AN 814: Intel Arria 10 Two x8-Lane JESD204B (Duplex) IP Cores Multi-Device Synchronization Reference Design

Subscribe
Send Feedback

AN-814 | 2018.01.30
Latest document on the web: PDF | HTML

Contents

1 Intel® Arria® 10 Two x8-Lane JESD204B IP Cores (Duplex) Multi-Device Synchronization Reference Design

This reference design demonstrates the implementation of two x8-lane JESD204B (Duplex) IP cores synchronization in Intel® Arria® 10 device through FMC loopback card. This is done by emulating the interface between one converter card with two x8-lane JESD204B (Duplex) IP cores. The transmitters of the IP cores are emulated to have two analog-to-digital converters (ADC) to form 16 lanes. The serial data from the TX of the IP cores are loopback to the RX through FMC loopback card. Additionally, the design allows internal serial loopback mode for debugging purpose.

The following are the parameters used for the reference design.

- LMF=841
- S=1
- K=32
- Scrambler enabled
- Data rate of the transceiver lanes is 6.0 Gbps in bonding mode (TX)

Related Links

- Reference Design Files for Intel Quartus® Prime Pro Edition
- Reference Design Files for Intel Quartus Prime Standard Edition

1.1 Hardware and Software Requirements

The reference design requires the following hardware and software to run the test:

- Intel Arria 10 GX FPGA Development Board (10AX115S2F45I1SG)
- FMC Loopback Card
- Intel Quartus Prime Standard Edition or Intel Quartus Prime Pro Edition version 17.0
- Nios® II Embedded Design Suite (EDS)

1.2 Reference Design Walkthrough

1.2.1 Hardware Setup

Perform the following steps to setup the hardware for the reference design:

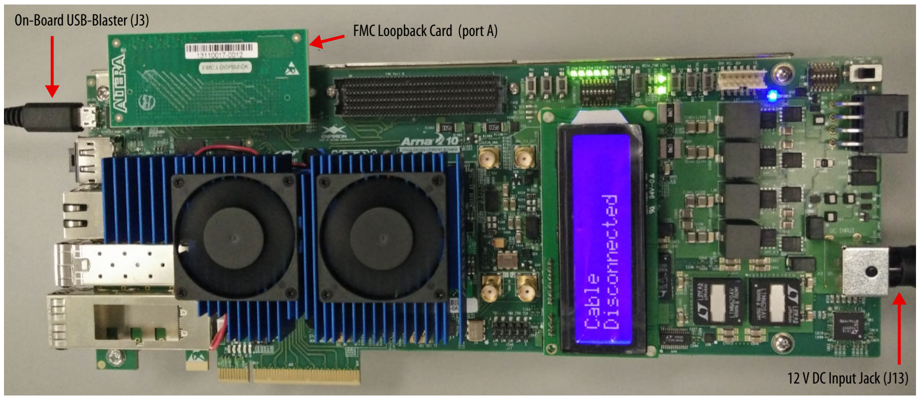
1. Install the FMC Loopback Card module to the FMC port A (J1) on the Intel Arria 10 FPGA development board.
2. Do not change the default switching settings.

Note: For more details about default switching settings, refer to the Intel Arria 10 FPGA development board user guide.

3. Connect the micro-USB cable to the micro-USB connector (J3) on the development board.
4. Connect the power adapter shipped with the development board to power supply jack (J13).
5. Turn On the power for the Intel Arria 10 FPGA development board.

   The hardware system is now ready for programming.

Figure 1. Reference Design Hardware Setup



Related Links

Arria 10 FPGA Development Kit User Guide

1.2.2 Running the Reference Design

Perform the following steps to run the reference design:

1. Configure the FPGA
2. Execute the software C code and initialize the JESD204B link

1.2.2.1 Configuring the FPGA

Perform the following steps to configure the FPGA:

1. To test the reference design targeted Intel Arria 10 device, download the reference design file to your local project directory.
2. Launch the Intel Quartus Prime software.
3. On the **File** menu, click **New Project Wizard**.
4. On the **New Project Wizard** page, open **Design Template Installation**. Select the design template you want install. Click **Next**, then **Finish**.
5. Ensure the following:
   a. The Intel FPGA Download Cable II driver are installed on the host computer
   b. The board is powered.
   c. No other running application is uses the JTAG chain.
6. On the **Tools** menu, click **Programmer**.

7. Click **Auto Detect** to display the devices in the JTAG chain and select a device.
8. Right click and select **Change File**. Then, select the appropriate `jesd204b_ed.sof` file from the `<project directory>/master_image` and click **Open**.
9. Turn on the **Program/Configure** option for the `.sof` file.
10. Click **Start** to download the `.sof` file to the FPGA.

1.2.2.2 Executing the Software C Code and Initializing the JESD204B Link

Perform the following steps to execute the software C code and initialize the JESD204B link:

1. After device programming, navigate to the **Tools** menu and select **Nios II Software Build Tools for Eclipse**.
2. In the **Select a workspace dialog box**, navigate to the software workspace, `<project directory>/software` and click **OK**.
3. To import the software project into Eclipse, right click in the project explorer window and select **Import**.
4. Under the general category select **Existing Projects into Workspace** and click **Next**.
5. Select archive file and navigate to the zip file of interest. Click **Finish** and the project will be setup in your workspace.
6. To download the executable code to the development board, right click on the **jesd204b_nios2_ed** in **Project Explorer** window and select **Run As** and select **Nios II Hardware**.

   Click **Yes** if a dialog box stating "*Errors exist in required project. Continue to launch.*" appears.

   The C codes need to be recompiled if you want to test other supported pattern or internal serial loopback. In this case, you must regenerate the BSP files. In the **Project Explorer** window, right click the **jesd204b_nios2_ed_bsp** project, navigate to Nios II and click **Generate BSP**. This regenerates the BSP files based on your current `jesd204b_ed_qsys.sopcinfo`. Refer to Table 3 on page 6 to change related software parameters.

   Note: Follow steps in Reconstructing Design and Running in Hardware on page 10 to rebuild the Nios II project if you recompile your project or the project doesn't work.

The code performs the JESD204B link initialization sequence and exits after successful download the `.elf` file. You can view the code execution results on the Nios II **Console** tab. The following tables list the expected values of the link status register for each JESD204B IP core.

Table 1. TX Status 0 Registers 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 Sequences	10

continued...

Bit	Name	Description	Expected Binary Value
		10: User Data Mode 11: D21.5 test mode	

Table 2. RX Status 0 Registers 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

Note: You can refer to reconstruct design section if you want to rebuilt the Intel Quartus Prime and Nios II project before running in the hardware.

1.2.3 Software Parameters

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

Table 3. 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 configurations: • 1: TX data path only • 2: RX data path only • 3: Duplex data path (TX and RX data path)
`MAX_LINKS`	1	Set to 1 in this design to indicate a JESD204B subsystem that contains two JESD204B IP cores instances.(1)

continued...

(1) The maximum supported values of MAX_LINKS are 1 for two JESD204B IP instances in a subsystem. You may need to modify Nios II C code and Platform Designer base addresses space if more than 4 JESD204B IP instances in a JESD204B subsystem or multiple JESD204B subsystem.

Parameter	Default Value	Description
MAX_INSTANCES	2	Set to indicate the number of instances in a JESD204B subsystem. Set to 2 in this design to indicate 2 JESD204B instances per subsystem.(2)
LOOPBACK_INIT	0	Initial value of the loopback. Set to 1 for internal serial loopback mode, else set to 0.
SOURCEDEST_INIT	PRBS	Initial value of source/destination. Set to indicate test pattern generator or checker type or user mode: • USER: User mode (no test pattern generator or checker in data path). • ALT: Test pattern generator or checker set in alternate checkerboard mode. • RAMP: Test pattern generator or checker set in ramp wave mode. • PRBS: Test pattern generator or checker set in parallel PRBS mode.

1.2.4 Software Functions Description

This section describes the functions used in `main.c` code and the macros library that facilitates access to configuration and status registers (CSR) of the reference design. These functions and macros provide the building blocks for you to customize the software code based on your system specifications.

1.2.4.1 Functions in main.c Source File

Most of the function prototypes of the sub functions listed in the following table can be found in the `functions.h` or `main.c` header file located in the `software` folder.

Table 4. Functions in main.c Source File

Function Prototype	Description
`int StringIsNumeric (char *)`	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 approximately 1 second.
`int Initialize(char *options[ ] [MAX_OPTIONS_CHAR], int *held_resets)`	Performs the link initialization by executing Test, Sysref, and DelayCounter functions. Return1 if fail.
`int Status(char *options[] [MAX_OPTIONS_CHAR])`	Executes the report link status command according to the options.
`int Loopback (char* options [ ] [MAX_OPTIONS_CHAR ], int *held_resets, int dnr)`	Executes the loopback command according to the options. Returns 0 if success, 1 if fail.
`int SourceDest (char *options[] [MAX_OPTIONS_CHAR], int *held_resets, int dnr)`	Executes source or destination datapath selection command according to the options. Return 0 if success, 1 if fail.

continued...

(2) The maximum supported value of MAX_INSTANCES is 2. You can add up to 4 JESD204B IP cores instances per subsystem without modifying the address space, but you need to modify the Nios II C code.

