LogiCORE IP JESD204 v2.1

User Guide

UG774 April 24, 2012

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/11	1.0	Initial Xilinx release
04/24/12	2.0	Updated for core version 2.1. ISE Release 14.1. Add 8 lane, 10.3125 Gb/s and support for transceiver sharing.

Table of Contents

XILINX®

Chapter 1

Introduction

The LogiCORE™ IP JESD204 core implements a JESD204A or JESD204B interface supporting line rates of 1, 2.5, 3.125 and 6.25 Gb/s on 1, 2 or 4 lanes using GTX transceivers in Virtex®-6 and Kintex™-7 FPGAs. The JESD204 core can be configured as Transmit or Receive.

The JESD204 core is a fully-verified solution design delivered by using the Xilinx® CORE Generator™ tool as an NGC netlist. In addition, an example design is provided in Verilog.

System Requirements

For a list of operating system requirements, see the ISE Design Suite 14: Release Notes Guide.

About the Core

The JESD204 core is a Xilinx CORE Generator core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the JESD204 product page.

For information about licensing options, see Chapter 2, Licensing the Core.

Recommended Design Experience

Although the JESD204 core is a fully-verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high performance, pipelined FPGA designs using Xilinx implementation software and the UCF is recommended.

Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

Verification and Interoperability

The JESD204 core has been verified using both simulation and hardware testing.

Simulation

A highly parameterizable transaction-based simulation test suite has been used to verify the core. Tests include:

- scrambling and alignment
- loss and regain of synchronization
- frame transmission
- frame reception
- recovery from error conditions

Hardware Testing

The core has been used in many hardware test platforms within Xilinx and in interoperability testing with external hardware vendors.

Technical Support

To obtain technical support for the JESD204 core, visit www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the JESD204 core.

In addition, support resources such as Application Notes, User Guides, Reference and Software Manuals, and Answer Records can also be accessed from the support site. Xilinx provides technical support for use of this product as described in this guide. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the JESD204 core and the documentation supplied with the core.

JESD204 Core

For comments or suggestions about the JESD204 core, submit a WebCase from www.xilinx.com/support. Be sure to include this information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about this document, submit a WebCase from www.xilinx.com/support. Be sure to include this information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

XILINX®

Chapter 2

Licensing the Core

The JESD204 core is provided under the terms of the Xilinx End User License Agreement and can be generated using the Xilinx® CORE Generator™ system. The CORE Generator system is shipped with the Xilinx ISE® Design Suite.

License Options

The JESD204 core provides three licensing options. After installing Xilinx ISE and the required IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the JESD204 core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the JESD204 core using the example design and demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Gate-level functional simulation support
- Back annotated gate-level simulation support
- Functional simulation support
- Full-implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time-outs

Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator tool.

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, perform these steps:

1. Navigate to the JESD204 product page for this core.
2. Click Evaluate.
3. Follow the instructions to install Xilinx ISE and the required IP Service Packs.

Obtaining a Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the 'Access Core' link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator tool and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Chapter 3

Core Architecture

System Overview

The JESD204 core supports JESD204, JESD204A and JESD204B. The original JESD204 specification defined a serial link between one data converter and a logic device. The link was made up of one lane. Revision A and B extend this to cover multiple converters, each linked to the logic device using multiple lanes. See Figure 3-1.

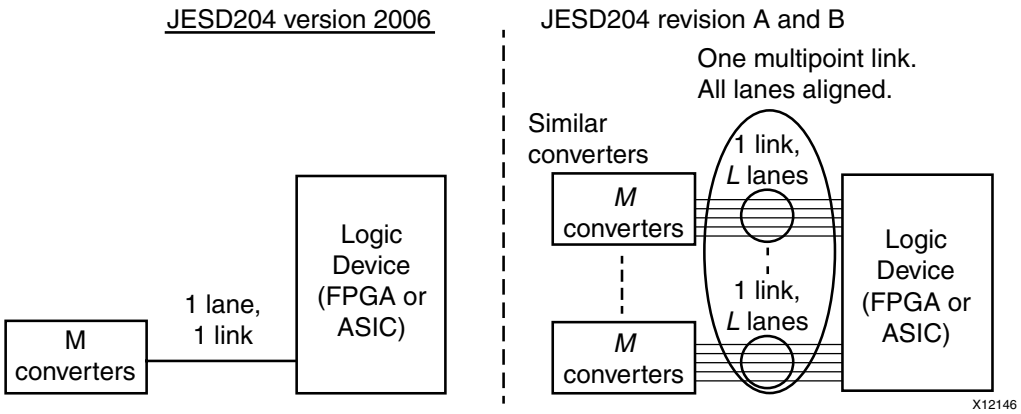

Figure 3-1: **System Overview**

Functional Description

Block Level Architecture

The JESD204 core is delivered as a netlist and supporting wrapper files by the Xilinx® CORE Generator™ tool. Depending on the options selected in core generation, the core netlist consists of the following: JESD204 transmitter or JESD204 receiver

