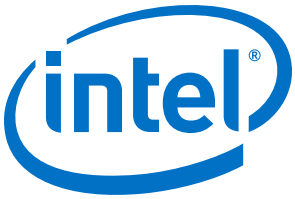


AN 803: Implementing Analog-to-Digital Converter Multi-Link Designs with Intel® Arria® 10 JESD204B RX IP Core

AN-803 | 2018.02.09
Latest document on the web: PDF | HTML

Contents

Implementing Analog-to-Digital Converter Multi-Link Designs with Intel® Arria® 10 JESD204B RX IP Core

This application note provides guidelines to scale up the single link in the JESD204B IP core design example generated from the Intel® Quartus® Prime software to handle multipoint link (multi-link) system. A single link in JESD204B has one or more high speed transceiver lanes or channel.

In some JESD204B applications, multiple analog-to-digital converters (ADC) and digital-to-analog converters (DAC) are used to sample or transmit independent analog signals; hence synchronization is not required. In other applications, there may be requirements to synchronize multiple converters in the array. In these applications, multiple converters interface with single logic devices, such as the Intel Arria® 10 FPGA device. This interface is known as the multipoint link in the JESD204B specification.

Before implementing the multi-link design, you must generate the receiver (RX) single-link design example from the Intel Quartus Prime software. Intel recommends that you perform an RTL simulation on this single link design example to confirm the functionality matches your expectation before transforming the design example to the multi-link design. The guidelines in the following section assume the JESD204B parameters for each link in the multi-link design are identical.

Related Links

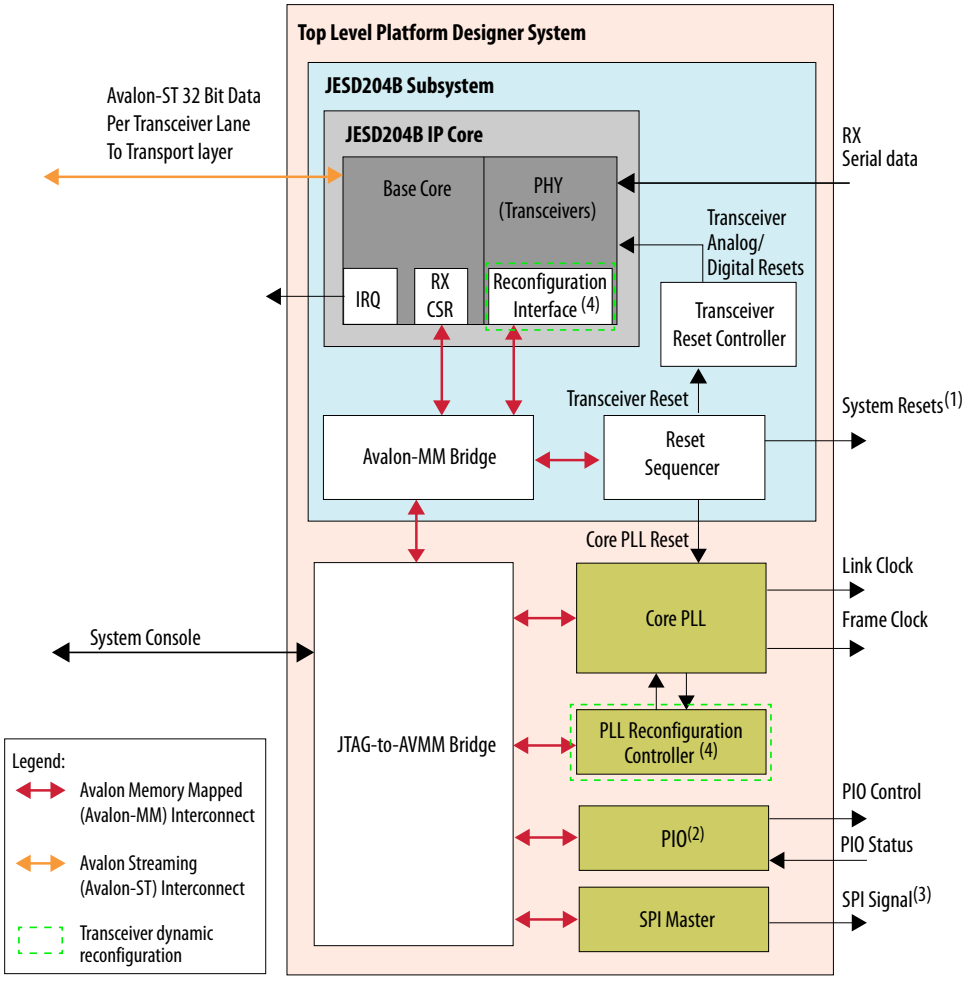
Serial Interface for Data Converters JEDEC Standard: JESD204B.01

Figure 1 —Scope of original JESD204 and revisions A and B in the JESD204B.01 JEDEC Standard visualizes a multi-link system.

ADC-Intel Arria 10 Multi-Link Design Overview

The design example Platform Designer system and top-level HDL file are designed for easy implementation of a JESD204B multi-link use case. In the top-level HDL file, each link in a JESD204B multi-link use case corresponds to an instantiation of a JESD204B IP core, a RX transport layer and a pattern checker. Multi-link design is created by adding multiple JESD204B IP cores, transport layers and pattern checkers to the single link design example. The LINK parameter at the top-level HDL generates multiple transport layers and checkers. You must duplicate JESD204B IP cores in Platform Designer and make connections to the transport layers and pattern checkers. For synchronized multi-link, AND gates are used to combine the synchronization and alignment signals. This section assumes that each RX transport layer and pattern checker in the multi-link design have identical parameter configurations.

Figure 1. Platform Designer System of the Design Example



Notes:

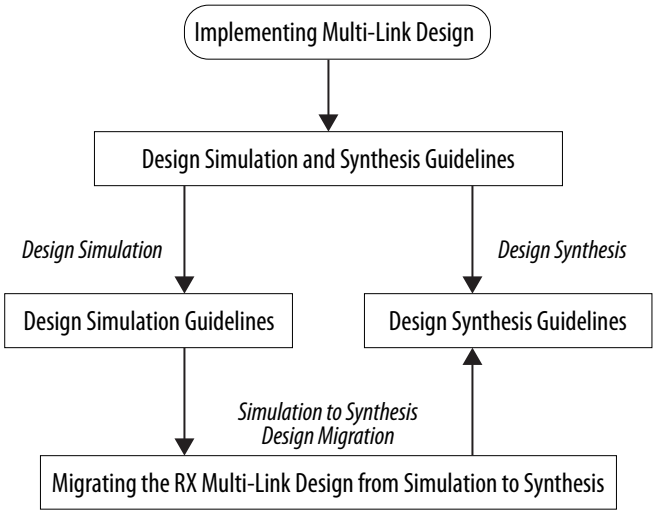
1. System resets comprise the following resets: RX JESD204B IP core CSR resets, RX link resets, RX frame resets.
2. Parallel input/output modules. Parallel 32-bit output for control signals from JTAG to -Avalon master bridge to HDL components. Parallel 32-bit input for status signals from HDL components to JTAG to Avalon master.
3. If Generate 3-Wire SPI Module option is not selected, 4-wire SPI signal to external converter SPI interface. If Generate 3-Wire SPI Module option is selected, 3-wire SPI signal to external converter SPI interface.
4. The core PLL reconfiguration controller and the PHY reconfiguration interface will be generated only when you enable the dynamic reconfiguration option in the IP core.

The *altera_jesd204_subsystem_RX* contains one RX IP core to interface with one ADC. To interface with multiple synchronized converters, the *altera_jesd204_subsystem_RX* should contain multiple IP cores. To interface with multiple unsynchronized converters, you need to instantiate multiple *altera_jesd204_subsystem_RX*, with each subsystem containing one IP core.

ADC-Intel Arria 10 Multi-Link Design Implementation Guidelines

Figure 2. Design Simulation and Synthesis Implementation Guidelines

Follow these set of guidelines to implement your simulation or synthesis ADC-Intel Arria 10 multi-link design.



The following subsections guide you to implement the multi-link design on FPGA. Before you implement the design in the FPGA, you can simulate the design to verify the functionality. You can migrate your simulated design for synthesis and implement the design on the FPGA to interface with ADC. You can follow the synthesis flow guidelines to create the multi-link design for implementation on a FPGA, without performing the simulation.

Here are the steps required to perform simulation and synthesis:

1. Simulation flow:
   a. Generate the single link JESD204B example design in Intel Quartus Prime software.
   b. Simulate the design and confirm the functionality meets your expectation.

      The testbench prints the status of the simulation results.
   c. Modify the RX Platform Designer system to include the additional JESD204B IP cores to form the multi-link.
   d. Modify the RX top-level module to adjust the reset signal connections and to connect the additional JESD204B IP cores to the transport layers and pattern checkers.
   e. Modify the link partner TX Platform Designer system to include the additional JESD204B IP cores to form the multi-link.
   f. Modify the link partner TX top-level module to adjust the reset signal connections and to connect the additional JESD204B IP cores to the transport layers and pattern generators.
   g. Modify the testbench to connect both RX and TX top-level modules.

h. Optionally, you can add signals to the simulation waveform for the additional links in the multi-link design.
i. Elaborate and simulate the multi-link design.
j. Review the simulation results.

2. Synthesis flow:
   a. Generate the single link JESD204B example design in Intel Quartus Prime software.
   b. Optional: Simulate the design and confirm the functionality meets your expectation.

      The testbench prints the status of the simulation results.
   c. Modify the RX Platform Designer system to include the additional JESD204B IP cores to form the multi-link.
   d. Modify the RX top-level module to adjust the reset signals connections and to connect the additional JESD204B IP cores to the transport layers and pattern checkers.
   e. Perform pin assignment in the Intel Quartus Prime assignment editor.
   f. Modify the timing constraint SDC file to include the additional link or links.
   g. Compile the design in Intel Quartus Prime software.
3. Migration flow from simulation to synthesis:
   a. Modify the RX Platform Designer system to replace Avalon-MM master.
   b. Modify the top-level module to adjust the reset signal connections.
   c. Perform pin assignment in the Intel Quartus Prime assignment editor.
   d. Modify the timing constraint SDC file to include the additional link.
   e. Compile the design in Intel Quartus Prime software.

Synchronized ADC-Intel Arria 10 Multi-Link

To synchronize multiple RX IP cores within the Intel Arria 10 device, connect the `dev_lane_aligned` signal from each IP core with an AND gate. The output of the AND gate connects to the `alldev_lane_aligned` port of each IP core. The IP cores need to be out of reset simultaneously to complete the link initialization sequence. Multiple IP cores are put into the same JESD204B subsystem so that each IP core's reset is released by the same reset sequencer simultaneously.

Table 1. IP Core Ports Connection Summary

Table summarizes the IP core ports connection for each Subclass in the multi-link ADCs-Intel Arria 10 interfaces for multi-device synchronization.

Subclass	SYNC_N (ADC-FPGA)	alldev_lane_aligned	Reset	Remark
0	Combined	ANDed and re-distribute	Simultaneous	Refer to Figure 3 on page 7
1	Combined or Non-combined			Refer to Figure 3 on page 7 and Figure 4 on page 7
2				

The SYNC_N signal can be combined or separated within the FPGA. Combining the SYNC_N is mandatory for Subclass 0 subsystem. This is to enable the ADCs to transmit initial lane alignment sequence simultaneously.