Function Prototype	Description
`int Test(char *options[][MAX_OPTIONS_CHAR], int *held_resets)`	Executes test mode command according to the options. Test mode: • Set source/destination datapath selection to PRBS/RAMP/ALT test pattern generator/checker. • Set transceiver to internal/external serial loopback. Return 0 if success, 1 if fail.
`void Sysref (void)`	Pulse SYSREF one time
`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 hr, int *held_resets)`	Forces reset assertion or deassertion on submodule resets indicated by `reset_val` for the indicated link. The function also decides whether assert and hold, deassert, or pulse the indicated resets. Returns 0 if success, 1 if fail.
`int Reset_X_L_F_Release(int link, int instances, int *held_resets)`	Deasserts the transceiver, link, and frame resets. The function deasserts the TX transceiver reset first, after the TX transceiver ready signal asserts, it clears the TX error status register. Next, deasserts the TX link and TX frame resets. The function repeats the above actions for the RX side. Return 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 suit your design.
`void ProgCSR(void)`	Programs the JESD204B IP core CSR (e.g. program RBD offset)
`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. Consequently, the ISR clears all valid and active status registers in the `rx_err0` and `rx_err1` registers.
`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. Consequently, the ISR clears all the valid and active status registers in the `tx_err` registers.
`static void ISR_SPI(void * context)`	The SPI master ISR. Upon an interrupt event (IRQ assert), it clears the IRQ flag and return.

1.2.4.2 Custom Peripheral Access Macros in macros.c Source File

You can use the peripheral access macros to access specific information in CSR of the following peripherals:

- Reset sequencer
- JESD204B TX
- JESD204B RX
- PIO control
- PIO status
- Transceiver PHY Reset Controller
- fPLL
- Core PLL Reconfiguration

Table 5. Custom Peripheral Access Macros in macros.c

Most of the function prototypes below can be found in the `macros.h` header file located in the `software` folder.

Function Prototype	Description
`int CALC_BASE_ADDRESS_LINK (int base, int link)`	Calculates and returns the base address based on the link provided. Note the link needs to be 0 because there is only one reset sequencer in the JESD204B subsystem. Hence, set the MAX_LINKS parameter in the main.h header file to 1. You are responsible to set the parameter correctly to reflect the system configuration.
`int CALC_BASE_ADDRESS_XCVR_NATIVE (int base, int instance)`	Calculate and returns the base address of transceiver reconfiguration based on the instance number of JESD204B IP core. The number of transceiver per JESD204B subsystem in the design is defined by MAX_INSTANCES parameter in the main.h header file. This function is not used due to the design doesn't perform data rate reconfiguration.
`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, int instance)`	Read the JESD204B TX CSR `tx_status0` register at link and instance and return the value.
`int IORD_JESD204_TX_SYNCN_SYSREF_CTRL_REG (int link, int instance)`	Read the JESD204B TX CSR `syncn_sysref_ctrl` register at link and instance and return the value.
`void IOWR_JESD204_TX_SYNCN_SYSREF_CTRL_REG (int link, int instance, int val)`	Write val value into the JESD204B TX CSR `syncn_sysref_ctrl` register at link and instance.
`int IORD_JESD204_RX_STATUS0_REG (int link, int instance)`	Read JESD204B CSR `rx_status0` register at link and instance and return value.
`int IORD_JESD204_RX_SYNCN_SYSREF_CTRL_REG (int link, int instance)`	Read JESD204B RX CSR `syncn_sysref_ctrl` register at link and instance and return value.
`void IOWR_JESD204_RX_SYNCN_SYSREF_CTRL_REG (int link, int instance, int val)`	Write val value into JESD204B RX CSR `syncn_sysref_ctrl` register at link and instance.
`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 the value.
`int IORD_JESD204_TX_TEST_MODE_REG (int link, int instance)`	Read the JESD204B TX CSR `tx_test` register at link and instance and return the value.
`int IORD_JESD204_RX_TEST_MODE_REG (int link, int instance)`	Read the JESD204B RX CSR `rx_test` register at link and instance and return the value.
`void IOWR_JESD204_TX_TEST_MODE_REG (int link, int instance, int val)`	Write val value into the JESD204B TX CSR `tx_test` register at link and instance.
`void IOWR_JESD204_RX_TEST_MODE_REG (int link, int instance, int val)`	Write val value into the JESD204B RX CSR `rx_test` register at link and instance.

continued...

Function Prototype	Description
`int IORD_JESD204_RX_ERR0_REG (int link, int instance)`	Read the JESD204B RX CSR `rx_err0` register at link and instance and return the value.
`void IOWR_JESD204_RX_ERR0_REG (int link, int instance, int val)`	Write val value the JESD204B `rx_err0` register at link and instance.
`int IORD_JESD204_RX_ERR1_REG (int link, int instance)`	Read the JESD204B RX CSR `rx_err1` register at link and instance and return the value.
`void IOWR_JESD204_RX_ERR1_REG (int link, int instance, int val)`	Write val value the JESD204B `rx_err1` register at link and instance.
`int IORD_JESD204_TX_ERR_REG (int link, int instance)`	Read the JESD204B TX CSR `tx_err` register at link and instance and return the value.
`void IOWR_JESD204_TX_ERR_REG (int link, int instance, int val)`	Write val value into the JESD204B TX CSR `tx_err` register at link an instance.
`int IORD_JESD204_TX_ERR_EN_REG (int link, int instance)`	Read the JESD204B TX CSR `tx_err_enable` register at link and instance and return the value.
`void IOWR_JESD204_TX_ERR_EN_REG (int link, int instance, int val)`	Write val value into the JESD204B TX CSR `tx_err_enable` register at link and instance.
`int IORD_JESD204_RX_ERR_EN_REG (int link, int instance)`	Read the JESD204B RX CSR `rx_err_enable` register at link and instance and return the value.
`void IOWR_JESD204_RX_ERR_EN_REG (int link, int instance, int val)`	Write the JESD204B RX CSR `rx_err_enable` register at link and instance.

1.2.5 Reconstructing Design and Running in Hardware

Perform the following steps to reconstruct the design and run it in hardware:

1. Regenerating files and configuring the FPGA.
2. Rebuilding Nios II software and initializing the JESD204B link.

1.2.5.1 Regenerating Files, Recompiling Design and Configuring the FPGA

Perform the following steps to regenerate HDL files, compile, generate programming file, and configure the FPGA:

1. Download the reference design file to your local project directory.
2. Launch the Intel Quartus Prime software.
3. On the **File** menu, click **New Project Wizard**.
4. On the **New Project Wizard** page, open **Design Template Installation**. Select the design template you want install. Click **Next**, then **Finish**.

5. *Note:* The reference design provided does not enable Signal Tap Logic Analyzer in the Intel Quartus Prime project. Remove the following commands in `jesd204b_ed.sdc` file to avoid warnings if you do not compile the design with the provided Signal Tap file:
   - Intel Quartus Prime Standard Edition
     —
     — u_jesd204b_ed_qsys|jesd204b_subsystem_0|jesd204b_duplex0|g_xcvr_native_insts[*]|tx_clkout \
     — u_jesd204b_ed_qsys|jesd204b_subsystem_0|jesd204b_duplex1|g_xcvr_native_insts[*]|tx_clkout \
   - Intel Quartus Prime Pro Edition
     — u_jesd204b_ed_qsys|jesd204b_subsystem_0|jesd204b_duplex0|jesd204b_duplex0|g_xcvr_native_insts[*]|tx_clkout \
     — u_jesd204b_ed_qsys|jesd204b_subsystem_0|jesd204b_duplex1|jesd204b_duplex1|g_xcvr_native_insts[*]|tx_clkout \

   To include the Signal Tap file into Intel Quartus Prime project, navigate to the **Assignments** menu and select **Settings**. Click on the Signal Tap Logic Analyzer under **Category**, and then check the **Enable Signal Tap Logic Analyzer** checkbox. Browse the `stp1.stp` file (located at `/stp` directory) and hit **OK** button.
6. To regenerate HDL files and compile the design with or without Signal Tap file,
   a. Open the Intel FPGA GPIO parameter editor (`se_outbuf_1bit.qsys`) and click **Generate HDL**.
   b. Repeat step a. to generate HDL files for `jesd204b_ed_qsys.qsys` and `dl_count.qsys`.
   c. Once the Platform Designer generation are done for all system files, navigate to the **Processing** menu and select **Start Compilation**.
7. Ensure the following:
   a. The Intel FPGA Download Cable II driver are installed on the host computer.
   b. The board is powered.
   c. No other running application is uses the JTAG chain.
8. On the **Tools** menu, click **Programmer**.
9. Click **Auto Detect** to display the devices in the JTAG chain and select a device.
10. Right click and select **Change File**. Then, select the appropriate `jesd204b_ed.sof` file from the `<project directory>/output_files` and click **Open**.
11. Turn on the **Program/Configure** option for the `.sof` file.
12. Click **Start** to download the `.sof` file to the FPGA.