Block-level Verilog wrappers are provided to instantiate the JESD204 core, the clock/reset logic, Management Block, and the GTX transceiver. The block-level is intended to be instantiated in user designs rather than using the netlist directly.

The Management Block providing core Control and Status registers with a standard AXI4-Lite interface is provided as source code. This is shown in Figure 3-2. You can choose to remove the Management block and configure the core using the configuration signals directly, either statically or using some other CPU interface.

A Verilog example design is provided that instantiates the block-level wrapper together with example interface modules. This is a device-level design and can be used to run the core through the Xilinx tool flow but is not intended to be used directly in customer designs. Figure 3-2 shows the core hierarchy.

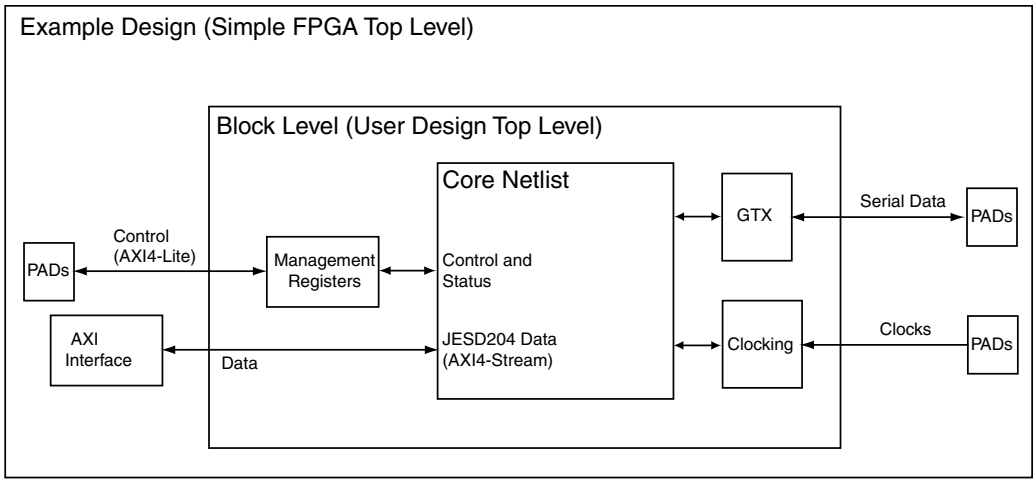

Figure 3-2: **Hierarchy**

Core Architecture

The Transmit and Receive logic is completely separate; a core can be generated to be a transmitter or a receiver. Optionally, the core can be generated with a wrapper which allows sharing of a single transceiver between transmit and receive, see Sharing Transceivers between Transmit and Receive.

Transmitter

The transmitter is shown in Figure 3-3.