For Subclass 1, the SYSREF pulse is the timing reference for the entire JESD204B subsystem. The rise of SYNC_N signals and the transition from code group synchronization (CGS) to initial lane alignment sequence (ILAS) is controlled by the SYSREF. It is important to phase-aligned the SYSREF pulses to the FPGA and converters. Combining SYNC_N is not mandatory unless there is a huge skew beyond one Local Multi-Frame Clock (LMFC) period between the SYNC_N signals to the ADCs.

For Subclass 2, the SYNC_N can be separated and the phase adjustment is between each ADC and IP core pair. Combining SYNC_N has the advantage of ensuring the rise of SYNC_N reaches ADCs at the same time.

Note: For Subclass 0 and Subclass 2 IP cores, you can connect the `sysref` port to ground

Figure 3. Multi-Link Use Case of Synchronized ADCs and FPGA with Combined SYNC_N

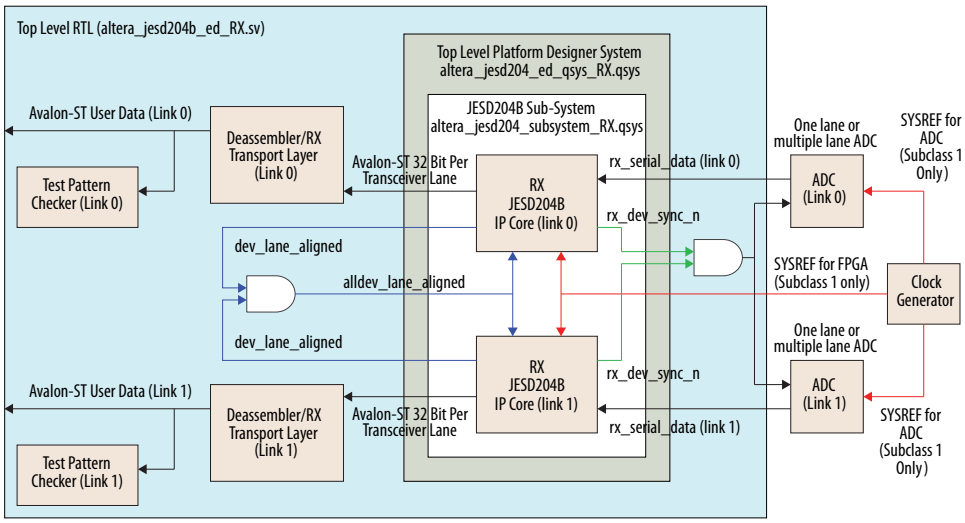


Figure 4. Multi-Link Use Case of Synchronized ADCs and FPGA with Non-combined SYNC_N

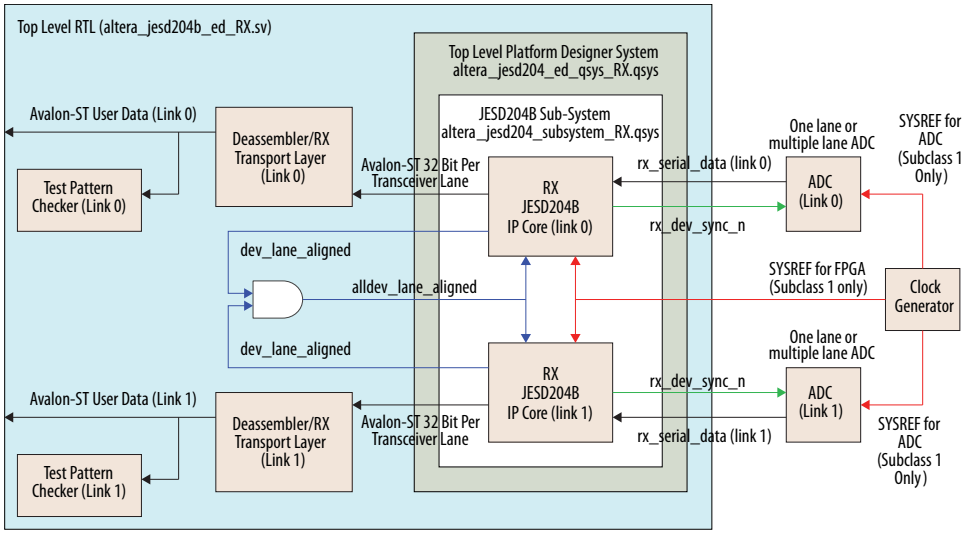
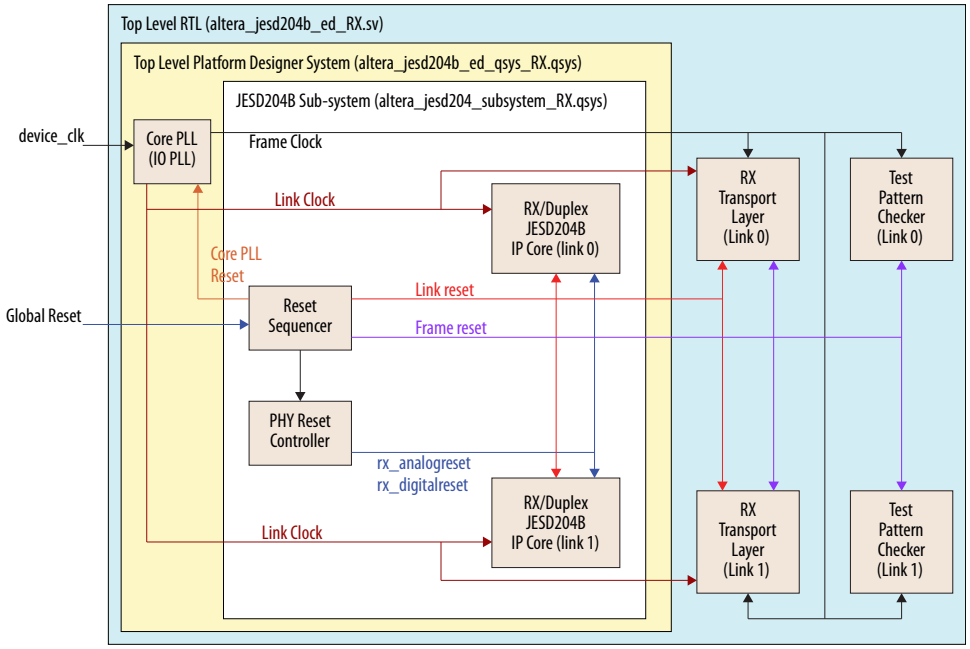


Figure 5. Clock and Reset Scheme of The Synchronized Multi-link



Note: To add JESD204B IP cores for interfacing with more than one ADC, modifications are needed to the Platform Designer system and top-level HDL of the design example.

Related Links

- Synchronized ADC-Arria 10 Multi-link Implementation Design Example Files
  Contains design example files of the synchronized ADC-Arria 10 multi-link described in the following sections. The top level is synthesis design migrated from the simulation design. The ed_sim folder contains the simulation design. The ed_sim/testbench/models folder contains the RX multi-link design and the ed_sim/testbench/sim_models folder contains the link partner TX multi-link design. For more information about the directory structure, refer to the Intel FPGA JESD204B Design Example User Guide for Intel Arria 10 Devices.
- Intel FPGA JESD204B Design Example User Guide for Intel Arria 10 Devices

Design Simulation and Synthesis Guidelines

The following guidelines are for both design simulation and synthesis.

Editing Design Example Platform Designer System for Synchronized ADC-Intel Arria 10 Multi-Link

1. Open the top-level system, `altera_jesd204_ed_qsys_RX.qsys`, in Platform Designer.

   Additional steps for simulation:

   a. The RX `.qsys` file is located at `ed_sim/testbench/models/` folder.

b. To open the `.qsys` file in Platform Designer, you must have an associated Intel Quartus Prime project. Copy the `altera_jesd204_ed_RX.qpf` and `altera_jesd204_ed_RX.qsf` files from the `ed_synth` folder into `ed_sim/testbench/models` folder.

c. Select the `altera_jesd204_ed_RX.qpf` and click **Open**.

d. The **IP Synchronization Result** window opens and click **OK** to proceed.

2. In the **System Contents** tab, right-click the `altera_jesd204_subsystem_RX` instance and select **Drill into Subsystem**. This opens the `altera_jesd204_subsystem_RX` Platform Designer subsystem.

3. Right-click on the `altera_jesd204_RX` component and select **Duplicate**.

   This duplicates the JESD204B IP core. You can rename the duplicated IP core as `altera_jesd204_RX1`.

   Note: Select **No** if the Platform Designer prompts `Do you want to also duplicate the IP Variant file on the disk?`. This is because the duplicated JESD204B IP core has the same parameters as the original JESD204B IP core.

4. Connect the duplicated IP core port as shown in the following table.

Ports for Duplicated IP Core	Connection
`jesd204_rx_avs`	`mm_bridge.m0`
`jesd204_rx_avs_clk`	`mgmt_clk.clk`
`jesd204_rx_avs_rst_n`	`reset_seq.reset_out5`
`rxlink_clk`	`link_clk.clk`
`pll_ref_clk`	`device_clk.clk`
`reconfig_clk`	`mgmt_clk.clk`(1)
`reconfig_reset`	`mgmt_clk.clk_reset`(1)
`reconfig_avmm`	`mm_bridge.m0`(1)

5. Export the rest of the ports to the top-level Platform Designer system by clicking on the **Double-click to export** in the **Export** column of the **System Contents** tab.

6. At the `altera_jesd204_RX` component, disconnect the connections at the following ports. Export them to the top-level Platform Designer system.
   - `alldev_lane_aligned`
   - `dev_lane_aligned`
   - `rx_analogreset`
   - `rx_digitalreset`
   - `rx_cal_busy`
   - `rx_islockedtodata`

(1) Connection is applicable only if you turn on the Enable Transceiver Dynamic Reconfiguration option.

7. Change the number of transceiver channels in the Transceiver PHY reset controller to the total number of transceiver channels of all the JESD204B IP cores in the `altera_jesd204_subsystem_RX`.
8. Exports the following ports from the Transceiver PHY reset controller:
   - `rx_analogreset`
   - `rx_digitalreset`
   - `rx_cal_busy`
   - `rx_is_lockedtodata`
9. At the Address Map, adjust the starting address of `altera_jesd204_RX1` interface so that there is no conflict with other components. For example, you can set the starting address of altera_jesd204_RX1 IP core to `0x000d_0400` as shown in the following table.

Table 2. Synchronized ADC-FPGA Multi-Link Address Map for System Console Control Path (Dynamic reconfiguration for the PHY is disabled)

	mm_bridge.m0
altera_jesd204_RX.jesd204_rx_avs	`0x000d_0000` – `0x000d_03ff`
altera_jesd204_RX1.jesd204_rx_avs	`0x000d_0400` – `0x000d_07ff`

Table 3. Synchronized ADC-FPGA Multi-Link Address Map for System Console Control Path (Dynamic reconfiguration for the PHY is enabled)

	mm_bridge.m0
altera_jesd204_RX.jesd204_rx_avs	`0x000d_0000` – `0x000d_03ff`
altera_jesd204_RX.reconfig_avmm	`0x0000_0000` – `0x0000_1fff`(2)
altera_jesd204_RX1.jesd204_rx_avs	`0x000d_0400` – `0x000d_07ff`
altera_jesd204_RX1.reconfig_avmm	`0x0000_2000` – `0x0000_3fff`(2)

10. Repeat step 3 on page 9 until step 9 on page 10 for subsequent links in your design.
11. Go up to the top-level Platform Designer system.
12. Leave the `xcvr_reset_control_0_pll_powerdown` port unconnected. Export the rest of unconnected ports of altera_jesd204_subsystem_RX instance.
13. Click **Generate HDL** to generate the design files needed for Intel Quartus Prime compilation.