1.2.5.2 Rebuilding Software and Initializing the JESD204B Link

Perform the following steps to rebuild Nios II software, re-execute the software C code and initialize the JESD204B link:

1. After device programming, navigate to the **Tools** menu and select **Nios II Software Build Tools for Eclipse**.
2. In the **Select a workspace** dialog box, navigate to the software workspace, `<project directory>/software` and click **OK**.
3. On the **File** menu, navigate to **New** and click **Nios II Application and BSP From Template**.
4. In the **Nios II Application and BSP From Template** window, enter the following information:
   a. SOPC Information File Name:
      - Intel Quartus Prime Pro Edition: `<project directory>/jesd204b_ed_qsys/jesd204b_ed_qsys.sopcinfo`
      - Intel Quartus Prime Standard Edition: `<project directory>/jesd204b_ed_qsys.sopcinfo`

   b. Project name: `jesd204b_nios2_ed`

      Note: Project name is user customizable.

   c. User default location: **Checked**

   d. Templates: **Blank Project**
5. Click **Next**. Verify that the default BSP name is `jesd204b_nios2_ed_bsp` and click **Finish**. The Nios II application project (`jesd204b_nios2_ed`) and BSP (`jesd204b_nios2_ed_bsp`) appears in the **Project Explorer** window.

   Note: Whenever you regenerate the `jesd204b_ed_qsys.qsys`, you must regenerate the BSP files. In the **Project Explorer** window, right click the `jesd204b_nios2_ed_bsp` project, navigate to **Nios II** and click **Generate BSP**. This regenerates the BSP files based on your most current `jesd204b_ed_qsys.sopcinfo`.
6. Import the design example source (`*.c`) and header (`*.h`) files into the application directory. In the **Project Explorer** window, right click on the `jesd204b_nios2_ed` project and click **Import**.
7. In the **Import** window, select **General** > **File System** as the import source. Click **Next**.
8. Browse to the `<project directory>/software/source directory` and click **OK**. Check the source box on the left panel. Verify that the list of source and header files are as follows:
   - altera_jesd204_regs.h
   - functions.h
   - macros.h
   - macros.c
   - main.h
   - main.c

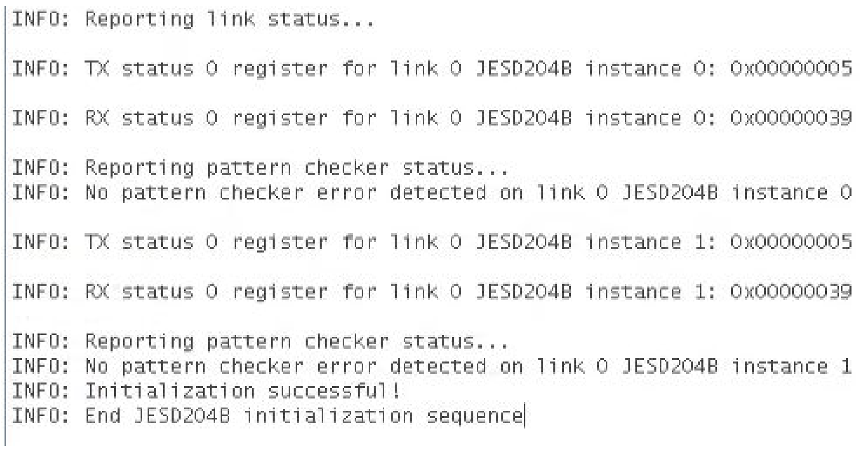
9. Verify that the destination folder is `jesd204b_nios2_ed`. Click **Finish**. All the source and header files should be imported into the `jesd204b_nios2_ed` project directory.
10. Right click the `jesd204b_nios2_ed_bsp` project, navigate to **Nios II and click BSP Editor**. Under the **Drivers** tab, 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 JTAG UART module. After the BSP files have been generated, click **Exit**.
11. Expand the jesd204b_nios2_ed application project in the **Project Explorer** window and verify that 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, right click on the jesd204b_nios2_ed in **Project Explorer** window and select **Run As** and then **Nios II Hardware**.
14. Refer to Table 3 on page 6 to change `SOURCEDEST_INIT` parameter if you want to test other supported pattern. Change the `LOOPBACK_INIT` parameter to 1 if you want test internal serial loopback. Next, repeat 12 on page 13 to 13 on page 13.

1.2.6 Viewing the Results

Perform the followings steps to confirm the link for JESD204B instances 0 and 1 up successfully.

1. In the Nios II console, you should observe it reports no pattern check error detected on JESD204B instances 0 and 1 (subsystem 0 in HDL) shown in the following figure. You can refer to Table 1 on page 5 and Table 2 on page 6 for the expected value of TX/RX status 0 registers for JESD204B instances 0 and 1.

Figure 2. Pass or Fail Indication in Nios II Console



```
INFO: Reporting link status...

INFO: TX status 0 register for link 0 JESD204B instance 0: 0x00000005

INFO: RX status 0 register for link 0 JESD204B instance 0: 0x00000039

INFO: Reporting pattern checker status...
INFO: No pattern checker error detected on link 0 JESD204B instance 0

INFO: TX status 0 register for link 0 JESD204B instance 1: 0x00000005

INFO: RX status 0 register for link 0 JESD204B instance 1: 0x00000039

INFO: Reporting pattern checker status...
INFO: No pattern checker error detected on link 0 JESD204B instance 1
INFO: Initialization successful!
INFO: End JESD204B initialization sequence
```

2. You should observe LEDs D3-D7 illuminate while LEDs D8-D10 off after successful bring up the links.

Figure 3. On-Board User LED

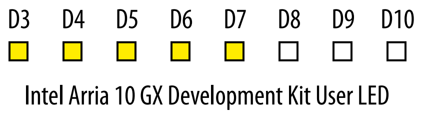
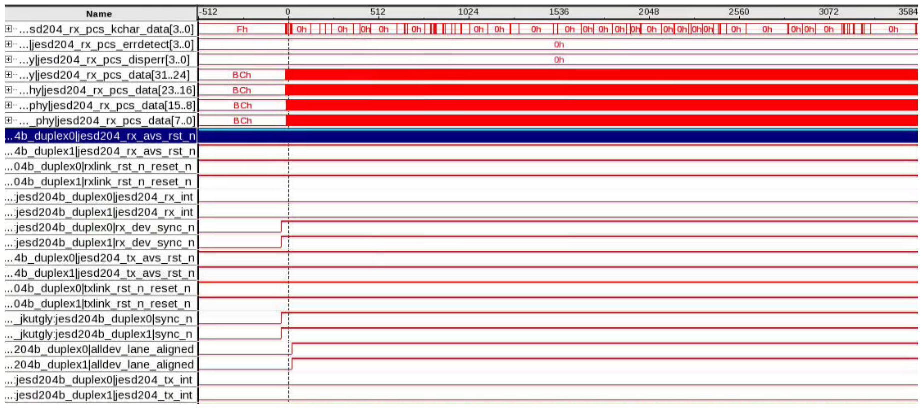


Table 6. Reference Design Status LED on the Intel Arria 10 GX Development Kit

LED Board Reference	Signal Name	Description
D3	`frame_rst_n`	Transport layer, test pattern generator and checker reset signal. The LED illuminates to indicate that the transport layer, test pattern generator and checker are out of reset.
D4	`link_rst_n`	JESD204B IP cores link layer, transport layer link interfaces reset signal. The LED illuminates to indicate that the JESD204B IP cores link layer, and transport layer link interfaces are out of reset.
D5	`sync_n`	SYNC_N from the JESD204B IP cores receiver. The LED illuminates to indicate it successfully received K28.5 characters.
D6	`alldev_lane_aligned`	All lanes align signal for JESD204B IP cores instance 0 and 1. The LED illuminates to indicate all lanes are aligned.
D7	`rx_dev_sync_n_out`	Single AND SYNC~ signals from JESD204B IP cores instance 0 and 1 to the transmitter. The LED illuminates to indicate that the transmitter has received sufficient K28.5 characters.
D8	`jesd204_rx_int`	The link layer RX interrupt status signal from both JESD204B IP cores instance 0 & 1. The LED illuminates to indicate a RX interrupt.
D9	`jesd204_tx_int`	The link layer TX interrupt status signal from both JESD204B IP cores instance 0 & 1. The LED illuminates to indicate a TX interrupt.
D10	`avst_patchk_data_error`	Single AND error status signal for PRBS, RAMP or ATL checkers for JESD204B IP cores instance 0 & 1. The LED illuminates to indicate a data error.

3. You can also observe the behavior of `frame_rst_n`, `link_rst_n`, `sync_n`, `alldev_lane_aligned`, `rx_dev_sync_n`, `jesd204_rx_int`, `jesd204_tx_int` and `avst_pathchk_data_error` signals through Signal Tap file to verify the links up successfully.