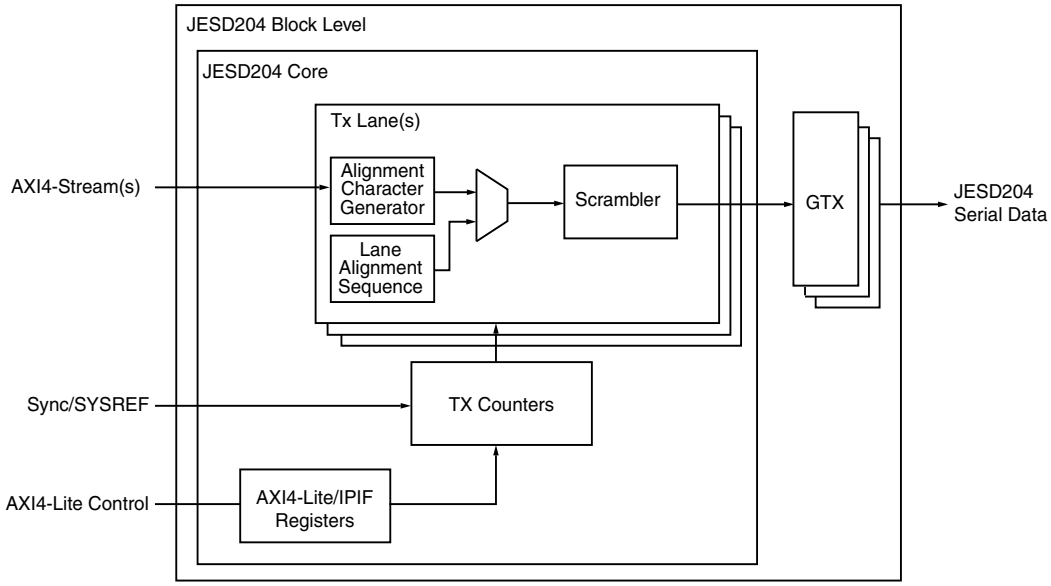

Figure 3-3: **Transmitter**

- tx_lane - The lanes are completely independent, taking in 32 bits of data and generating 32-bit data to a single GTX transceiver. The lane logic implements the ILA sequence generation, the optional scrambling logic, and the insertion of alignment characters based on the rules in JESD204A and JESD204B.
- tx_counters - The generation of the timing signals for the ILA generation of each lane is common and carried out by this logic. This block also controls when the lanes should transmit data based on the received sync from the JESD204A receiver.
- tx_top - Top level of the transmitter containing 1, 2, 4 or 8 lanes and the common logic.

Receiver

The receiver is shown in Figure 3-4.

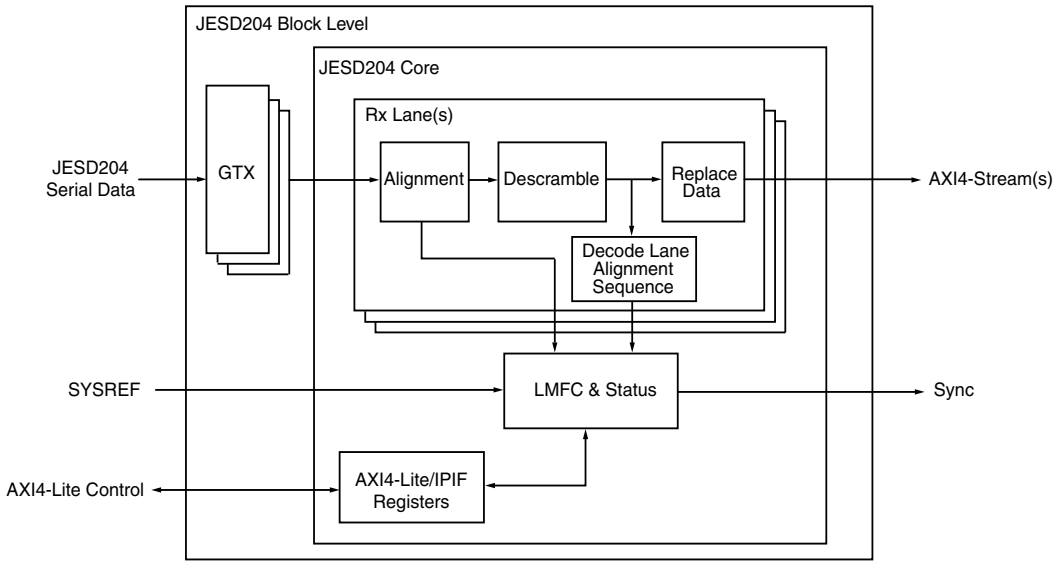

Figure 3-4: **Receiver**

- Rx lane - The lanes are completely independent taking in 32 bits of data and generating 32-bit data to a single GTX transceiver. The lane logic implements the ILA sequence generation, the optional scrambling logic and the insertion of alignment characters based on the rules in JESD204A and JESD204B.
- Misalignment Detection- The generation of the timing signals for the ILA generation of each lane is common and carried out by this logic. This block also controls when the lanes should transmit data based on the received sync from the JESD204A receiver.
- Rx top - Top level of the transmitter containing 1, 2, 4 or 8 lanes and the common logic.

Block Interfaces

The top-level signals of the block level wrapper are described in Table 3-1. The signals of the core netlist are described for reference in Chapter 6, Interfacing to the Core. See the *Xilinx AXI Reference Guide* [Ref 9] for AXI4 interface information.