    Additional steps for simulation:

    a. Ensure you select the HDL language of your choice in the **Simulation** section of the **Generation** windows to generate the simulation models.

    b. Click **Generate** and **Yes** to save and generate the design files needed for simulation.

(2) The address span of the PHY reconfiguration interface depends on the number of transceiver channels.

14. After the HDL generation is completed, select **Generate** from the menu. Select **Show Instantiation Template...** and click **Copy**.
15. Paste the instantiation template of `altera_jesd204_ed_qsys_RX` Platform Designer to a text editor.

    You must update the instantiated Platform Designer ports at the top-level HDL.
16. After the HDL generation is completed, click **Finish** to save your Platform Designer settings and exit the Platform Designer window.

Editing Design Example Top-Level HDL for Synchronized ADC-Intel Arria 10 Multi-Link

The generate statement in the Verilog HDL file uses the LINK system parameter as an index variable to generate the requisite number of instances for the multi-link use case.

1. Open the top-level HDL file (`altera_jesd204_ed_RX.sv`) in a text editor.
2. Modify the LINK system parameter to reflect the number of links in your design.
3. Insert the newly exported ports from the Platform Designer at the Platform Designer instantiation.
4. Follow these steps to make the connections for the Platform Designer ports:
   a. For RX link reset, distribute the `rx_link_rst_n[0]` wire from the reset sequencer in Platform Designer to IP cores and transport layers of the second and subsequent links. One way to achieve this is to hard code the index in `rx_link_rst_n[i]` wire in the transport layer instantiations generation loop with `rx_link_rst_n[0]`.
   b. For RX frame reset, distribute the `rx_frame_rst_n[0]` wire from the reset sequencer in Platform Designer to the transport layers and pattern checkers of the second and subsequent links. One way to achieve this is to hard code the index in `rx_frame_rst_n[i]` wire in the transport layer and pattern checker instantiations generation loop with `rx_frame_rst_n[0]`.
   c. Create the following wires to handle PHY resets. Example is shown in Verilog HDL:
      i. `wire [LINK*L-1:0] rx_analogreset;`
      ii. `wire [LINK*L-1:0] rx_digitalreset;`
      iii. `wire [LINK*L-1:0] rx_cal_busy;`
      iv. `wire [LINK*L-1:0] rx_is_lockedtodata;`
   d. Distribute the `PHY reset`, `calibration busy` and `CDR locked` signals from Transceiver PHY Reset Controller equally to each IP core.

      Example: For IP core with two transceiver channels, the `rx_analogreset[1:0]` is distributed to the link 0 IP core and `rx_analogreset[3:2]` is distributed to the link 1 IP core.
   e. Create the following wires for the `dev_lane_aligned` signals. Example is shown in Verilog HDL:
      i. `wire [LINK*L-1:0] dev_lane_aligned;`
      ii. `wire alldev_lane_aligned;`

f. Connect the `dev_lane_aligned` port of each IP core to an AND gate. Distribute the output of the AND gate to `alldev_lane_aligned` port of each IP core.

g. For combined SYNC_N, connect the `dev_sync_n` port of the IP core to an AND gate. Connect the output of the AND gate to `sync_n_out`.

Example:

```
assign sync_n_out = &rx_dev_sync_n;
```

h. For non-combined SYNC_N, scale up the dimension of `sync_n_out` port to match with the number of links.

Example:

```
output wire [LINK-1:0] sync_n_out,
```

i. Leave the following ports unconnected:

- `sof`
- `somf`

Example :

```
.altera_jesd204_subsystem_rx_altera_jesd204_rx1_sof_export
(/*leave open*/),
```

j. For the rest of the ports, increase the index wires from 0 to 1 and subsequent numbers for the subsequent links.

Example: `jesd204_rx_link_data[1]` wire should be connected to link 1 IP core and transport layer.

5. Because there is only one PHY reset controller, the `rx_ready` signal of the subsequent link must be wired to the `rx_ready` signal of link 0 so that the `rx_ready_or_rx_csr_lane_powerdown` signal is connected correctly.

```
generate
    for (i=1; i<LINK; i=i+1) begin: RX_READY
        assign xcvr_rst_ctrl_rx_ready[i] = xcvr_rst_ctrl_rx_ready[0];
    end
endgenerate
```

6. Save the top-level HDL file changes.
7. Additional step for synthesis: Ensure that any additional pins that are created from the addition of multi-links (for example, `rx_serial_data pins`) have proper pin assignments in the Intel Quartus Prime settings file (`altera_jesd204_ed_RX.qsf`).

a. For `sync_n_out` port, add the index to the port as `sync_n_out[0]` for link 0, `sync_n_out[1]` for link 1. Assign the pin number to these ports.

Design Simulation Guidelines

The following guidelines are for design simulation only.

Editing TX Simulation Model Platform Designer System for Synchronized ADC-Intel Arria 10 Multi-Link

1. Open the top-level system, `altera_jesd204_ed_qsys_TX.qsys`, in Platform Designer.
   a. The TX `.qsys` file is located at `ed_sim/testbench/sim_models/` folder.
   b. To open the `.qsys` file in Platform Designer, you must have an associated Intel Quartus Prime project. Copy the `altera_jesd204_ed_RX.qpf` and `altera_jesd204_ed_RX.qsf` files from the `ed_synth` folder into `ed_sim/testbench/sim_models` folder.
   c. Select the `altera_jesd204_ed_RX.qpf` and click **Open**.
   d. The **IP Synchronization Result** window opens and click **OK** to proceed.
2. In the **System Contents** tab, right-click the altera_jesd204_subsystem_TX instance and select **Drill into Subsystem**. This opens the `altera_jesd204b_subsystem_TX.qsys` Platform Designer subsystem.
3. Right-click on the `altera_jesd204_TX` component and select **Duplicate** to duplicate the JESD204B IP core. You can rename the duplicated IP core as `altera_jesd204_TX1`.

   Note: Select **No** if the Platform Designer prompts `Do you want to also duplicate the IP Variant file on the disk?`. This is because the duplicated JESD204B IP core has the same parameters as the original JESD204B IP core.
4. Connect the duplicated IP core port as shown in the following table.

Ports for Duplicated IP Core	Connection
`jesd204_tx_avs`	`mm_bridge.m0`
`jesd204_tx_avs_clk`	`mgmt_clk.clk`
`jesd204_tx_avs_rst_n`	`reset_seq.reset_out2`
`txlink_clk`	`link_clk.clk`

5. Export the rest of the ports to the top-level Platform Designer system. To export a port, click on the **Double-click to export** in the **Export** column of the **System Contents** tab.
6. At the `altera_jesd204_TX` component, disconnect the connections at the following ports. Export them to the top-level Platform Designer system.
   - `dev_sync_n`
   - `mdev_sync_n`
   - `tx_analogreset`
   - `tx_digitalreset`
   - `tx_cal_busy`
7. Change the number of transceiver channels in the Transceiver PHY reset controller to the total number of transceiver channels of all the JESD204B IP cores in the `altera_jesd204_subsystem_TX`.
8. Export the following ports from the Transceiver PHY reset controller:

a. `tx_analogreset`
b. `tx_digitalreset`
c. `tx_cal_busy`

9. At the Address Map, adjust the starting address of `altera_jesd204_TX1` interface so that there is no conflict with other components. For example, you can set the starting address of `altera_jesd204_TX1` IP core to `0x000c_0400` as shown in the following table.

Table 4. Synchronized ADC-FPGA Multi-Link TX Platform Designer Simulation Model Address Map for System Console Control Path (PHY Dynamic Reconfiguration Disabled)

	mm_bridge.m0
altera_jesd204_TX.jesd204_tx_avs	0x000c_0000 – 0x000c_03ff
altera_jesd204_TX1.jesd204_tx_avs	0x000c_0400 – 0x000c_07ff

Table 5. Synchronized ADC-FPGA Multi-Link TX Platform Designer Simulation Model Address Map for System Console Control Path (PHY Dynamic reconfiguration Enabled)

	mm_bridge.m0
altera_jesd204_TX.jesd204_tx_avs	0x000c_0000 – 0x000c_03ff
altera_jesd204_TX.reconfig_avmm	0x0000_0000 – 0x0000_1fff
altera_jesd204_TX1.jesd204_tx_avs	0x000c_0400 – 0x000c_07ff
altera_jesd204_TX1.reconfig_avmm	0x0000_2000 – 0x0000_3fff

10. Repeat step 3 on page 13 until step 9 on page 14 for subsequent links in your design.
11. Go up to the top-level Platform Designer system.
12. Connect the high speed serial clock ports of the duplicated TX IP core to the ATX PLL `mgcb_serial_clk` port. Example of 2 transceiver channels IP core:
    - `altera_jesd204_tx1_tx_serial_clk0_ch0`
    - `altera_jesd204_tx1_tx_serial_clk0_ch1`
13. Export the unconnected ports of `altera_jesd204_subsystem_TX` instance.
14. Click **Generate HDL**.
15. Ensure you select the HDL language of your choice in the **Simulation** section of the **Generation** windows to generate the simulation models.
16. Click **Generate** and **Yes** to save and generate the design files needed for simulation.
17. After the HDL generation is completed, select **Generate** from the menu, select **Show Instantiation Template...** and click **Copy**.
18. Paste the instantiation template of `altera_jesd204_ed_qsys_TX` Platform Designer to a text editor.

You must update the instantiated Platform Designer ports at the top-level HDL.

19. Click **Finish** to save your Platform Designer settings and exit the Platform Designer window.

Editing TX Simulation Model Top-Level HDL for Synchronized ADC-Intel Arria 10 Multi-Link

The generate statement in the Verilog HDL file uses the LINK system parameter as an index variable to generate the requisite number of instances for the multi-link use case.

1. Open the top-level HDL file (`altera_jesd204_ed_TX.sv`) in a text editor.
2. Modify the LINK system parameter to reflect the number of links in your design.
3. Insert the newly exported ports from the Platform Designer at the Platform Designer instantiation.
4. Follow these steps to make the connections for the Platform Designer ports:
   a. For TX link reset, distribute the `tx_link_rst_n[0]` wire from the reset sequencer in Platform Designer to IP cores and transport layers of the second and subsequent links. One way to achieve this is to hard code the index in `tx_link_rst_n[i]` wire in the transport layer instantiations generation loop with `tx_link_rst_n[0]`.
   b. For TX frame reset, distribute the `tx_frame_rst_n[0]` wire from the reset sequencer in Platform Designer to the transport layers and pattern checkers of the second and subsequent links. One way to achieve this is to hard code the index in `tx_frame_rst_n[i]` wire in the transport layer and pattern generator instantiations generation loop with `tx_frame_rst_n[0]`.
   c. Create the following wires to handle PHY resets. Example is shown in Verilog HDL:
      - `wire [LINK*L-1:0] tx_analogreset;`
      - `wire [LINK*L-1:0] tx_digitalreset;`
      - `wire [LINK*L-1:0] tx_cal_busy;`
   d. Distribute the PHY reset and calibration busy signals from Transceiver PHY Reset Controller equally to each IP core.