Figure 4. Example of the Expected Signal Tap Waveform (rx_link instance)


Name
-512
0
512
1024
1536
2048
2560
3072
3584
...sd204_rx_pcs_kchar_data[3..0]
...|jesd204_rx_pcs_errdetect[3..0]
...y|jesd204_rx_pcs_disperr[3..0]
...y|jesd204_rx_pcs_data[31..24]
...hy|jesd204_rx_pcs_data[23..16]
...phy|jesd204_rx_pcs_data[15..8]
..._phy|jesd204_rx_pcs_data[7..0]
...4b_duplex0|jesd204_rx_avs_rst_n
...4b_duplex1|jesd204_rx_avs_rst_n
...04b_duplex0|rxlink_rst_n_reset_n
...04b_duplex1|rxlink_rst_n_reset_n
...jesd204b_duplex0|jesd204_rx_int
...jesd204b_duplex1|jesd204_rx_int
...jesd204b_duplex0|rx_dev_sync_n
...jesd204b_duplex1|rx_dev_sync_n
...4b_duplex0|jesd204_tx_avs_rst_n
...4b_duplex1|jesd204_tx_avs_rst_n
...04b_duplex0|txlink_rst_n_reset_n
...04b_duplex1|txlink_rst_n_reset_n
..._jkutgly:jesd204b_duplex0|sync_n
..._jkutgly:jesd204b_duplex1|sync_n
...204b_duplex0|alldev_lane_aligned
...204b_duplex1|alldev_lane_aligned
...jesd204b_duplex0|jesd204_tx_int
...jesd204b_duplex1|jesd204_tx_int
Fh
0h
BCh

In the output receiver transport layer, expect to see the PRBS, RAMP, ALT pattern based on the pattern source defined in `main.h` file.

Figure 5. Example of the Expected Signal Tap Waveform (trpt instance)

For example, you should see the pattern as indicated in the following figure if you set the `SOURCEDEST_INIT` in `main.h` file as RAMP pattern.

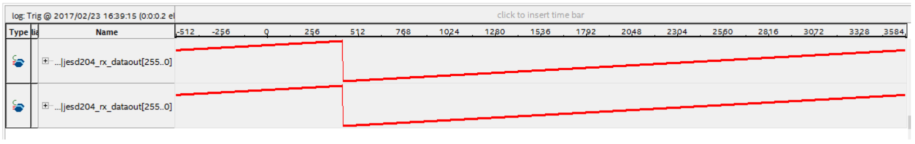
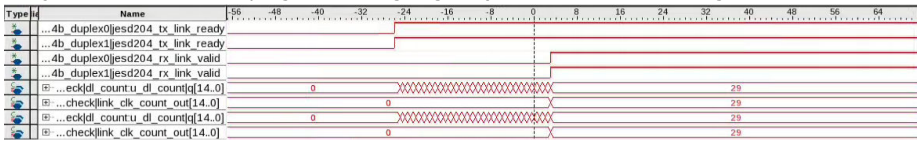


Figure 6. Link Latency Measurement for RBD offset=5

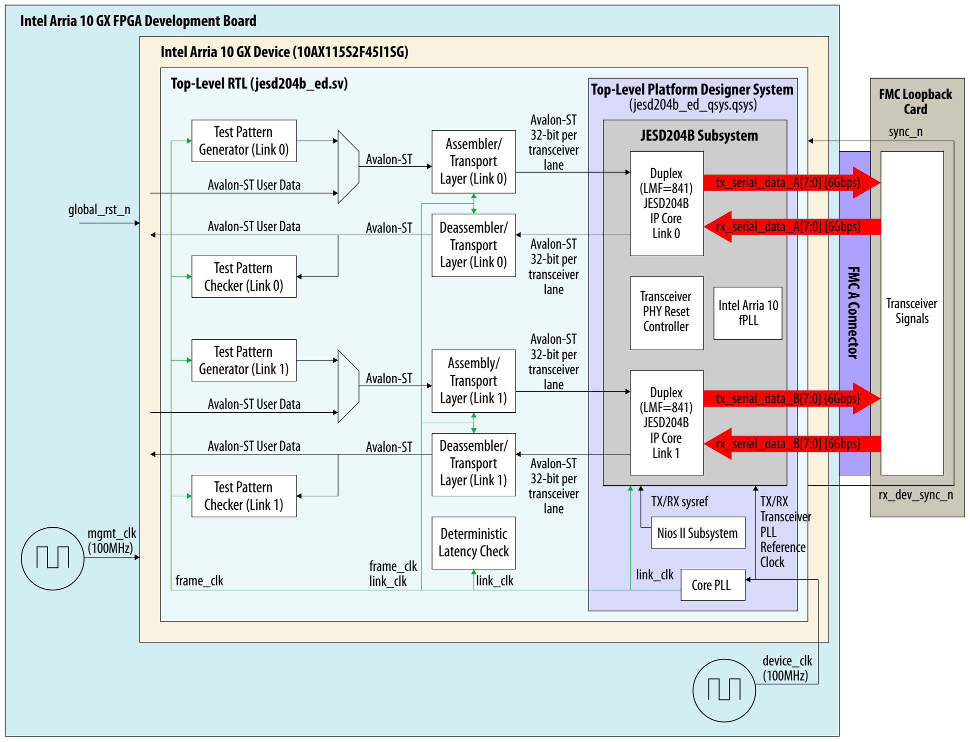
Latency of link measured between transmitter and receiver in this design. The link latency is measured from rising edge of jesd204_tx_link_ready signal to rising edge of jesd204_rx_link_valid signal.


Type
Name
-56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64
...4b_duplex0|jesd204_tx_link_ready
...4b_duplex1|jesd204_tx_link_ready
...4b_duplex0|jesd204_rx_link_valid
...4b_duplex1|jesd204_rx_link_valid
...eck|dl_count:u_dl_count|q[14..0]
...check|link_clk_count_out[14..0]
...eck|dl_count:u_dl_count|q[14..0]
...check|link_clk_count_out[14..0]
0
29

1.3 Reference Design Functional Description

The design runs on the Intel Arria 10 GX FPGA Development Kit.

Figure 7. System level Block Diagram


Intel Arria 10 GX FPGA Development Board
Intel Arria 10 GX Device (10AX115S2F45I1SG)
Top-Level RTL (jesd204b_ed.sv)
Top-Level Platform Designer System (jesd204b_ed_qsys.qsys)
JESD204B Subsystem
FMC Loopback Card
sync_n
Test Pattern Generator (Link 0)
Avalon-ST User Data
Avalon-ST
Assembler/ Transport Layer (Link 0)
Avalon-ST 32-bit per transceiver lane
Duplex (LMF=841) JESD204B IP Core Link 0
tx_serial_data_A[7:0] (6Gbps)
rx_serial_data_A[7:0] (6Gbps)
global_rst_n
Deassembler/ Transport Layer (Link 0)
Test Pattern Checker (Link 0)
Transceiver PHY Reset Controller
Intel Arria 10 fPLL
FMC A Connector
Transceiver Signals
Test Pattern Generator (Link 1)
Assembly/ Transport Layer (Link 1)
Duplex (LMF=841) JESD204B IP Core Link 1
tx_serial_data_B[7:0] (6Gbps)
rx_serial_data_B[7:0] (6Gbps)
Deassembler/ Transport Layer (Link 1)
Test Pattern Checker (Link 1)
rx_dev_sync_n
TX/RX sysref
TX/RX Transceiver PLL Reference Clock
Nios II Subsystem
mgmt_clk (100MHz)
Deterministic Latency Check
frame_clk
frame_clk link_clk
link_clk
Core PLL
device_clk (100MHz)

The reference design consists of the following key modules:

- Top Level HDL
- Platform Designer Top Level System
  - JESD204B Subsystem
    - Reset Sequencer
    - JESD204B IP cores
    - Transceiver PHY Reset Controller
    - fPLL
    - Avalon®-MM Pipeline Bridge
  - Nios Subsystem
    - Nios II Processor
    - On-chip Memory
    - JTAG UART
    - Timer
    - PIO
    - Avalon-MM Bridge
  - Core PLL and core PLL reconfiguration controller
  - SPI Master
- Transport layer
  - Assembler (TX data path)
  - Deassembler (RX data path)
- Test Pattern Generator and Checker
- Deterministic Latency Measurement Module
- Frequency Checker

1.3.1 Top Level HDL

The Platform Designer top system, transport layer, test pattern generator and checker, deterministic latency measurement and frequency checker modules are instantiated in the top-level HDL file called `jesd204b_ed.sv`. The top-level HDL file located at the `<project directory>/` directory includes the project parameters that define the configuration of the reference design.

Table 7. Top level HDL File Parameters

Parameter	Description
LINK	Number of JESD204B link. One link represents one JESD204B instance. A link composed of multiple lanes. Set to match the configuration in JESD204B IP core parameter editor.
SUBSYSTEM	Number of JESD204B subsystem in Platform Designer top level system. A JESD204B subsystem may contain multiple links that requires for multi-devices synchronization.
L	Number of JESD204B lanes per converter device. Set to match the configuration in JESD204B IP core parameter editor.

continued...

Parameter	Description
M	Number of JESD204B converters per device. Set to match the configuration in JESD204B IP core parameter editor.
F	Number of JESD204B octets per frame. Set to match the configuration in JESD204B IP core parameter editor.
N	Number of JESD204B conversion bits per converter device. Set to match the configuration in JESD204B IP core parameter editor.
N_PRIME	Number of JESD204B transmitted bits per sample. Set to match the configuration in JESD204B IP core parameter editor.
S	Number of JESD204B transmitted samples per converter device per frame. Set to match the configuration in JESD204B IP core parameter editor.
CS	Number of JESD204B control bits per conversion sample. Set to match the configuration in JESD204B IP core parameter editor.
F1_FRAMECLK_DIV	Divider ratio for frame_clk when F=1. Refer to the *JESD204B IP Core Design Example User Guide* for more details.
F2_FRAMECLK_DIV	Divider ratio for frame_clk when F=2. Refer to the *JESD204B IP Core Design Example User Guide* for more details.
POLYNOMIAL_LENGTH	Defines the polynomial length for the PRBS pattern generator and checker. Refer to the *JESD204B IP Core Design Example User Guide* for more details.
FEEDBACK_TAP	Defines the feedback tap for the PRBS pattern generator and checker. Refer to the *JESD204B IP Core Design Example User Guide* for more details.