Table 3-1: **Block Level Interface Signals**

Name	Direction	Description
Clocks and Resets		
refclk0p/n	In	Reference clock to transceiver (differential)
refclk1p/n	In	Second reference clock to transceiver used only when sharing a GTX transceiver between transmit and receive (differential).
reset	In	Asynchronous reset
refclkout	Out	Buffered version of reference clock
Transceiver Interface		
txp[LANES-1:0]	Out	Serial data (Transmit only)
txn[LANES-1:0]	Out	Serial data (Transmit only)
rxp[LANES-1:0]	In	Serial data (Receive only)
rxn[LANES-1:0]	In	Serial data (Receive only)
Management Interface (AXI4-Lite)		
s_axi_aclk	In	Clock
s_axi_aresetn	In	Active-Low reset
s_axi_awaddr[31:0]	In	Write Address
s_axi_awvalid	In	Write Address Valid
s_axi_awready	Out	Write Address Ready
s_axi_wdata[31:0]	In	Write Data
s_axi_wstrb[3:0]	In	Write Data Byte Strobe
s_axi_wvalid	In	Write Data Valid
s_axi_wready	Out	Write Data Ready
s_axi_bresp[1:0]	Out	Write Response (Always = 00 = OK)
s_axi_bvalid	Out	Write Response Valid
s_axi_bready	In	Write Response Ready
s_axi_araddr[31:0]	In	Read Address
s_axi_arvalid	In	Read Address Valid
s_axi_arready	Out	Read Address Ready
s_axi_rdata[31:0]	Out	Read Data
s_axi_rresp[1:0]	Out	Read Response (Always = 00 = OK)

Table 3-1: **Block Level Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_rvalid	Out	Read Data Valid
s_axi_rready	In	Read Data Ready
AXI4-Stream Interface Signals (Transmit Only) One per lane, N=0..(LANES-1)		
tx_aclk	Out	Clock (shared by all lanes)
tx_aresetn	Out	Active-Low reset (shared by all lanes)
txN_tdata[31:0]	In	AXI transmit data (samples and control words)
txN_tvalid	In	AXI transmit data valid
txN_tready	Out	AXI slave ready for data
AXI4-Stream Interface Signals (Receive Only) One per lane, N=0..(LANES-1)		
rx_aclk	Out	Clock (shared by all lanes)
rx_aresetn	Out	Active-Low reset (shared by all lanes)
rxN_tdata[31:0]	Out	AXI receive data (samples and control words)
rxN_tvalid	Out	AXI receive data valid
rxN_tready	In	AXI master ready for data
Non-AXI Data Interface Signals (Transmit Only)		
tx_start_of_frame[3:0]	Out	Frame boundary indication
tx_start_of_multiframe[3:0]	Out	Frame boundary indication
tx_sysref_in	In	Sysref Input (JESD204B only)
tx_sysref_out	Out	Sysref output (JESD204B only)
tx_sync	In	Sync signal common for all lanes
Non-AXI Data Interface Signals (Receive Only)		
rx_start_of_frame[3:0]	Out	Frame boundary indication
rx_end_of_frame[3:0]	Out	Frame boundary indication
rx_frame_error[(LANES*4)-1:0]	Out	Error one bit per data output byte
rx_sync	Out	Sync signal common for all lanes
rx_sysref_in	In	Sysref Input (JESD204B only)
rx_sysref_out	Out	Sysref Output (JESD204B only)

Chapter 4

Customizing and Generating the Core

The JESD204 core is generated using the Xilinx® CORE Generator™ tool. The core is generated as an NGC netlist and is customized at generation time. This chapter describes how to generate the core and how to customize the JESD204 core to your site-specific requirements.

Figure 4-1: **Main GUI**

GUI Interface

For general help with starting and using the CORE Generator tool on your development system, see the documentation supplied with the Xilinx ISE® Design Suite.

Component Name

The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed from these characters: a through z, 0 through 9 and "_" (underscore).

Lanes Per Link

The core supports 1, 2, 4 or 8 lanes.

Transmit/Receive

The core can be configured as a transmitter, for connection to DAC devices, or receiver, for connection to ADC devices. Additionally, the core can be configured to allow sharing a transceiver between a transmit and receive core.

JESD204A or JESD204B

The core can be generated to support A only or B which is backwards compatible with A but includes extra features and faster line rates.

