystem_1_* prefix in the same manner as shown below:

```
.jesd204b_subsystem_1_jesd204b_txlink_rst_n_reset_n(tx_link_rst_n[1])
```

6. Save the file and compile the design in the Quartus Prime software.

Ensure that any additional pins that are created from the addition of multi-links (for example, tx_serial_data and rx_serial_data pins) have proper pin assignments in the Quartus settings file (`jesd204b_ed.qsf`).

Editing the Software C Code

1. Open the main.h header file in the <*your project*>`/ed_nios/software` directory in any text editor.
2. Change the *MAX_LINKS* parameter to match the number of links implemented in your design and save the file.
3. Compile and execute the C code as described in **Executing the Software C Code** on page 94.

JESD204B IP Core Design Example User Guide Document Revision History

Date	Version	Changes
October 2016	2016.10.31	• Updated supported configuration. • Updated supported parameter values for static reconfiguration. • Added note stating that the Qsys does not allow multiple system with same name in *Nios II Subsystem in Qsys*. • Added steps in *Compiling and Simulating Design*. • Updated document title.
May 2016	2016.05.02	Initial release.