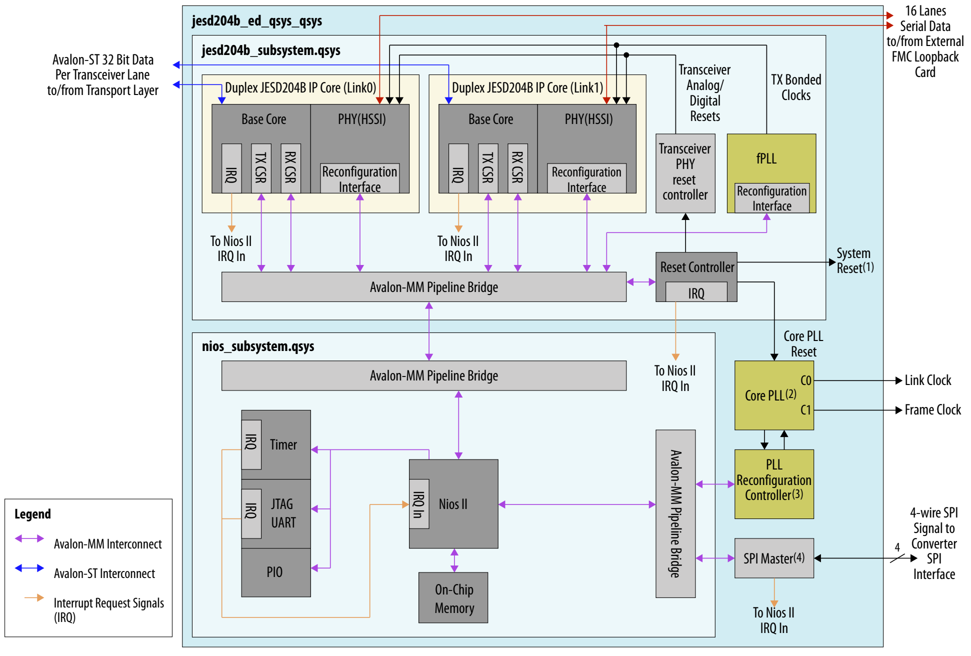
Related Links

JESD204B IP Core Design Example User Guide

1.3.1.1 Platform Designer Top System Component

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

Figure 8. Platform Designer System Top Level Block Diagram


jesd204b_ed_qsys_qsys
jesd204b_subsystem.qsys
Avalon-ST 32 Bit Data Per Transceiver Lane to/from Transport Layer
Duplex JESD204B IP Core (Link0)
Duplex JESD204B IP Core (Link1)
Base Core
PHY(HSSI)
IRQ
TX CSR
RX CSR
Reconfiguration Interface
Transceiver Analog/ Digital Resets
TX Bonded Clocks
16 Lanes Serial Data to/from External FMC Loopback Card
Transceiver PHY reset controller
fPLL
To Nios II IRQ In
Avalon-MM Pipeline Bridge
Reset Controller
System Reset(1)
Core PLL Reset
nios_subsystem.qsys
Core PLL(2)
C0
C1
Link Clock
Frame Clock
Timer
JTAG UART
PIO
IRQ In
Nios II
On-Chip Memory
PLL Reconfiguration Controller(3)
SPI Master(4)
4-wire SPI Signal to Converter SPI Interface
4
Legend
Avalon-MM Interconnect
Avalon-ST Interconnect
Interrupt Request Signals (IRQ)

Notes:

1. System resets consists of: TX/RX JESD204B IP cores CSR resets, TX/RX link resets, TX/RX frame resets.
2. The core PLL receives the device clock from an on-board Arria 10 GX FPGA development board 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 each transport and link layer. The clock from output counter C1 is the frame clock for each transport layer, and pattern generator and checker.
3. PLL Reconfiguration Controller is an optional component added in this design. It is required only if you want to update the output clock frequency, PLL bandwidth, and phase shifts in real time without reconfiguring the entire FPGA.
4. SPI Master is an optional component added in this design. It is required if you want to use Nios II processor to write the configuration data to external converters via 4-wire SPI interconnect.

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

- JESD204B subsystem
- Nios II subsystem
- Core PLL
- PLL reconfiguration controller
- IRQ bridge

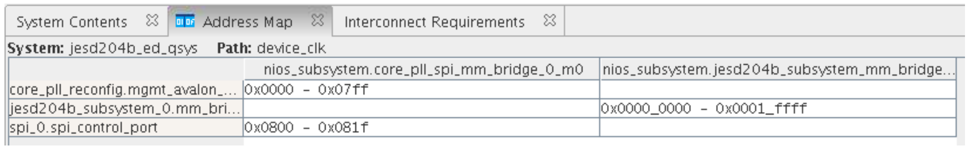
The main data flows through the JESD204B subsystem. The frame data from pattern generator is forwarded to the assembler (TX transport layer). The assembler maps the frame data to non-scrambled octet stream data to the transceiver lane. Each lane has 32-bits of parallel data in Avalon-ST protocol. For each L=8 IP core, the total parallel data bus width is 256 bits. The TX link layer (JESD204B IP core) receives this octet stream data from the assembler, performs scrambling (if enabled) and inserts alignment characters before forwarding the octet stream data to the physical layer. The physical layer (PHY) is part of the JESD204B IP core. The 8b/10 encoded data from the physical layer is transmitted from the high speed serial link transmitter into the FMC loopback card. The FMC loopback card provides a passive pass through of the serial data from the transmitter to the receiver. At the physical layer of the receiver, the 8b/10b decoding is performed and the running disparity and not-in-table errors are monitored. The RX link layer performs lane/frame alignment, octet reconstruction and de-scrambles (if enabled) the data. The RX link layer outputs 8 lanes of 32-bit parallel data/lane in Avalon-ST protocol to the de-assembler (RX transport layer). The de-assembler maps the octet data stream to the frame data. Finally, the pattern checkers verify the data integrity of the frame data.

The Nios II processor in the Nios II subsystem is the Avalon-MM master of the control path. The Avalon-MM slaves such as the JESD204B IP cores and peripherals are connected to the Nios II processor through the Avalon-MM Pipeline Bridges. The core PLL generates the link clock and frame clock for the system. The `mgmt_clk` is directly sourced from external oscillator. To view the top level Platform Designer system in Platform Designer, perform the following steps:

1. Launch the Intel 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 Platform Designer system.

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

Figure 9. Top-Level Address Map View in Platform Designer



System: jesd204b_ed_qsys **Path:** device_clk

	nios_subsystem.core_pll_spi_mm_bridge_0_m0	nios_subsystem.jesd204b_subsystem_mm_bridge...
core_pll_reconfig.mgmt_avalon_...	0x0000 – 0x07ff	
jesd204b_subsystem_0.mm_bri...		0x0000_0000 – 0x0001_ffff
spi_0.spi_control_port	0x0800 – 0x081f	

1.3.1.1.1 JESD204B Subsystem

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

- Two x8-lane JESD204B IP cores configured in duplex, bonded mode (with TX and RX data paths)
- Reset sequencer
- Transceiver PHY reset controller
- fPLL
- Avalon-MM bridge

JESD204B IP Core

The JESD204B IP base core and PHY layer connect to 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 TX CSR parameters
- Base core RX data path—For dynamic reconfiguration of RX CSR parameters
- PHY layer—For dynamic reconfiguration of transceiver PHY CSR (including data rate reconfiguration)

Table 8. JESD204B IP Core Parameter Configuration

Parameter	Value	Description
Subclass	1	Subclass mode
L	8	Number of lanes per converter device
M	4	Number of converters per device
F	1	Number of octets per frame
S	1	Number of transmitted samples per converter per frame
N	16	Number of conversion bits per converter
N'	16	Number of transmitted bits per sample
K	32	Number of frames per multiframe
CS	0	Number of control bits per conversion sample
CF	0	Number of control words per frame clock period per link
HD	1	High Density user data format
SCR	On	Enable scrambler