JESD204B Subclass

If JESD204B is enabled then subclass 0, 1 or 2 can be selected.

Advanced Options

When JESD204B is selected, the size of the LMFC buffer can be adjusted. In addition the LMFC buffer can be implemented using block RAM or distributed RAM.

Checksum All Bytes

When selected, the transmitter performs a checksum on the configuration bytes in the ILA by summing all 13 bytes; otherwise, only the individual fields are summed. The default is to sum over only the individual fields as per JESD204A and JESD204B specifications but some converter devices might require the checksum to be generated over the 13 configuration bytes.

Parameter Values in the XCO file

XCO files contain parameterization information for an instance of a core; an XCO file is created when a core is generated and can be used to recreate a core. As most of the core parameters are available in the Verilog source code as generics, the XCO contains very little information about the core. The text in an XCO file is not case-sensitive. Table 4-1 shows the XCO file parameters and values and summarizes the GUI defaults.

This is an example extract from an XCO file:

```
SELECT JESD204 family Xilinx,_Inc. 2.1
# END Select
# BEGIN Parameters
CSET c_component_name=jesd204_v1_1
CSET c_device_subclass=1
CSET c_jesd204b=1
CSET c_lanes=2
CSET c_lmfc_buffer_size=1024
CSET c_node_is_transmit=1
CSET c_use_bram=true
# END Parameters
GENERATE
```

Output Generation

Table 4-1: **XCO Parameters**

Parameter	XCO File Values	Default
c_component_name	ASCII text starting with a letter and based on the following character set:a..z,0..9 and _	jesd204_v1_1
c_node_is_transmit	1 = Transmit or 0= Receive or 2 = Share GTX	1
c_number_of_lanes	1, 2, 4, 8	1
c_device_subclass	0, 1, 2	0
c_lmfc_buffer_size	32, 64, 128, 256, 512, 1024	1024
c_use_bram	true, false	true

The output files generated from the CORE Generator tool are placed in the project directory. The list of output files includes:

- The netlist file for the core
- XCO files
- Release notes and documentation
- A Verilog example design and demonstration test bench
- Scripts to synthesize, implement and simulate the example design

See Chapter 9, Quick Start Example Design and Chapter 10, Detailed Example Design for a complete description of the CORE Generator output files and for details of the HDL example design.

Updating Transceiver Configuration

The generated lock-level design contains wrapper files instantiating the transceiver(s). These files are created using the appropriate Transceiver Wizard in the CORE Generator tool.

This section describes how to regenerate the transceiver wrapper files if necessary. This is normally not required but can be necessary when an update to the wizard output has been added after the core was last released.

This can be necessary if the Xilinx recommended transceiver configuration changes.

See the appropriate device family Transceiver User Guide for details:

- *7 Series FPGAs GTX Transceivers User Guide* [Ref 2]
- *Virtex-6 FPGA GTX Transceivers User Guide* [Ref 3]

7 Series FPGA Transceivers Wizard

Included in the core is an example instantiation of the GTXE2 transceiver targeted at 7v285tffg1157 (Virtex®-7) or 7k160tffg676 (Kintex™-7) device. The generated files are compatible with any Kintex-7 or Virtex-7 device.

To regenerate the transceiver wrapper files:

1. Using the CORE Generator tool, create a project using the appropriate target device (Kintex-7 or Virtex-7).
2. Run the 7 series FPGA Transceivers Wizard.
3. Use the default name for the component of `gtwizard_v2_1`.
4. Select the appropriate settings in the GUI.
5. Replace the following files in the `example_design` directory with the newly created versions from the CORE Generator project directory:
   - `gtwizard_v2_1_gt.v`
   - `gtwizard_v2_1.v`
6. Re-synthesize and re-implement.

Virtex-6 FPGA GTX Transceiver Wizard

Included in the core is an example instantiation of the GTXE1 transceiver targeted to a XC6VLX130T-FF1156 FPGA.

To regenerate the transceiver wrapper files:

1. Using the CORE Generator tool, run the Virtex-6 FPGA GTX Transceiver Wizard. Use the default name for the component of `v6_gtxwizard_v1_12`.
2. Select the appropriate settings in the GUI.
3. Replace the following files in the `example_design` directory with the newly created versions from the CORE Generator project directory.
   - v6_gtxwizard_v1_12_gtx.v
   - v6_gtxwizard_v1_12.v
4. Re-synthesize and re-implement.