      Example: For IP core with two transceiver channels, the `tx_analogreset[1:0]` is distributed to the link 0 IP core and `tx_analogreset[3:2]` is distributed to the link 1 IP core.
   e. Create the `mdev_sync_n` wire. Example is shown in Verilog HDL:
      - `wire mdev_sync_n;`
   f. Connect the `dev_sync_n` port of each IP core to an AND gate. Distribute the output of the AND gate to `mdev_sync_n` port of each IP core. You can reuse the `sync_n` wire to connect the `dev_sync_n` port to the AND gate.
   g. For combined `SYNC_N` at the RX subsystem, connect the `sync_n_in` input port to the `sync_n` port of each IP core.
   h. For non-combined `SYNC_N` at the RX subsystem, scale up the dimension of `sync_n_in` port to match with the number of links.

Example:

```
output wire [LINK-1:0] sync_n_in,
```

Assign `sync_n_in[index]` to each IP core with index as the link number. For example, `sync_n_in[0]` for Link 0 IP core, `sync_n_in[1]` for Link 1 IP core.

i. Leave the following ports unconnected:
   - `csr_tx_testpattern_a`
   - `csr_tx_testpattern_b`
   - `csr_tx_testpattern_c`
   - `csr_tx_testpattern_d`
   - `jesd204_tx_dlb_data`
   - `jesd204_tx_dlb_kchar_data`
   - `jesd204_tx_link_ready`
   - `jesd204_tx_somf`

   Example:

```
.altera_jesd204_subsystem_tx_altera_jesd204_tx1_csr_tx_testpattern_a_ex
port   (/*leave open*/),
```

j. For the rest of the ports, increase the index wires from 0 to 1 and subsequent numbers for the subsequent links.

   Example: `jesd204_tx_link_data[1]` wire should be connected to link 1 IP core and transport layer.

5. Because of there is only one PHY reset controller, the `tx_ready` signal of the subsequent link must be wired to the `tx_ready` signal of link 0 so that the `tx_ready_or_tx_csr_lane_powerdown` signal is connected correctly.

```
generate
    for (i=1; i<LINK; i=i+1) begin: TX_READY
        assign xcvr_rst_ctrl_tx_ready[i] = xcvr_rst_ctrl_tx_ready[0];
    end
endgenerate
```

6. Because of there is only one TX PLL for all the IP cores, copy the transceiver PLL locked status pin for link 1 IP core.

```
generate
    for (i=1; i<LINK; i=i+1) begin: XCVR_PLL_LOCKED
        assign xcvr_pll_locked[i] = xcvr_pll_locked[0];
    end
endgenerate
```

7. Save the top-level HDL file changes.

Editing TX Simulation Testbench for Synchronized ADC-Intel Arria 10 Multi-Link

The simulation testbench, `tb_top.sv` is located at `ed_sim/testbench/models` folder. Follow these steps to edit the testbench.

1. Open the testbench (`tb_top.sv`) in a text editor.
2. Add the LINK parameter at the `localparam` declaration section. Example:

```
localparam LINK = 2;     // Number of IP core in the multi-link design
```

3. For non-combined SYNC_N at the RX subsystem, change the dimension of the sync_n wire:

```
wire [LINK-1:0] sync_n;
```

4. Change the dimension and assignment of the following wires and registers:
   - `reg [LINK-1:0] tx_link_error_reg = {LINK{1'b0}};`
   - `reg [LINK-1:0] rx_link_error_reg = {LINK{1'b0}};`
   - `reg [LINK-1:0] data_error_reg = {LINK{1'b0}};`
   - `wire [LINK*L-1:0] tx_serial_data;`
   - `wire [LINK*L-1:0] rx_serial_data;`
   - `wire [LINK-1:0] data_valid;`
   - `wire [LINK-1:0] data_error;`
   - `wire [LINK-1:0] tx_link_error;`
   - `wire [LINK-1:0] rx_link_error;`
5. Create the generation loops for the link error and data error signals:

```
// Pass/Fail Mechanism
   genvar i;
   generate
      for (i=0; i<LINK; i=i+1) begin LINK_ERROR
         // Make sure interrupts do not assert
         always @(posedge mgmt_clk or negedge txlink_rst_n) begin
            if(!txlink_rst_n)
               tx_link_error_reg[i] <= 1'b0;
            else if (tx_link_error[i])
               tx_link_error_reg[i] <= 1'b1;
            else
               tx_link_error_reg[i] <= tx_link_error_reg[i];
         end

         always @(posedge mgmt_clk or negedge rxlink_rst_n) begin
            if(!rxlink_rst_n)
               rx_link_error_reg[i] <= 1'b0;
            else if (rx_link_error[i])
               rx_link_error_reg[i] <= 1'b1;
            else
               rx_link_error_reg[i] <= rx_link_error_reg[i];
         end
      end
   endgenerate

   generate
      for (i=0; i<LINK; i=i+1) begin DATA_ERROR
          always @ (posedge data_error[i]) begin
            if (data_valid[i] == 1'b1)
               data_error_reg[i] <= 1'b1;
            end
      end
   endgenerate
```

6. To monitor the combined simulation results of the multi-link, modify the `test_passed` assignment statement so that if IP core in any of the links has interrupt, the simulation reports failure:

```
 assign test_passed = (&data_error_reg==1'b0) & (&data_valid==1'b1) & ~(|
tx_link_error_reg) & ~(|rx_link_error_reg);
```

7. Configure the test mode for the IP cores and link partners in the subsequent links.

   Note: The address in the BFM write/read task consists of base address + IP core register offset. Refer to the Platform Designer address map to set the address in the BFM write/read task for the IP cores in the subsequent links.

   Example:

```
avalon_mm_csr_sim_model_wr(32'h000C04D0, pat_testmode);    // Link 1 CSR
avalon_mm_csr_dut_wr(32'h000D04D0, pat_testmode);          // Link 1 CSR
```

8. Edit the criteria for displaying link error message for `data_error_reg`, `tx_link_error_reg` and `rx_link_error_reg` signals so that if IP core in any of the links has interrupt, the simulation reports failure. Example:

```
   if (&data_valid) begin
      if (|data_error_reg) begin
         $display("Pattern Checker(s): Data error(s) found!");
      end else begin
         $display("Pattern Checker(s): OK!");
      end
   end else begin
      $display("Pattern Checker(s): No valid data found!");
   end


   if (|tx_link_error_reg) begin
      $display("JESD204B Tx Core(s): Tx link error(s) found!");
   end else begin
      $display("JESD204B Tx Core(s): OK!");
   end

   if (|rx_link_error_reg) begin
      $display("JESD204B Rx Core(s): Rx link error(s) found!");
   end else begin
      $display("JESD204B Rx Core(s): OK!");
   end
```

Adding IP Cores Signals in the Subsequent Links to the Simulation Waveform

Note: This is an optional step.

You can add the signals of the IP cores in subsequent links to the simulation waveform to monitor the link initialization. For ModelSim-Intel FPGA Edition, include the signals of interest into the `tb_top_waveform.do` file at `ed_sim/testbench/mentor` folder. Example:

```
add wave -noupdate -divider {LINK 1}

add wave -noupdate -group {JESD204B BASE CORE 222 L1} {/tb_top/
u_altera_jesd204_ed_RX/u_altera_jesd204_ed_qsys_RX/
altera_jesd204_subsystem_rx/altera_jesd204_rx1/alldev_lane_aligned}

add wave -noupdate -group {RX TRANSPORT L1}
{/tb_top/u_altera_jesd204_ed_RX/GEN_BLOCK[1]/u_jesd204b_transport_rx/csr_f}
```

```
add wave -noupdate -group {PATTERN CHK L1}
{/tb_top/u_altera_jesd204_ed_RX/GEN_BLOCK[1]/u_chk/avst_datain}
```

Updating the Simulation Script

Because of the additional JESD204B IP cores and connection changes in the Platform Designer system, some of the generated Platform Designer filenames are changed. These includes the filenames of the components within the Platform Designer interconnect. The simulation scripts must be updated to avoid elaboration error during the simulation. The simulation script for each simulator is located at the `ed_sim/testbench/setup_scripts` folder.

The ip-make-simscript utility is used to generate the combined simulation scripts for multiple IP cores, custom components and Platform Designer systems. Specify all Simulation Package Descriptor files (`.spd`), each of which lists the required simulation files for the corresponding IP core, custom component or Platform Designer system. The IP parameter editor generates the `.spd` files.

The syntax of this utility is:

```
ip-make-simscript --spd=<your_ip.spd> --spd=<your_ip.spd> --
output_directory=<your directory>
```

Download the design example from Design Store and extract the `jesd204b_sim.sh` for a sample of the shell script that contains the ip-make-simscript utility. The `custom.spd` file contains entries for custom components such as transport layers, RX top-level, TX top-level and testbench.

Follow these steps to generate an updated list of the Platform Designer interconnect components simulation script:

1. Place the `jesd204b_sim.sh` and `custom.spd` files at the `ed_sim/testbench` folder.
2. Back-up the `setup_scripts` folder. The simulation scripts within the `setup_scripts` folder are overwritten when `jesd204b_sim.sh` script is executed.
3. Launch a Nios® II command shell in Intel Quartus Prime software.
4. Navigate to `ed_sim/testbench` folder.
5. At the shell prompt, type the following command:

   ```
   ./jesd204b_sim.sh
   ```

6. The simulation script for each simulator is updated in the `setup_scripts` folder.

Note: If you add custom components to the multi-link design, register your custom components RTL source files into the `custom.spd` file. If you add components to the Platform Designer system, add the `.spd` files paths to the ip-make-simscript utility in the `jesd204b_sim.sh` script.

Related Links

Synchronized ADC-Arria 10 Multi-link Implementation Design Example Files

Simulating the Multi-Link Design

After the modification to the `altera_jesd204_ed_qsys_RX.qsys`, `altera_jesd204_ed_RX.sv`, `altera_jesd204_ed_qsys_TX.qsys`, `altera_jesd204_ed_TX.sv` and `tb_top.sv`, you are ready to simulate the multi-link design using the simulator of your choice. The following example uses ModelSim-Intel FPGA Edition.

1. Launch the ModelSim-Intel FPGA Edition.
2. At the **File** menu, select **Change Directory**.
3. Select `ed_sim/testbench/mentor`.
4. To run the simulation script, type the following command at the transcript prompt:

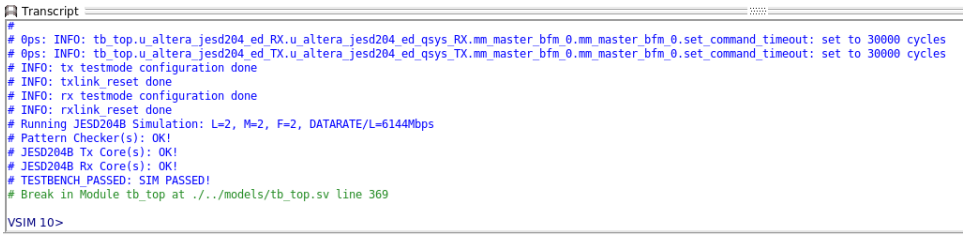
```
do run_tb_top.tcl
```

Viewing the Simulation Results

The simulation testbench prints the results at the transcript or terminal where you execute the simulation script.The following example shows the printout and waveform of the simulation in ModelSim-Intel FPGA Edition:

If the simulation passes, the transcript section prints `TESTBENCH_PASSED: SIM PASSED!` as shown in the following figure.