Reset Sequencer

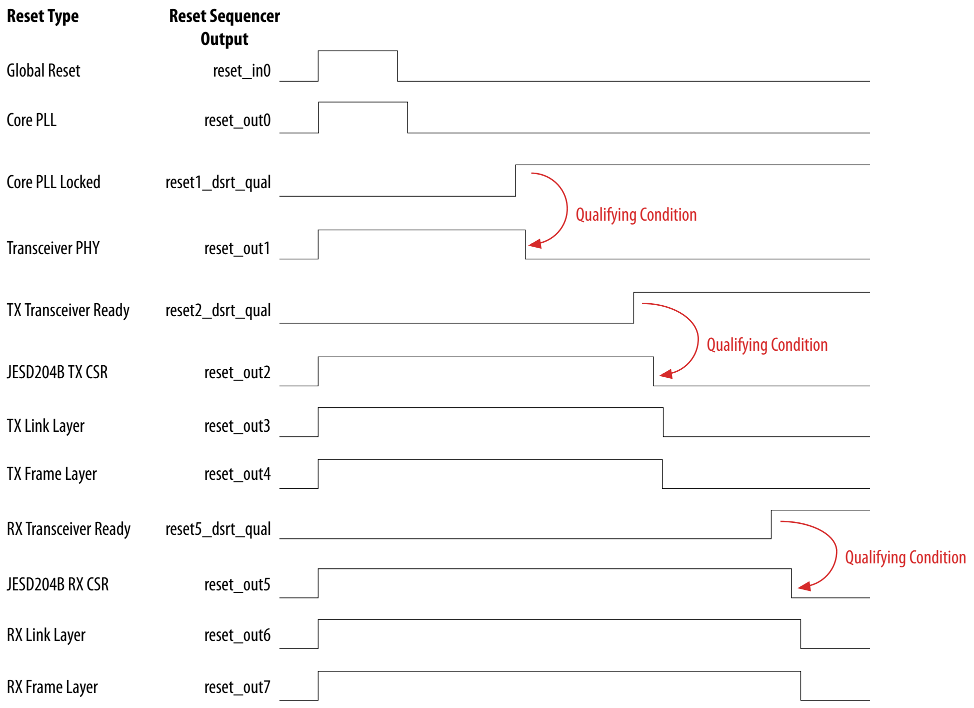
The reset sequencer generates the system resets for the following modules in the system:

- Core PLL reset—resets the core PLL.
- TX/RX Transceiver PHY reset—resets the TX/RX JESD204B IP cores PHY module.
- TX/RX JESD204B IP core CSR reset—resets the TX/RX JESD204B IP cores configuration status registers.
- TX/RX link reset—resets the TX/RX JESD204B IP cores base module, transport layer.
- TX/RX frame reset—resets the TX/RX transport layer, pattern generator and checker modules.

You can issue hardware reset or software reset to the reset sequencer. The reset input pin of the reset sequencer is connected to the PB0 push button on the Intel Arria 10 FPGA development kit. Press PB0 push button to generate the hardware reset. The Nios II software generates the software reset to reset sequencer. The reset sequencer has a pre-defined reset sequence as shown in the following figure. The figure also shows the components that are affected by the reset outputs of the reset sequencer.

The hardware reset follows this predefined sequence. The software reset has the flexibility of following this predefined reset sequence or overwriting the reset sequence.

Figure 10. Reset Sequence



Transceiver PHY Reset Controller

The transceiver PHY reset controller receives the transceiver PHY reset output from the reset sequencer. It generates the proper analog and digital reset sequence for the two x8-lane JESD204B IP cores PHY module.

fPLL

The fPLL is the TX PLL. It generates the fast clock for the serial data in PMA and parallel clock for the parallel data in PMA and PCS. The reference clock input to the fPLL is the `device_clk`.

Avalon-MM Bridge

All the Avalon-MM slaves 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 system.

JESD204B Subsystem Address Map

You can access the address map of slaves in the JESD204B subsystem by clicking on the **Address Map** tab in the Platform Designer window. The design contains two JESD204B IP cores instances only per subsystem, so instance 2 and 3 in the following table are excluded.

Table 9. JESD204B Subsystem Address Map

Note:

- Due to the address space limit (0xc000-0xcfff and 0xd000-0xdfff) defined for JESD204B IP core TX/RX CSR, you can only add up to 4 JESD204B IP cores instances for address ranges defined in this design. You can define these base addresses if your design contains four JESD204B IP cores instances per subsystem.
- You may need to define the new base address for the Avalon-MM peripherals to avoid conflict addressing occur if you have more than four JESD204B IP cores instances in your design.

Avalon-MM Peripheral	Address Map
JESD204B IP cores transceiver reconfiguration interfaces	• `0x0000_0000 - 0x0000_7fff (Instance 0)` • `0x0001_0000 - 0x0001_7fff (Instance 1)` • `0x0002_0000 - 0x0002_7fff (Instance 2)` • `0x0003_0000 - 0x0003_7fff (Instance 3)`
fPLL	`0x0000_8000 - 0x0000_8fff`
JESD204B IP cores CSR - TX	• `0x0000_c000 - 0x0000_c3ff (Instance 0)` • `0x0000_c400 - 0x0000_c7ff (Instance 1)` • `0x0000_c800 - 0x0000_cbff (Instance 2)` • `0x0000_cc00 - 0x0000_cfff (Instance 3)`
JESD204B IP cores CSR - RX	• `0x0000_d000 - 0x0000_d3ff (Instance 0)` • `0x0000_d400 - 0x0000_d7ff (Instance 1)` • `0x0000_d800 - 0x0000_dbff (Instance 2)` • `0x0000_dc00 - 0x0000_dfff (Instance 3)`
Reset Sequencer	`0x0000_e000 - 0x0000_e0ff`

1.3.1.1.2 Nios II Subsystem in Platform Designer

The Nios II subsystem Platform Designer project, *nios_subsystem.qsys*, instantiates the following peripherals:

- Nios II processor
- On-chip memory (altera_avalon_onchip_memory2)—provides both instruction and data memory space
- Timer—provides a general timer function for the software
- JTAG UART—serves as the main communications portal between the user and the Nios II processor via the terminal console in Nios II SBT for Eclipse tool
- Avalon-MM bridges—two Avalon-MM bridge modules; one to interface to the JESD204B subsystem and the other interface to the Platform Designer components (core PLL reconfiguration controller and SPI master module)
- 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

Table 10. Signal Connectivity for `io_status` Registers

Bit	Signal
0	Core PLL locked
1	TX transceiver ready (Link 0/Instance 0)
2	RX transceiver ready (Link 0/Instance 0)
3	Test pattern checker data error (Link 0/Instance 0)
4–31	TX transceiver ready, RX transceiver ready, and test pattern checker data error signals for subsequent JESD204B IP cores (Link 1-9/ Instance 1-9), if present.

Table 11. Signal Connectivity for `io_control` Registers

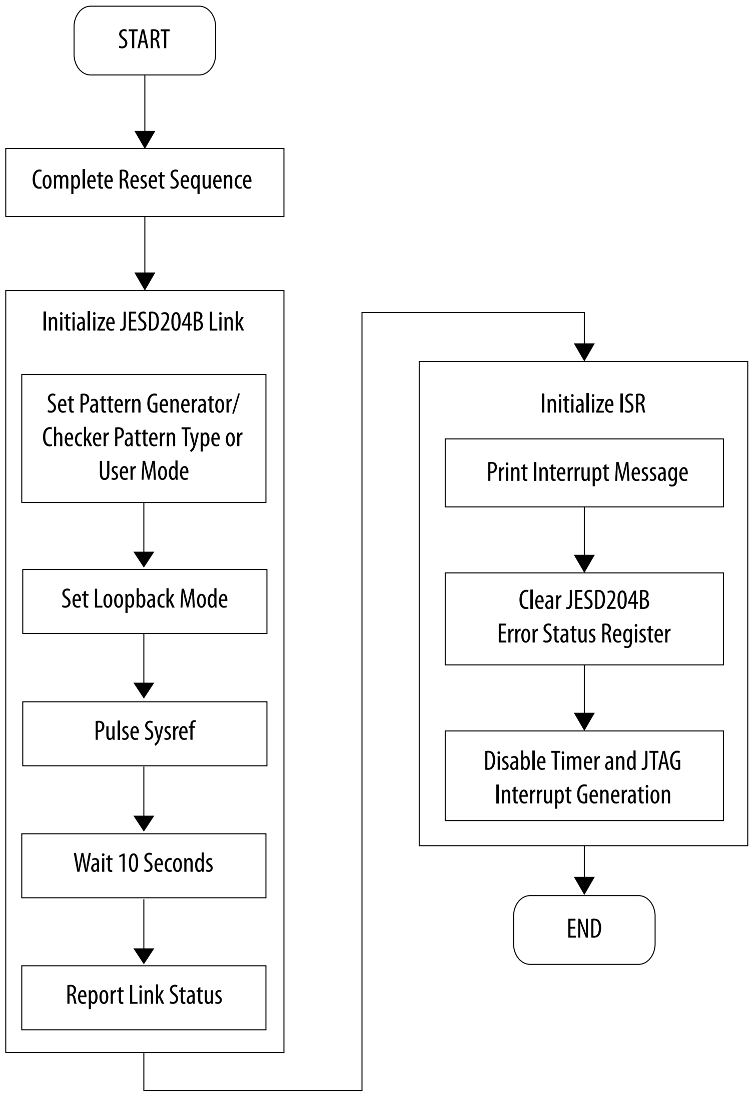
Bit	Signal
0	RX serial loopback enable for two JESD204B IP cores in this design
1	Reserved
2	Reserved
3	Reserved
4–30	Reserved
31	Sysref

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

Nios II Processor

The reference design uses Nios II processor as a control unit to control certain aspects of JESD204B system using C-based software control flow. The software C codes included as part of this reference design only perform basic JESD204B link initialization. You can modify the C codes based on your system specifications.