XILINX®

Chapter 5

Designing with the Core

This chapter provides a general description of how to use the JESD204 core in your designs and should be used in conjunction with Chapter 6, Interfacing to the Core, which describes specific core interfaces.

General Design Guidelines

This section describes the steps required to turn a JESD204 core into a fully-functioning design with user-application logic. It is important to know that not all implementations require all of the design steps listed in this chapter. Follow the logic design guidelines in this manual carefully.

Use the Example Design as a Starting Point

Each instance of the JESD204 core created by the Xilinx® CORE Generator™ tool is delivered with an example design that can be implemented in an FPGA and simulated. This design can be used as a starting point for your own design or can be used to troubleshoot your application, if necessary.

See Chapter 9, Quick Start Example Design and Chapter 10, Detailed Example Design for information about using and customizing the example designs for the JESD204 core.

Know the Degree of Difficulty

JESD204 designs are challenging to implement in any technology, and the degree of difficulty is further influenced by:

- Maximum system clock frequency
- Targeted device architecture
- Nature of your application

All JESD204 implementations require careful attention to system performance requirements. Pipelining, logic mapping, placement constraints, and logic duplication are all methods that help boost system performance.

Keep It Registered

To simplify timing and increase system performance in an FPGA design, keep all inputs and outputs registered between your application and the core. This means that all inputs and outputs from your application should come from or connect to a flip-flop. While registering signals might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place-and-route the design.

Recognize Timing Critical Signals

The UCF provided with the example design for the core identifies the critical signals and the timing constraints that should be applied. See Chapter 7, Constraining the Core for further information.

Use Supported Design Flows

The core is synthesized in the CORE Generator tool and is delivered as Verilog. The example implementation scripts provided currently use XST as the synthesis tool for the Verilog example design that is delivered with the core. Other synthesis tools can be used.

Make Only Allowed Modifications

The JESD204 core is not user-modifiable. Any modifications can have adverse effects on system timing and protocol compliance. Supported user configurations of the JESD204 core can only be made by selecting the options from within the CORE Generator tool and using the top-level parameters described in this document. See Chapter 4, Customizing and Generating the Core.

XILINX®

Chapter 6

Interfacing to the Core

Block Level

Clocks and Resets

Table 6-1: **Clocks and Reset**

Signal Name	Direction	Description
refclk0p/n	In	Differential reference clock to transceiver. The reference clock must be set correctly for the selected line rate. See GTX Transceiver Clock Configuration for details.
refclk1p/n	In	A second differential reference clock is available for independent transmit and receive clock domains. See GTX Transceiver Clock Configuration for details.
tx_reset(1)	In	Asynchronous reset for transmit logic (transmit only).
rx_reset(1)	In	Asynchronous reset for receive logic (receive only).

1. See the *Xilinx 7 Series FPGAs Configurable Logic Block User Guide* [Ref 10] for signal specification.

JESD204 Serial Interface

GTX transceiver signals

See the *GTX Transceiver User Guide* [Ref 3] for more information.

Table 6-2: **GTX Transceiver Signals**

Signal Name	Direction	Description
txp[LANES-1:0]	Out	Positive differential JESD204 serial output from the serial transceiver (Transmit Only)
txn[LANES-1:0]	Out	Negative differential JESD204 serial output from the serial transceiver (Transmit Only)
rxp[LANES-1:0]	In	Positive differential JESD204 serial output to the serial transceiver (Receive Only)
rxn[LANES-1:0]	In	Negative differential JESD204 serial output to the serial transceiver (Receive Only)