Figure 6. ModelSim-Intel FPGA Edition Simulation Results Transcript

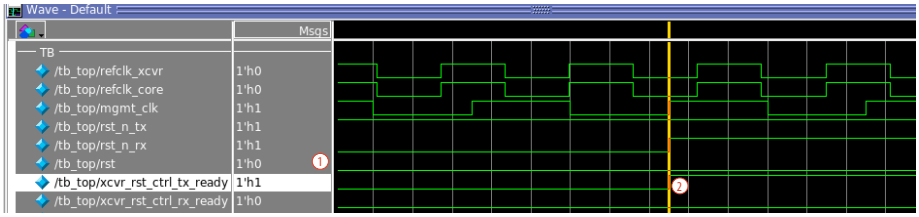


If the simulation fails, the transcript section prints `TESTBENCH_FAILED: SIM FAILED!` along with the failure reason.

If you want to view the waveform, the following are the explanations of a series of events during the link initialization.

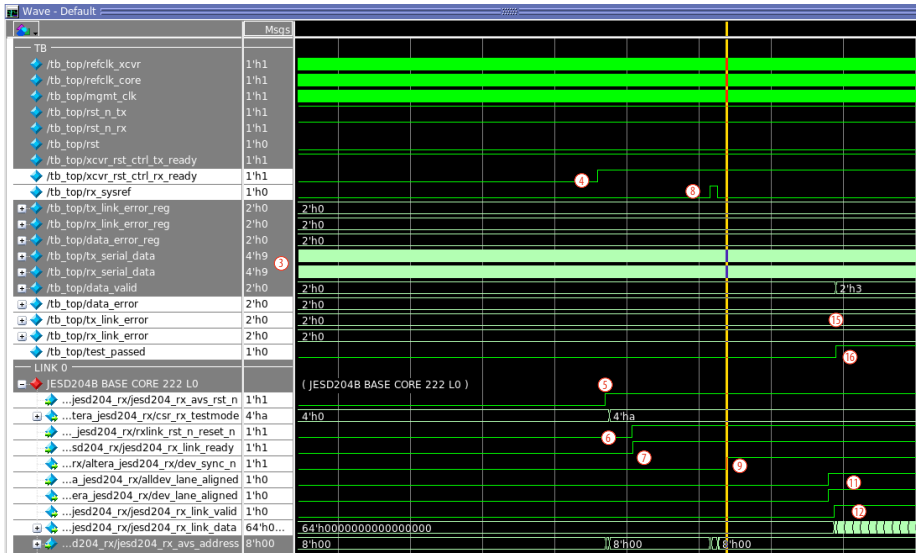
1. After the `/tb_top/rst` is de-asserted, the transceiver reset controller sequences the reset for the transceiver in both Link 0 and Link 1 JESD204B IP core.
2. The `/tb_top/xcvr_rst_ctrl_tx_ready` is asserted when the TX transceiver channels and TX PLL are out of reset.

Figure 7. ModelSim-Intel FPGA Edition Simulation Waveform during Global Reset De-assertion



3. The TX channels send data to RX channels.
4. When the RX channels recovered the data and clock successfully, the `/tb_top/xcvr_rst_ctrl_rx_ready` is asserted.
5. The reset sequencer in the Platform Designer system de-asserts the `jesd204_rx_avs_rst_n` so that the Avalon-MM BFM master configures the test mode CSR.
6. The reset sequencer de-asserts the `rxlink_rst_n_reset_n` and the IP core is out of reset.
7. The RX transport layer asserts the `jesd204_rx_link_ready` to IP core.
8. The testbench sends a `rx_sysref` pulse to the IP core.
9. The IP core de-assert the `dev_sync_n`.
10. After the IP core exits the code group synchronization (CGS) phase, the IP core enters the initial lane alignment sequence (ILAS) phase.
11. The `alldev_lane_aligned` is asserted when both IP cores achieve lane alignment.
12. The `jesd204_rx_link_valid` is asserted when the IP core enters the user data phase and sample data is sent to RX transport layer.
13. The RX transport layer performs the lane mapping of the sample data.
14. The pattern checker checks the received sample data from RX transport layer.
15. No data error is detected by the pattern checker and no interrupt is asserted by both the RX and TX IP cores.
16. The testbench asserts the `test_passed` flag when the above conditions in item 15 are met.

Figure 8. ModelSim-Intel FPGA Edition Simulation Waveform for Successful Link Initialization



Design Synthesis Guidelines

The following guidelines are for design synthesis or when simulation to synthesis design migration.

Editing Design Example Top-Level SDC Constraint for Synchronized ADC-Intel Arria 10 Multi-Link

At the `set_clock_groups` SDC constraint, add the clock domain of the newly added IP core.

Top-level SDC Constraint:

```
set_clock_groups -asynchronous -group {device_clk \
u_altera_jesd204_ed_qsys_RX|core_pll|core_pll|frame_clk \
u_altera_jesd204_ed_qsys_RX|core_pll|core_pll|link_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clkout \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_coreclkin \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_pma_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|<IP core instance
name> |g_xcvr_native_insts[*]|rx_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|<IP core instance
name> |g_xcvr_native_insts[*]|rx_clkout \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|<IP core instance
name> |g_xcvr_native_insts[*]|rx_coreclkin \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|<IP core instance
name> |g_xcvr_native_insts[*]|rx_pma_clk} \
-group {mgmt_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|avmmclk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|<IP core instance
name> |g_xcvr_native_insts[*]|avmmclk}
```

<ip core instance name> is the name for the duplicated copy of the altera_jesd204_RX IP core that you name in 3 on page 9.

The following example has the newly added design entities:

```
set_clock_groups -asynchronous -group {device_clk \
u_altera_jesd204_ed_qsys_RX|core_pll|core_pll|frame_clk \
u_altera_jesd204_ed_qsys_RX|core_pll|core_pll|link_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clkout \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_coreclkin \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_pma_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx1|
g_xcvr_native_insts[*]|rx_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx1|
g_xcvr_native_insts[*]|rx_clkout \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx1|
g_xcvr_native_insts[*]|rx_coreclkin \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx1|
g_xcvr_native_insts[*]|rx_pma_clk} \
-group {mgmt_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|avmmclk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx1|
g_xcvr_native_insts[*]|avmmclk}
```

Compiling the design in Intel Quartus Prime Software

After modifying the Platform Designer system, top-level HDL file and top-level SDC constraint file, compile the design with the Intel Quartus Prime software. Intel recommends that you perform **Analysis and Synthesis** and use the **RTL Viewer** to check the correctness of the connections before fully compile your multi-link design.

Unsynchronized ADC-Intel Arria 10 Multi-Link

For unsynchronized RX IP cores within the Intel Arria 10 FPGA, each IP core operates independently. Because there is no alignment needed between the IP cores, the `dev_lane_aligned` port is connected to the `alldev_lane_aligned` port of each IP core. The IP cores can be out of reset at different time to complete the link initialization sequence. Each IP core is put into the different JESD204B subsystems and reset by the reset sequencers independently.

The combined `SYNC_N` signal is not necessary for unsynchronized ADC interfaces. `SYNC_N` signal from each IP core is connected to corresponding the ADC separately. This connection applies to Subclass 0, 1 and 2 ADC-FPGA pair.

For Subclass 1, the SYSREF pulse is the timing reference for the entire JESD204B subsystem. The rise of `SYNC_N` signals and the transition from code group synchronization (CGS) to initial lane alignment sequence (ILAS) is controlled by SYSREF. It is important to phase-aligned the SYSREF pulses to the FPGA and ADC pair within the same subsystem only. Phase alignment between subsystem is not required.

Figure 9. Multi-Link Use Case of Unsynchronized ADCs and FPGA with Non-combined SYNC_N

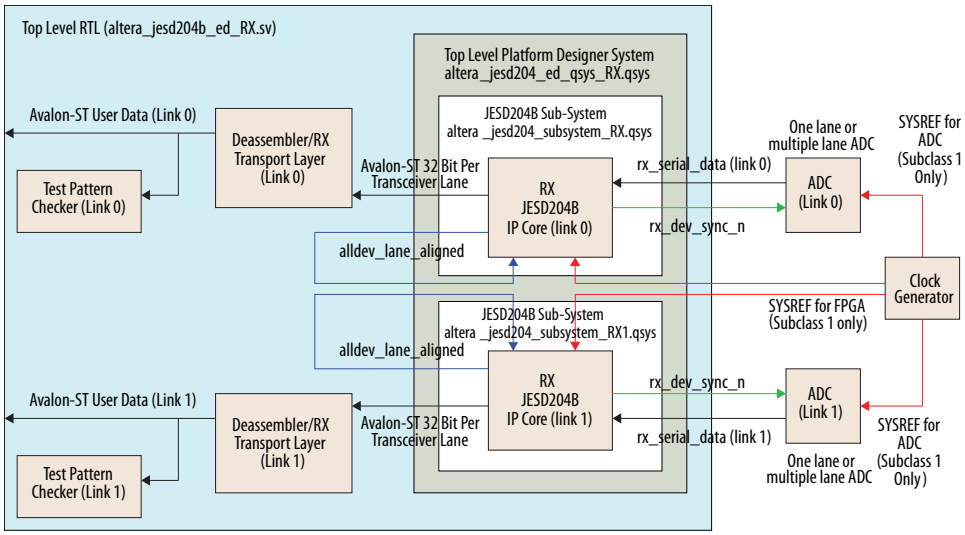
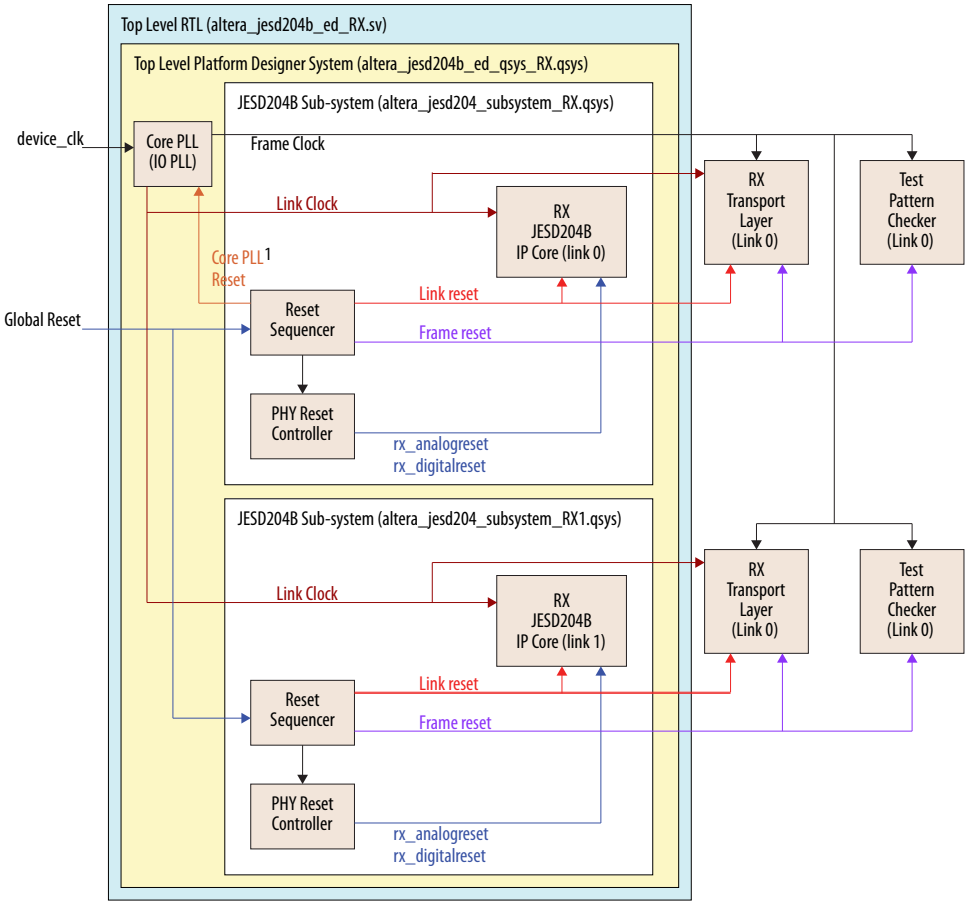


Figure 10. Clock and Reset Scheme of The Unsynchronized Multi-link



Note:
1. Intel recommends that you reset the core PLL once during the link 0 initialization.