Figure 11. Software Main C Code Execution Flow

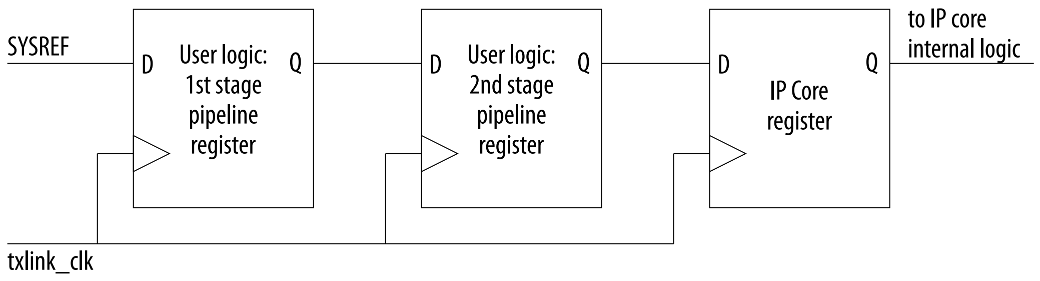


The JESD204B link initialization performs the following tasks:

- 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 external 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.

The SYSREF signal acts as a common timing reference for multiple JESD204B Subclass 1 devices in a JESD204B system to achieve deterministic latency. In general, SYSREF is a source synchronous to the device clock from single clock chip. In most cases, the register in the IP core, which detects this SYSREF signal, is far away from the SYSREF pin. The long interconnect routing delay may result in timing violation. Because of there is no clock generator on the development kit that can provide SYSREF to JESD204B IP cores, the PIO is used to generate the SYSREF pulse. The Nios II processor instructs the PIO to generate SYSREF pulse, which is re-timed by two stages pipeline registers before being sampled by each IP core. Refer to the *JESD204B IP Core User Guide* for constraining SYSREF signal if it is coming from FPGA input pin.

Figure 12. Two Stage Pipeline Registers for SYSREF Signal


SYSREF
D
User logic: 1st stage pipeline register
Q
D
User logic: 2nd stage pipeline register
Q
D
IP Core register
Q
to IP core internal logic
txlink_clk

The initialized ISR performs the following tasks:

- Print and clear the RX/TX JESD204B error status registers.
- Disable timer and JTAG UART interrupt generation.

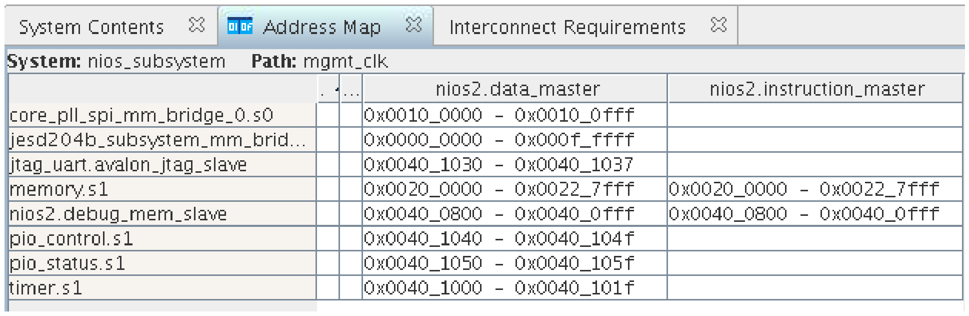
Related Links

JESD204B IP Core User Guide

Nios II Subsystem Address Map

You can access the address mapping of the peripherals in Nios II subsystem by clicking on the Address Map tab in the Platform Designer window.

Figure 13. Nios II Subsystem Address Map View in Platform Designer



System: nios_subsystem **Path:** mgmt_clk

	. ·	·...	nios2.data_master	nios2.instruction_master
core_pll_spi_mm_bridge_0.s0			0x0010_0000 - 0x0010_0fff	
jesd204b_subsystem_mm_brid...			0x0000_0000 - 0x000f_ffff	
jtag_uart.avalon_jtag_slave			0x0040_1030 - 0x0040_1037	
memory.s1			0x0020_0000 - 0x0022_7fff	0x0020_0000 - 0x0022_7fff
nios2.debug_mem_slave			0x0040_0800 - 0x0040_0fff	0x0040_0800 - 0x0040_0fff
pio_control.s1			0x0040_1040 - 0x0040_104f	
pio_status.s1			0x0040_1050 - 0x0040_105f	
timer.s1			0x0040_1000 - 0x0040_101f	

1.3.1.1.3 Core PLL

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

For the frame clock, when F=1 and F=2, the resulting frame clock frequency can easily exceed the capability of the core. Thus, 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/checker to operate at a divided factor of required frame clock rate by widening the transport layer Avalon-ST interface data width accordingly. For this reference design, the *F1_FRAMECLK_DIV* is set to 4 and *F2_FRAMECLK_DIV* is set to 2. For example, the actual frame clock and link clock required for a serial data rate of 6.0 Gbps and F=1 in this design is calculated as below:

Table 12. Clocks

Clock	Formula	Description
Link Clock (from output C0)	Serial data rate/40	The link clock clocks the link layer JESD204B IP cores, link interface of transport layer, multi-stage pipeline for SYSREF, deterministic latency measurement module.
Frame Clock (from output C1)	Serial data rate/(10 × F)	The frame clock clocks the transport layer, test pattern generators and checkers.

Frame clock = 6000/(10x1)/*F1_FRAMECLK_DIV*= 600/4= 150Mhz

Link clock = 6000/40= 150Mhz

1.3.1.1.4 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. This is an optional component because the design only showcases a static data rate operation.

1.3.1.1.5 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 connected to the Nios II processor via the Avalon-MM interconnect. This module is an optional component because the design only showcases external FMC loopback.

1.3.1.2 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 Platform Designer 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 Platform Designer system to implement such feature in the software.

1.3.1.3 Test Pattern Generator

The test pattern generator has three patterns options; PRBS, alternate checkerboard, or RAMP wave. The test pattern is sent to the transport layer when test mode is selected. By default, the PRBS pattern test mode is selected.

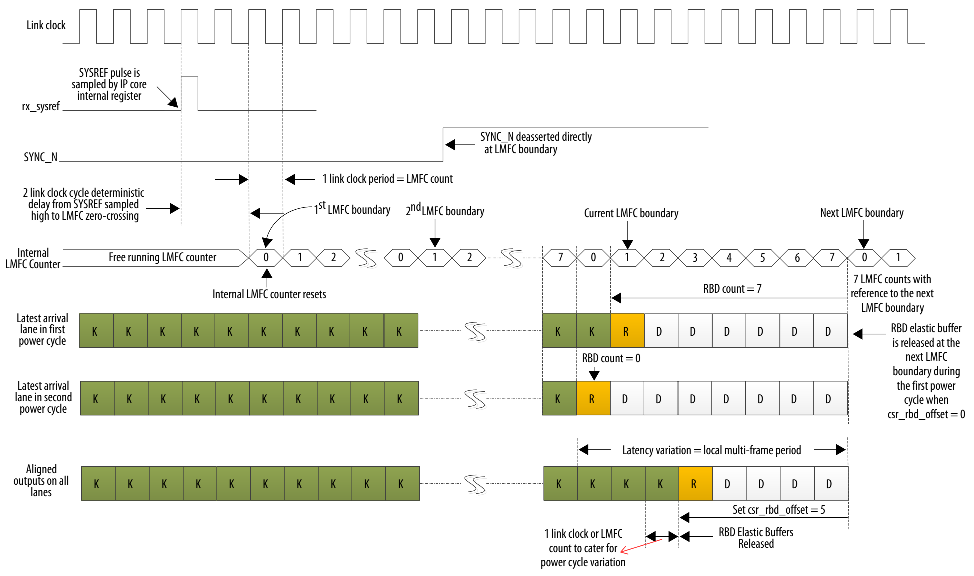
1.3.1.4 Test Pattern Checker

The test pattern checker has three patterns options: PRBS, alternate checkerboard, or RAMP wave. The test pattern checker verifies the test pattern outputs from the transport layer when test mode is selected.

1.3.1.5 Deterministic Latency Measurement Module

The link latency of the design is measured from the TX core is ready to accept data (the rise of) until the data to the RX transport layer is valid. The RBD count varies from 0 to ((FxK/4)-1), which is 0 to 7. You may observe the RBD count=0 or 7 in this FMC external loopback setup. In this scenario, deterministic latency is not guaranteed because the RBD elastic buffer can be released either at the current LMFC boundary when RBD count=0 or one local multi-frame period later at the next LMFC boundary when RBD count=7. During multiple power cycles of the board, you may observe the link latency of 26 or 34 link clock cycles when RBD count is 0 or 7. Due to this, the lane de-skew error could happen during link initialization. To fix the deterministic latency issue, the csr_rbd_offset is programmed to 5 to force the release of RBD elastic buffer 5 LMFC counts before the next LMFC boundary. By setting RBD offset value equal to 5, the counter consistently shows link latency of 29 link clock cycles from one power cycle to another power cycle.