Related Links

- Unsynchronized ADC-Arria 10 Multi-link Implementation Design Example Files
  Contains design example files of the unsynchronized ADC-Arria 10 multi-link described in the following sections. The top level is synthesis design migrated from the simulation design. The ed_sim folder contains the simulation design. The ed_sim/testbench/models folder contains the RX multi-link design and the ed_sim/testbench/sim_models folder contains the link partner TX multi-link design. For more information about the directory structure, refer to the Intel FPGA JESD204B Design Example User Guide for Intel Arria 10 Devices.
- Intel FPGA JESD204B Design Example User Guide for Intel Arria 10 Devices

Design Simulation and Synthesis Guidelines

The following guidelines are for both design simulation and synthesis.

Editing Design Example Platform Designer System for Unsynchronized ADC-Intel Arria 10 Multi-Link

1. Open the top-level system, `altera_jesd204_ed_qsys_RX.qsys`, in Platform Designer.

   Additional steps for simulation:

   a. The RX Platform Designer file is located at `ed_sim/testbench/models/` folder.

   b. To open the `.qsys` file in Platform Designer, you must have an associated Intel Quartus Prime project. Copy the `altera_jesd204_ed_RX.qpf` and `altera_jesd204_ed_RX.qsf` files from the `ed_synth` folder into `ed_sim/testbench/models` folder.

   c. Select the `altera_jesd204_ed_RX.qpf` and click **Open**.

   d. The **IP Synchronization Result** window opens and click **OK** to proceed.

2. Each JESD204B link is represented by a single `altera_jesd204_subsystem_RX` instantiation. To implement multi-link in Platform Designer, right click at the `altera_jesd204_subsystem_RX` instantiations and select **Duplicate**.

   You can rename the duplicated module as `altera_jesd204_subystem_RX1`.

3. Connect the `altera_jesd204_subsystem_RX1` ports as shown in the following table.

Ports for altera_jesd204_subsystem_RX1 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
mm_bridge_s0	JTAG_AVMM_Bridge.master (synthesis) mm_master_bfm_0.m0 (simulation)
link_clk	link_clk.clk
mgmt_clk	mgmt_clk.clk
mgmt_reset	reset_controller_0.reset_out
reset_seq_reset_in0	reset_controller_0.reset_out

4. Leave `reset_seq_pll_reset` and `xcvr_reset_control_0_pll_powerdown` ports of the `altera_jesd204_subsystem_RX1` module unconnected.
5. Export the rest of the ports to the top-level Platform Designer system by clicking on the **Double-click to export** in the **Export** column of the **System Contents** tab.
6. Assign the address map of the `altera_jesd204_subsystem_RX1` module in the **Address Map** tab. Assign the address map according to the following table.

Table 6. Unsynchronized ADC-FPGA Multi-Link Address Map for System Console Control Path

Applicable when the dynamic reconfiguration for the PHY is either disabled or enabled.

	mm_master_bfm_0 (simulation) JTAG_AVMM_Bridge.master (synthesis)
altera_jesd204_subsystem_RX.mm_bridge_s0	0x0000_0000 - 0x000f_ffff
altera_jesd204_subsystem_RX1.mm_bridge_s0	0x0010_0000 - 0x001f_ffff

7. Repeat steps 2 on page 26 until step 6 on page 26 for subsequent links in your design.
8. Click **Generate HDL** to generate the design files needed for Intel Quartus Prime compilation.

   Additional steps for simulation:

   a. Ensure you select the HDL language of your choice in the **Simulation** section of the **Generation** windows to generate the simulation models.

   b. Click **Generate** and **Yes** to save and generate the design files needed for simulation.
9. After the HDL generation is completed, select **Generate** from the menu. Select **Show Instantiation Template...** and click **Copy**.
10. Paste the instantiation template of `altera_jesd204_ed_qsys_RX` Platform Designer to a text editor.

    You must update the instantiated Platform Designer ports at the top-level HDL.
11. After the HDL generation is completed, click **Finish** to save your Platform Designer settings and exit the Platform Designer window.

Editing Design Example Top-Level HDL for Unsynchronized ADC-Intel Arria 10 Multi-Link

The generate statement in the Verilog HDL file uses the LINK system parameter as an index variable to generate the requisite number of instances for the multi-link use case.

1. Open the top-level HDL file (`altera_jesd204b_ed_RX.sv`) in a text editor.
2. Modify the LINK system parameter to reflect the number of links in your design.
3. Insert the newly exported ports from the Platform Designer system at the Platform Designer system instantiation.
4. Follow these steps to make the connections for the Platform Designer ports:

   a. For `SYNC_N`, scale up the dimension of `sync_n_out` port to match with the number of links.

   Example:

   ```
   output wire [LINK-1:0] sync_n_out,
   ```

   b. Leave the following ports unconnected:

   - `sof`
   - `somf`
   - `reset_seq_irq`

c. For the rest of the ports, increase the index of the wires from 0 to 1 and subsequent numbers for the subsequent links.

   For example, `jesd204_rx_link_data[1]` wire should be connected to link 1 IP core and transport layer.

5. Save the top-level HDL file changes.
6. Additional step for synthesis: Ensure that any additional pins that are created from the addition of multi-links (for example, `rx_serial_data` pins) have proper pin assignments in the Intel Quartus Prime settings file (`altera_jesd204_ed_RX.qsf`).

   The transport layer and pattern checker are reset by the reset sequencer in the corresponding altera_jesd204_subsystem_RX and altera_jesd204_subsytem_RX1. No modification is needed at the transport layer and pattern checker generation loops.

Design Simulation Guidelines

The following guidelines are for design simulation only.

Editing TX Simulation Model Platform Designer System for Unsynchronized ADC-Intel Arria 10 Multi-Link

1. Open the top-level system, `altera_jesd204_ed_qsys_TX.qsys`, in Platform Designer.
   a. The TX `.qsys` file is located at `ed_sim/testbench/sim_models/` folder.
   b. To open the `.qsys` file in Platform Designer, you must have an associated Intel Quartus Prime project. Copy the `altera_jesd204_ed_RX.qpf` and `altera_jesd204_ed_RX.qsf` files from the `ed_synth` folder into `ed_sim/testbench/sim_models` folder.
   c. Select the `altera_jesd204_ed_RX.qpf` and click **Open**.
   d. The **IP Synchronization Result** window opens and click **OK** to proceed.
2. Disconnect the `xcvr_reset_control_0_pll_cal_busy` port at the `altera_jesd204_subsystem_TX` from the `pll_cal_busy` port at the `xcvr_atx_pll_0` component. Export these 2 ports to the top-level.
3. Each JESD204B link is represented by a single `altera_jesd204_subsystem_TX` instantiation. To implement multi-link in Platform Designer, right click at the `altera_jesd204_subsystem_TX` instantiations and select **Duplicate**.

   You can rename the duplicated module as `altera_jesd204_subystem_TX1`.
4. Connect the `altera_jesd204_subsystem_TX1` ports as shown in the following table.

Ports for altera_jesd204_subsystem_TX1 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

continued...

Ports for altera_jesd204_subsystem_TX1 Module	Connection
frame_clk	frame_clk.clk
mm_bridge_s0	JTAG_AVMM_Bridge.master (synthesis) mm_master_bfm_0.m0 (simulation)
link_clk	link_clk.clk
mgmt_clk	mgmt_clk.clk
mgmt_reset	reset_controller_0.reset_out
reset_seq_reset_in0	reset_controller_0.reset_out

5. Leave `reset_seq_pll_reset` port of the `altera_jesd204_subsystem_TX1` module unconnected.
6. Export the rest of the ports to the top-level Platform Designer system. To export a port, click on the **Double-click to export** in the **Export** column of the **System Contents** tab.
7. At the Address Map, adjust the starting address of `altera_jesd204_subsystem_TX1` interface so that there is no conflict with other components. For example, you can set the starting address of `altera_jesd204_subsystem_TX1` to `0x0010_0000` as shown in the following table.

Table 7. Unsynchronized ADC-FPGA Multi-Link TX Platform Designer Simulation Model Address Map for System Console Control Path

	mm_master_bfm_0.m0 (simulation) JTAG_AVMM_Bridge.master (synthesis)
altera_jesd204_subsystem_TX.mm_bridge_s0	0x0000_0000 – 0x000f_ffff
altera_jesd204_subsystem_TX1.mm_bridge_s0	0x0010_0000 – 0x001f_ffff

8. Repeat step 3 on page 28 until step 7 on page 29 for subsequent links in your design.
9. Connect the high speed serial clock ports of the duplicated TX IP core to the ATX PLL `mgcb_serial_clk` port. Example of 2 transceiver channels IP core:
   - `altera_jesd204_TX_tx_serial_clk0_ch0`
   - `altera_jesd204_TX_tx_serial_clk0_ch1`
10. Click **Generate HDL**.
11. Ensure you select the HDL language of your choice in the **Simulation** section of the **Generation** windows to generate the simulation models.
12. Click **Generate** and **Yes** to save and generate the design files needed for simulation.
13. After the HDL generation is completed, select **Generate** from the menu. Select **Show Instantiation Template...** and click **Copy**.
14. Paste the instantiation template of `altera_jesd204_ed_qsys_TX` Platform Designer to a text editor.

    You must update the instantiated Platform Designer ports at the top-level HDL.
15. Click **Finish** to save your Platform Designer settings and exit the Platform Designer window.

Editing TX Simulation Model Top-Level HDL for Unsynchronized ADC-Intel Arria 10 Multi-Link

The generate statement in the Verilog HDL file uses the LINK system parameter as an index variable to generate the requisite number of instances for the multi-link use case.

1. Open the top-level HDL file (`altera_jesd204_ed_TX.sv`) in a text editor.
2. Modify the LINK system parameter to reflect the number of links in your design.
3. Insert the newly exported ports from the Platform Designer system at the Platform Designer system instantiation.
4. Follow these steps to make the connections for the Platform Designer ports:
   a. For SYNC_N, scale up the dimension of `sync_n_in` port to match with the number of links.

      Example:

      ```
      input wire [LINK-1:0] sync_n_in,
      ```

   b. Leave the following ports unconnected:
      - `csr_tx_testpattern_a`
      - `csr_tx_testpattern_b`
      - `csr_tx_testpattern_c`
      - `csr_tx_testpattern_d`
      - `jesd204_tx_dlb_data`
      - `jesd204_tx_dlb_kchar_data`
      - `jesd204_tx_link_ready`
      - `jesd204_tx_somf`
      - `reset_seq_irq`

   c. For the exported ports, increase the index of the wires from 0 to 1 and subsequent numbers for the subsequent links.

      For example, `jesd204_rx_link_data[1]` wire should be connected to link 1 IP core and transport layer.
5. Follow these steps to make connections for the `pll_cal_busy` port:
   a. Create a wire for the `pll_cal_busy` port:

      ```
      wire       pll_cal_busy;
      ```

b. Distribute the `pll_cal_busy` port from `xcvr_atx_pll_0` to the `xcvr_reset_control_0_pll_cal_busy` port of each `altera_jesd204_subsystem_TX`. At the Platform Designer system instantiation:

```
.xcvr_atx_pll_0_pll_cal_busy_pll_cal_busy   (pll_cal_busy),
.altera_jesd204_subsystem_TX_xcvr_reset_control_0_pll_cal_busy_pll_cal_
busy   (pll_cal_busy),
.altera_jesd204_subsystem_TX1_xcvr_reset_control_0_pll_cal_busy_pll_cal
_busy   (pll_cal_busy),
```

6. Because of there is only one TX PLL for all the IP cores, copy the transceiver PLL locked status pin for link 1 IP core.

```
generate
    for (i=1; i<LINK; i=i+1) begin: XCVR_PLL_LOCKED
        assign xcvr_pll_locked[i] = xcvr_pll_locked[0];
    end
endgenerate
```

7. Save the top-level HDL file changes.

Editing TX Simulation Testbench for Unsynchronized ADC-Intel Arria 10 Multi-Link

The simulation testbench, `tb_top.sv` is located at `ed_sim/testbench/models` folder. Follow these steps to edit the testbench.

1. Open the testbench (`tb_top.sv`) in a text editor.
2. Add the LINK parameter at the `localparam` declaration section. Example:

```
localparam LINK = 2;     // Number of IP core in the multi-link design
```

3. Change the dimension of the `sync_n` wire:

```
wire [LINK-1:0] sync_n;
```

4. Scale-up the dimension of the `u_altera_jesd204_ed_TX.xcvr_rst_ctrl_tx_ready` and `u_altera_jesd204_ed_RX.xcvr_rst_ctrl_rx_ready` ports as the following:

```
assign xcvr_rst_ctrl_tx_ready =
&u_altera_jesd204_ed_TX.xcvr_rst_ctrl_tx_ready[LINK-1:0];
assign xcvr_rst_ctrl_rx_ready =
&u_altera_jesd204_ed_RX.xcvr_rst_ctrl_rx_ready[LINK-1:0];
```

5. Scale-up the dimension of the `u_altera_jesd204_ed_TX.tx_link_rst_n` and `u_altera_jesd204_ed_RX.rx_link_rst_n` ports as the following:

```
assign txlink_rst_n = &u_altera_jesd204_ed_TX.tx_link_rst_n[LINK-1:0];
assign rxlink_rst_n = &u_altera_jesd204_ed_RX.rx_link_rst_n[LINK-1:0];
```

6. Change the dimension and assignment of the following wires and registers:
   - `reg [LINK-1:0] tx_link_error_reg = {LINK{1'b0}};`
   - `reg [LINK-1:0] rx_link_error_reg = {LINK{1'b0}};`
   - `reg [LINK-1:0] data_error_reg = {LINK{1'b0}};`
   - `wire [LINK*L-1:0] tx_serial_data;`

- `wire [LINK*L-1:0] rx_serial_data;`
- `wire [LINK-1:0] data_valid;`
- `wire [LINK-1:0] data_error;`
- `wire [LINK-1:0] tx_link_error;`
- `wire [LINK-1:0] rx_link_error;`

7. Create the generation loops for the link error and data error signals:

```
// Pass/Fail Mechanism
   genvar i;
   generate
      for (i=0; i<LINK; i=i+1) begin LINK_ERROR
         // Make sure interrupts do not assert
         always @(posedge mgmt_clk or negedge txlink_rst_n) begin
            if(!txlink_rst_n)
               tx_link_error_reg[i] <= 1'b0;
            else if (tx_link_error[i])
               tx_link_error_reg[i] <= 1'b1;
            else
               tx_link_error_reg[i] <= tx_link_error_reg[i];
         end

         always @(posedge mgmt_clk or negedge rxlink_rst_n) begin
            if(!rxlink_rst_n)
               rx_link_error_reg[i] <= 1'b0;
            else if (rx_link_error[i])
               rx_link_error_reg[i] <= 1'b1;
            else
               rx_link_error_reg[i] <= rx_link_error_reg[i];
         end
      end
   endgenerate

   generate
      for (i=0; i<LINK; i=i+1) begin DATA_ERROR
          always @ (posedge data_error[i]) begin
            if (data_valid[i] == 1'b1)
               data_error_reg[i] <= 1'b1;
            end
      end
   endgenerate
```

8. To monitor the combined simulation results of the multi-link, modify the `test_passed` assignment statement so that if IP core in any of the links has interrupt, the simulation reports failure:

```
 assign test_passed = (&data_error_reg==1'b0) & (&data_valid==1'b1) & ~(|
tx_link_error_reg) & ~(|rx_link_error_reg);
```

9. Configure the test mode for the IP cores and link partners in the subsequent links.

   Note: The address in the BFM write/read task consists of base address + IP core register offset. Refer to the Platform Designer address map to set the address in the BFM write/read task for the IP cores in the subsequent links.

Example:

```
avalon_mm_csr_sim_model_wr(32'h001C00D0, pat_testmode);    // Link 1 CSR
avalon_mm_csr_dut_wr(32'h001D00D0, pat_testmode);          // Link 1 CSR
```

10. Edit the criteria for displaying link error message for `data_error_reg`, `tx_link_error_reg` and `rx_link_error_reg` signals so that if IP core in any of the links has interrupt, the simulation reports failure. Example:

```
if (&data_valid) begin
   if (|data_error_reg) begin
      $display("Pattern Checker(s): Data error(s) found!");
   end else begin
      $display("Pattern Checker(s): OK!");
   end
end else begin
   $display("Pattern Checker(s): No valid data found!");
end


if (|tx_link_error_reg) begin
   $display("JESD204B Tx Core(s): Tx link error(s) found!");
end else begin
   $display("JESD204B Tx Core(s): OK!");
end

if (|rx_link_error_reg) begin
   $display("JESD204B Rx Core(s): Rx link error(s) found!");
end else begin
   $display("JESD204B Rx Core(s): OK!");
end
```

Adding IP Cores Signals in the Subsequent Links to the Simulation Waveform

Note: This is an optional step.

You can add the signals of the IP cores in subsequent links to the simulation waveform to monitor the link initialization. For ModelSim-Intel FPGA Edition, include the signals of interest into the `tb_top_waveform.do` file at `ed_sim/testbench/mentor` folder. Example:

```
add wave -noupdate -divider {LINK 1}

add wave -noupdate -group {JESD204B BASE CORE 222 L1} {/tb_top/
u_altera_jesd204_ed_RX/u_altera_jesd204_ed_qsys_RX/
altera_jesd204_subsystem_rx/altera_jesd204_rx1/alldev_lane_aligned}

add wave -noupdate -group {RX TRANSPORT L1}
{/tb_top/u_altera_jesd204_ed_RX/GEN_BLOCK[1]/u_jesd204b_transport_rx/csr_f}

add wave -noupdate -group {PATTERN CHK L1}
{/tb_top/u_altera_jesd204_ed_RX/GEN_BLOCK[1]/u_chk/avst_datain}
```

Updating the Simulation Script

Because of the additional JESD204B IP cores and connection changes in the Platform Designer system, some of the generated Platform Designer filenames are changed. These includes the filenames of the components within the Platform Designer interconnect. The simulation scripts must be updated to avoid elaboration error during the simulation. The simulation script for each simulator is located at the `ed_sim/testbench/setup_scripts` folder.

The ip-make-simscript utility is used to generate the combined simulation scripts for multiple IP cores, custom components and Platform Designer systems. Specify all Simulation Package Descriptor files (`.spd`), each of which lists the required simulation files for the corresponding IP core, custom component or Platform Designer system. The IP parameter editor generates the `.spd` files.

The syntax of this utility is:

```
ip-make-simscript --spd=<your_ip.spd> --spd=<your_ip.spd> --
output_directory=<your directory>
```

Download the design example from Design Store and extract the `jesd204b_sim.sh` for a sample of the shell script that contains the ip-make-simscript utility. The `custom.spd` file contains entries for custom components such as transport layers, RX top-level, TX top-level and testbench.

Follow these steps to generate an updated list of the Platform Designer interconnect components simulation script:

1. Place the `jesd204b_sim.sh` and `custom.spd` files at the `ed_sim/testbench` folder.
2. Back-up the `setup_scripts` folder. The simulation scripts within the `setup_scripts` folder are overwritten when `jesd204b_sim.sh` script is executed.
3. Launch a Nios II command shell in Intel Quartus Prime software.
4. Navigate to `ed_sim/testbench` folder.
5. At the shell prompt, type the following command:

   ```
   ./jesd204b_sim.sh
   ```

6. The simulation script for each simulator is updated in the `setup_scripts` folder.

Note: If you add custom components to the multi-link design, register your custom components RTL source files into the `custom.spd` file. If you add components to the Platform Designer system, add the `.spd` files paths to the ip-make-simscript utility in the `jesd204b_sim.sh` script.

Related Links

Unsynchronized ADC-Arria 10 Multi-link Implementation Design Example Files

Simulating the Multi-Link Design

After the modification to the `altera_jesd204_ed_qsys_RX.qsys`, `altera_jesd204_ed_RX.sv`, `altera_jesd204_ed_qsys_TX.qsys`, `altera_jesd204_ed_TX.sv` and `tb_top.sv`, you are ready to simulate the multi-link design using the simulator of your choice. The following example uses ModelSim-Intel FPGA Edition.

1. Launch the ModelSim-Intel FPGA Edition.
2. At the **File** menu, select **Change Directory**.
3. Select `ed_sim/testbench/mentor`.
4. To run the simulation script, type the following command at the transcript prompt:

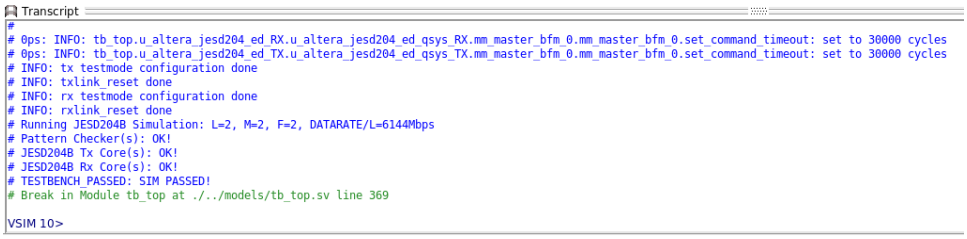
```
do run_tb_top.tcl
```

Viewing the Simulation Results

The simulation testbench prints the results at the transcript or terminal where you execute the simulation script.The following example shows the printout and waveform of the simulation in ModelSim-Intel FPGA Edition:

If the simulation passes, the transcript section prints `TESTBENCH_PASSED: SIM PASSED!` as shown in the following figure.