Figure 14. Early RBD Release Opportunity for Latest Arrival Lane Across Two Multi-frames


Link clock
SYSREF pulse is sampled by IP core internal register
rx_sysref
SYNC_N deasserted directly at LMFC boundary
SYNC_N
1 link clock period = LMFC count
2 link clock cycle deterministic delay from SYSREF sampled high to LMFC zero-crossing
1st LMFC boundary
2nd LMFC boundary
Current LMFC boundary
Next LMFC boundary
Internal LMFC Counter
Free running LMFC counter
0 1 2 0 1 2 7 0 1 2 3 4 5 6 7 0 1
Internal LMFC counter resets
RBD count = 7
7 LMFC counts with reference to the next LMFC boundary
Latest arrival lane in first power cycle
K K K K K K K K K K K K R D D D D D D
RBD elastic buffer is released at the next LMFC boundary during the first power cycle when csr_rbd_offset = 0
RBD count = 0
Latest arrival lane in second power cycle
K K K K K K K K K K K R D D D D D D D
Latency variation = local multi-frame period
Aligned outputs on all lanes
K K K K K K K K K K K K K K R D D D D
Set csr_rbd_offset = 5
1 link clock or LMFC count to cater for power cycle variation
RBD Elastic Buffers Released

1.3.1.6 Frequency Checker

The frequency checker module is added in this design to measure the device clock, PHY clock generated from the transceiver parallel clock for the TX path and the recovered clock generated from the CDR for the RX path. This module is useful for debugging and it ensures that the frequency of measured clocks is as close as the desired clock.

1.3.2 Clocking Scheme

The reference design requires two clock sources coming from the Intel Arria 10 GX FPGA development kit for proper operation. The reference design uses the default 100Mhz clock frequency from on-board oscillator.

Table 13. Reference Design System Clock Summary

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

Signal Name	Description	Usage
`device_clk`	External 100 Mhz clock from X3 Si570 Programmable Oscillator.	Input reference clock for core PLL (IO PLL), TX transceiver fPLL, RX transceiver CDR.
`link_clk`	Link and transport layer clock from core PLL (IO PLL).	Clock source for each JESD204B IP Core link layer and transport layer link interface, deterministic latency measurement module.
`frame_clk`	Transport and application layer clock from core PLL (IO PLL).	Clock source for each transport layer, test pattern generator and checker.
`mgmt_clk`	External 100 Mhz clock from X3 Si570 Programmable Oscillator.	Management clock for nios II subsystem, transceiver reconfiguration interfaces, JESD204B IP cores Avalon-MM interfaces, altera PLL reconfig, SPI master, frequency checker.

1.3.3 FPGA Pin Assignments

The top-level signals with its corresponding FPGA pin assignments on the Intel Arria 10 GX FPGA development board are listed in the table below.

Table 14. Top Level Signals with Corresponding FPGA Pin, IO Standard and Direction

In this design, the PIO internally generates the SYSREF signal in the Nios II Subsystem.

Top Level Signal Name	FPGA Pin Number	I/O Standard	Direction
`global_rst_n`	T12	1.8V	Input
`device_clk`	AC8	LVDS	Input
`device_clk (n)`	AC7	LVDS	Input
`mgmt_clk`	AR36	LVDS	Input
`mgmt_clk (n)`	AR37	LVDS	Input
`tx_serial_data[0]`	BC7	High Speed Differential I/O	Output
`tx_serial_data[0] (n)`	BC8	High Speed Differential I/O	Output
`tx_serial_data[1]`	BD5	High Speed Differential I/O	Output
`tx_serial_data[1] (n)`	BD6	High Speed Differential I/O	Output
`tx_serial_data[2]`	BB5	High Speed Differential I/O	Output
`tx_serial_data[2] (n)`	BB6	High Speed Differential I/O	Output
`tx_serial_data[3]`	BC3	High Speed Differential I/O	Output
`tx_serial_data[3] (n)`	BC4	High Speed Differential I/O	Output
`tx_serial_data[4]`	BB1	High Speed Differential I/O	Output
`tx_serial_data[4] (n)`	BB2	High Speed Differential I/O	Output
`tx_serial_data[5]`	BA3	High Speed Differential I/O	Output
`tx_serial_data[5] (n)`	BA4	High Speed Differential I/O	Output
`tx_serial_data[6]`	AY1	High Speed Differential I/O	Output
`tx_serial_data[6] (n)`	AY2	High Speed Differential I/O	Output
`tx_serial_data[7]`	AW3	High Speed Differential I/O	Output
`tx_serial_data[7] (n)`	AW4	High Speed Differential I/O	Output
`tx_serial_data[8]`	AV1	High Speed Differential I/O	Output
`tx_serial_data[8] (n)`	AV2	High Speed Differential I/O	Output
`tx_serial_data[9]`	AU3	High Speed Differential I/O	Output
`tx_serial_data[9] (n)`	AU4	High Speed Differential I/O	Output
`tx_serial_data[10]`	AT1	High Speed Differential I/O	Output
`tx_serial_data[10] (n)`	AT2	High Speed Differential I/O	Output
`tx_serial_data[11]`	AR3	High Speed Differential I/O	Output
`tx_serial_data[11] (n)`	AR4	High Speed Differential I/O	Output

continued...

Top Level Signal Name	FPGA Pin Number	I/O Standard	Direction
tx_serial_data[12]	AP1	High Speed Differential I/O	Output
tx_serial_data[12] (n)	AP2	High Speed Differential I/O	Output
tx_serial_data[13]	AM1	High Speed Differential I/O	Output
tx_serial_data[13] (n)	AM2	High Speed Differential I/O	Output
tx_serial_data[14]	AK1	High Speed Differential I/O	Output
tx_serial_data[14] (n)	AK2	High Speed Differential I/O	Output
tx_serial_data[15]	AH1	High Speed Differential I/O	Output
tx_serial_data[15] (n)	AH2	High Speed Differential I/O	Output
rx_serial_data[0]	AW7	High Speed Differential I/O	Input
rx_serial_data[0] (n)	AW8	High Speed Differential I/O	Input
rx_serial_data[1]	BA7	High Speed Differential I/O	Input
rx_serial_data[1] (n)	BA8	High Speed Differential I/O	Input
rx_serial_data[2]	AY5	High Speed Differential I/O	Input
rx_serial_data[2] (n)	AY6	High Speed Differential I/O	Input
rx_serial_data[3]	AV5	High Speed Differential I/O	Input
rx_serial_data[3] (n)	AV6	High Speed Differential I/O	Input
rx_serial_data[4]	AT5	High Speed Differential I/O	Input
rx_serial_data[4] (n)	AT6	High Speed Differential I/O	Input
rx_serial_data[5]	AP5	High Speed Differential I/O	Input
rx_serial_data[5] (n)	AP6	High Speed Differential I/O	Input
rx_serial_data[6]	AN3	High Speed Differential I/O	Input
rx_serial_data[6] (n)	AN4	High Speed Differential I/O	Input
rx_serial_data[7]	AM5	High Speed Differential I/O	Input
rx_serial_data[7] (n)	AM6	High Speed Differential I/O	Input
rx_serial_data[8]	AL3	High Speed Differential I/O	Input
rx_serial_data[8] (n)	AL4	High Speed Differential I/O	Input
rx_serial_data[9]	AK5	High Speed Differential I/O	Input
rx_serial_data[9] (n)	AK6	High Speed Differential I/O	Input
rx_serial_data[10]	AJ3	High Speed Differential I/O	Input
rx_serial_data[10] (n)	AJ4	High Speed Differential I/O	Input
rx_serial_data[11]	AH5	High Speed Differential I/O	Input
rx_serial_data[11] (n)	AH6	High Speed Differential I/O	Input
rx_serial_data[12]	AG3	High Speed Differential I/O	Input
rx_serial_data[12] (n)	AG4	High Speed Differential I/O	Input

continued...

Top Level Signal Name	FPGA Pin Number	I/O Standard	Direction
`rx_serial_data[13]`	AF5	High Speed Differential I/O	Input
`rx_serial_data[13] (n)`	AF6	High Speed Differential I/O	Input
`rx_serial_data[14]`	AE3	High Speed Differential I/O	Input
`rx_serial_data[14] (n)`	AE4	High Speed Differential I/O	Input
`rx_serial_data[15]`	AD5	High Speed Differential I/O	Input
`rx_serial_data[15] (n)`	AD6	High Speed Differential I/O	Input
`sync_n_in`	AR15	LVDS	Input
`sync_n_in (n)`	AT15	LVDS	Input
`sync_n_out`	AT17	LVDS	Output
`sync_n_out (n)`	AU17	LVDS	Output
`user_led_g[0]`	D18	1.8V	Output
`user_led_g[1]`	C18	1.8V	Output
`user_led_g[2]`	A19	1.8V	Output
`user_led_g[3]`	J24	1.8V	Output
`user_led_g[4]`	L25	1.8V	Output
`user_led_g[5]`	K25	1.8V	Output
`user_led_g[6]`	K26	1.8V	Output
`user_led_g[7]`	L28	1.8V	Output

1.4 Document Revision History for AN 814: Intel Arria 10 Two x8-Lane JESD204B IP Cores (Duplex) Multi-Device Synchronization Reference Design

Date	Version	Changes
January 2018	2018.01.30	Renamed the document as *AN 814: Intel Arria 10 Two x8-Lane JESD204B IP Cores (Duplex) Multi-Device Synchronization Reference Design*.
December 2017	2017.12.18	• Renamed the document as *AN 814: Intel Arria 10 Two x8-Lanes JESD204B IP Cores (Duplex) Multi-Device Synchronization Reference Design using Nios II Processor*. • Added a note to clarify that the IOPLL input reference clock is sourcing from device clock through global clock network in the *Clocking Scheme* topic. • Updated for latest branding standards. • Made editorial updates throughout the document.
June 2017	2017.06.15	Initial release.