Figure 11. ModelSim-Intel FPGA Edition Simulation Results Transcript

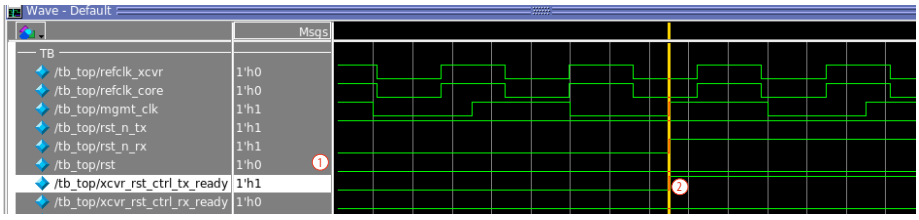


If the simulation fails, the transcript section prints `TESTBENCH_FAILED: SIM FAILED!` along with the failure reason.

If you want to view the waveform, the following are the explanations of a series of events during the link initialization.

1. After the `/tb_top/rst` is de-asserted, the transceiver reset controller sequences the reset for the transceiver in both Link 0 and Link 1 JESD204B IP core.
2. The `/tb_top/xcvr_rst_ctrl_tx_ready` is asserted when the TX transceiver channels and TX PLL are out of reset.

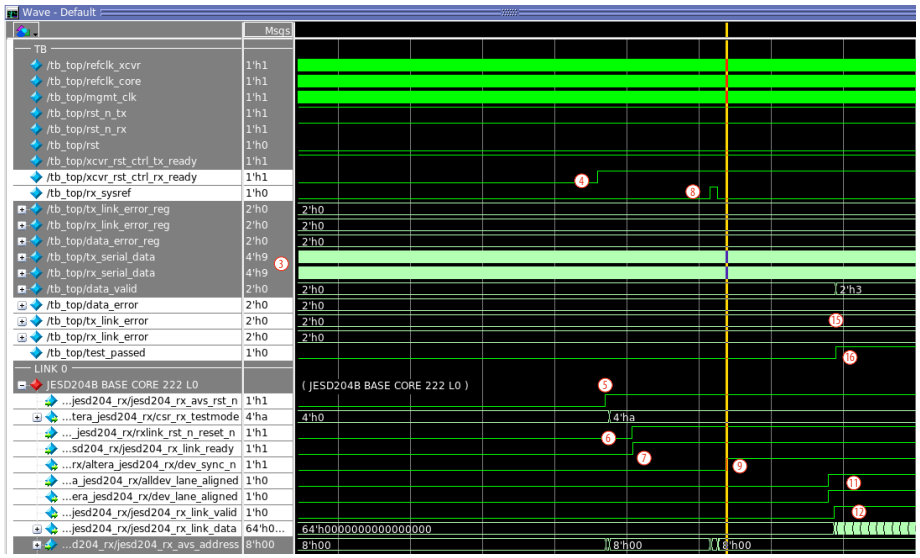
Figure 12. ModelSim-Intel FPGA Edition Simulation Waveform during Global Reset De-assertion



3. The TX channels send data to RX channels.
4. When the RX channels recovered the data and clock successfully, the `/tb_top/xcvr_rst_ctrl_rx_ready` is asserted.

5. The reset sequencer in the Platform Designer system de-asserts the `jesd204_rx_avs_rst_n` so that the Avalon-MM BFM master configures the test mode CSR.
6. The reset sequencer de-asserts the `rxlink_rst_n_reset_n` and the IP core is out of reset.
7. The RX transport layer asserts the `jesd204_rx_link_ready` to IP core.
8. The testbench sends a `rx_sysref` pulse to the IP core.
9. The IP core de-assert the `dev_sync_n`.
10. After the IP core exits the code group synchronization (CGS) phase, the IP core enters the initial lane alignment sequence (ILAS) phase.
11. The `alldev_lane_aligned` is asserted when both IP cores achieve lane alignment.
12. The `jesd204_rx_link_valid` is asserted when the IP core enters the user data phase and sample data is sent to RX transport layer.
13. The RX transport layer performs the lane mapping of the sample data.
14. The pattern checker checks the received sample data from RX transport layer.
15. No data error is detected by the pattern checker and no interrupt is asserted by both the RX and TX IP cores.
16. The testbench asserts the `test_passed` flag when the above conditions in item 15 are met.

Figure 13. ModelSim-Intel FPGA Edition Simulation Waveform for Successful Link Initialization



Design Synthesis Guidelines

The following guidelines are for design synthesis or when simulation to synthesis design migration.

Editing Design Example Top-Level SDC Constraint for Unsynchronized ADC-Intel Arria 10 Multi-Link

At the `set_clock_groups` SDC constraint, add the clock domain of the newly added IP core.

Top-level SDC Constraint:

```
set_clock_groups -asynchronous -group {device_clk \
u_altera_jesd204_ed_qsys_RX|core_pll|core_pll|frame_clk \
u_altera_jesd204_ed_qsys_RX|core_pll|core_pll|link_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clkout \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_coreclkin \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_pma_clk \
u_altera_jesd204_ed_qsys_RX|<subsystem instance name> |altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clk \
u_altera_jesd204_ed_qsys_RX|<subsystem instance name> |altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clkout \
u_altera_jesd204_ed_qsys_RX|<subsystem instance name> |altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_coreclkin \
u_altera_jesd204_ed_qsys_RX|<subsystem instance name> |altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_pma_clk} \
-group {mgmt_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|avmmclk \
u_altera_jesd204_ed_qsys_RX|<subsystem instance name> |altera_jesd204_rx|
g_xcvr_native_insts[*]|avmmclk}
```

<*Subsystem instance name*> is the name for the duplicated copy of the altera_jesd204_subsystem_RX that you name in 2 on page 26.

The following example has the newly added design entities:

```
set_clock_groups -asynchronous -group {device_clk \
u_altera_jesd204_ed_qsys_RX|core_pll|core_pll|frame_clk \
u_altera_jesd204_ed_qsys_RX|core_pll|core_pll|link_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clkout \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_coreclkin \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_pma_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx1|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx1|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_clkout \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx1|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_coreclkin \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx1|altera_jesd204_rx|
g_xcvr_native_insts[*]|rx_pma_clk} \
-group {mgmt_clk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx|altera_jesd204_rx|
g_xcvr_native_insts[*]|avmmclk \
u_altera_jesd204_ed_qsys_RX|altera_jesd204_subsystem_rx1|altera_jesd204_rx|
g_xcvr_native_insts[*]|avmmclk}
```

Compiling the design in Intel Quartus Prime Software

After modifying the Platform Designer system, top-level HDL file and top-level SDC constraint file, compile the design with the Intel Quartus Prime software. Intel recommends that you perform **Analysis and Synthesis** and use the **RTL Viewer** to check the correctness of the connections before fully compile your multi-link design.

Migrating the RX Multi-Link Design from Simulation to Synthesis

Steps to prepare for the simulation to synthesis design migration.

1. Make a copy of the `models` folder. This folder is the working folder to migrate the multi-link design from simulation to synthesis.
2. Copy the following files and folder from the `ed_sim/testbench` folder into the working folder for migration:
   - `pattern`
   - `transport_layer`
   - `spi_3wire.v`
   - `spi_mosi_oe.v`
   - `switch_debouncer.v`
3. Copy the following folders and files from the `ed_synth` folder into the working folder for migration:
   - `system_console`
   - `altera_jesd204_ed_RX.qpf`
   - `altera_jesd204_ed_RX.qsf`
   - `altera_jesd204_ed_RX_assignment_defaults.qdf`
   - `altera_jesd204_ed_RX.sdc`

Migrating RX Platform Designer System for Simulation to RX Platform Designer System for Synthesis

The RX Platform Designer system for simulation uses Avalon-MM BFM to perform write/read transactions to access the IP core CSR. For implementation on hardware, the Avalon-MM BFM is replaced with the JTAG to Avalon Master Bridge component.

1. In the working folder, open the Intel Quartus Prime project, `altera_jesd204_ed_RX.qpf`.
2. Open the top-level Platform Designer system, `altera_jesd204_ed_qsys_RX.qsys`, in Platform Designer.
3. Insert the JTAG to Avalon Master Bridge component and you can rename the component as `JTAG_AVMM_Bridge`.
4. Connect the JTAG to Avalon Master Bridge component ports as shown in the following table.

Ports for Duplicated IP Core	Connection
clk	mgmt_clk.clk
clk_reset	JTAG_reset.out_reset
master	altera_jesd204_subsystem_RX.mm_bridge_s0 altera_jesd204_subsystem_RX1.mm_bridge_s0 (unsynchronized multi-link) pio_control.s1 pio_status.s1 spi_0.spi_control_port

5. Export the `master_reset` port of the JTAG to Avalon Master Bridge component.
6. Export the `in_reset` port of the JTAG_reset component.
7. Disable or delete the Altera Avalon-MM Master BFM component, `mm_master_bfm_0`.
8. Click **Generate HDL**.
9. Click **Generate** and **Yes** to save and generate the design files for compilation.
10. Click **Finish** to save your Platform Designer settings and exit the Platform Designer window.
11. Remove the JTAG to Avalon Master Bridge component, `ip/altera_jesd204_ed_qsys_RX/altera_jesd204_ed_qsys_RX_JTAG_AVMM_Bridge.ip` from the to Intel Quartus Prime setting files.

    This IP file is originated from the `ed_synth` folder. In the current project, JTAG to Avalon Master Bridge has the `altera_jesd204_ed_qsys_RX_master_0.ip` file.

Migrating RX Top-Level HDL for Simulation to RX Top-Level HDL for Synthesis

There are minor differences between the top-level HDL for Simulation and Synthesis. Perform the following steps to migrate the top-level HDL.

1. Add the `jtag_avmm_rst` wire.

   Example:

   ```
   wire   jtag_avmm_rst;
   ```

2. Add `jtag_avmm_rst` signal to the `mgmt_rst_in_n` signal assignment statement. Replace `global_rst_n` signal with `db_global_rst_n` signal:

   ```
   assign mgmt_rst_in_n = db_global_rst_n & ~hw_rst & ~jtag_avmm_rst;
   ```

3. Add JTAG to Avalon-MM Master Bridge reset to the Platform Designer system instantiation.

   Example:

   ```
   .jtag_avmm_bridge_master_reset_reset (jtag_avmm_rst),
   .jtag_reset_in_reset_reset_n (1'b1),
   ```

4. Start Analysis and Synthesis. You can use RTL Viewer to confirm the connections made in the previous steps are correct.

After completing these steps, proceed to perform tasks described in the *Design Synthesis Guidelines*.

Document Revision History for Implementing Analog-to-Digital Converter Multi-Link Designs with Intel Arria 10 JESD204B RX IP Core

Document Version	Changes
2018.02.09	• Added simulation guidelines for synchronized and unsynchronized multi-link design. • Added simulation to synthesis migration guidelines. • Added design simulation and synthesis overview. • Updated instances of Qsys to Platform Designer. • Reorganized document structure. • Updated document title *AN803: Implementing ADC-Intel Arria 10 Multi-Link Design with JESD204B RX IP Core* to *AN803: Implementing Analog-to-Digital Converter Multi-Link Designs with Intel Arria 10 JESD204B RX IP Core*
2017.05.08	Initial release.