Note: N = Number of Lanes

Transmit Data Interface

Table 6-3: **Transmit Data Interface**

Signal Name	Direction	Description
AXI4-Stream Interface Signals (Transmit Only) One per lane, N=0..(LANES-1)		
tx_aclk	Out	Clock (shared by all lanes). The clock output by the core depends on the selected line rate and varies from 25 MHz at 1 Gb/s line rates to 257.8 MHz at 10.3 Gb/s line rates.
tx_aresetn	Out	Active-Low reset (shared by all lanes)
txN_tdata[31:0]	In	AXI transmit data (samples and control words)
txN_tvalid	In	AXI transmit data valid. Ignored by the core because data is expected to be continuous.
txN_tready	Out	AXI slave ready for data
Non-AXI Data Interface Signals (Transmit Only)		
tx_start_of_frame[3:0]	Out	Frame boundary indication. The signal is 4 bits to indicate the byte position of the first byte of a frame in tdata in the following clock cycle. • When start_of_frame = 0001, the first byte of a frame is in bits [7:0] of the tdata word with the next 3 bytes in bits[31:8]. • When start_of_frame = 0010, the first byte is in bits [15:8] of the tdata word with the next 2 bytes in bits[31:16]; bits [7:0] contain the end of the previous frame. • When start_of_frame = 0100, the first byte is in bits [23:16] of the tdata word with the next byte in bits[31:24]; bits [15:0] contain the end of the previous frame. • When start_of_frame = 1000, tdata contains the last 3 bytes of the previous frame in bits [23:0] and the first byte of a new frame in bits [31:24].
tx_start_of_multiframe[3:0]	Out	Multi-frame boundary indication. The position of the last byte in a frame is encoded in the same way as start_of_frame.
tx_sysref_in	In	Sysref Input. When JESD204B and subclass 1 are selected, this signal is included in the design. JESD204B specifies a sysref signal must be generated synchronous to the frame clock. This input should be tied to sysref_out or driven from an external device generating sysref for both Tx and Rx.
tx_sync(1)	In	Sync signal
tx_sysref_out	Out	Sysref Output. When JESD204B and subclass 1 are selected, this signal is included in the design. JESD204B specifies a sysref signal must be generated synchronous to the frame clock. If this output is used, it must also be tied to the input sysref. If an external sysref is used, this signal can be left open.

1. See the JEDEC JESD204 specifications [Ref 4] and [Ref 5] for details of this signal.

Receive Data Interface

Table 6-4: **Receive Data Interface**

Signal Name	Direction	Description
AXI4-Stream Interface Signals (Receive Only) One per lane, N=0..(LANES-1)		
rx_aclk	Out	Clock (shared by all lanes)
rx_aresetn	Out	Active-Low reset (shared by all lanes)
rxN_tdata[31:0]	Out	AXI receive data (samples and control words)
rxN_tvalid	Out	AXI receive data valid
rxN_tready	In	AXI master ready for data
Non-AXI Data Interface Signals (Receive Only)		
rx_start_of_frame[3:0]	Out	Frame boundary indication. The position of the last byte in a frame is encoded in the same way as start_of_frame in the transmit section. This signal is asserted one cycle before the AXI4-Stream data.
rx_end_of_frame[3:0]	Out	Frame boundary indication. The position of the last byte in a frame is encoded in the same way as start_of_frame.
rx_frame_error[(LANES*4)-1 :0]	Out	Error in byte. JESD204 specifies that data must be replicated from the previous frame if certain errors occur. The core does not buffer the previous frame. The user can choose to implement a frame buffer or use a buffer elsewhere in the system to perform this function if required.
rx_sync	Out	Sync signal
rx_sysref_in	In	Sysref Input. When JESD204B and subclass 1 are selected, this signal is included in the design. JESD204B specifies a sysref signal must be generated synchronous to the frame clock. This input should be tied to sysref_out or driven from and external device generating sysref for both Tx and Rx.
rx_sysref_out	Out	Sysref Output. When JESD204B and subclass 1 are selected, this signal is included in the design. JESD204B specifies a sysref signal must be generated synchronous to the frame clock; generating the sysref from the core logic clock gives the most accuracy. If this output is used, it must also be tied to the input sysref. If an external sysref is used, this signal can be left open.

Management Interface (AXI4-Lite)

Table 6-5: **Management Interface (AXI4-Lite)**

Signal Name	Direction	Description
s_axi_aclk	In	Clock
s_axi_aresetn	In	Active-Low reset
s_axi_awaddr[31:0]	In	Write Address
s_axi_awvalid	In	Write Address Valid
s_axi_awready	Out	Write Address Ready
s_axi_wdata[31:0]	In	Write Data
s_axi_wstrb[3:0]	In	Write Data Byte Strobe
s_axi_wvalid	In	Write Data Valid
s_axi_wready	Out	Write Data Ready
s_axi_bresp[1:0]	Out	Write Response (Always = “00” = OK)
s_axi_bvalid	Out	Write Response Valid
s_axi_bready	In	Write Response Ready
s_axi_araddr[31:0]	In	Read Address
s_axi_arvalid	In	Read Address Valid
s_axi_arready	Out	Read Address Ready
s_axi_rdata[31:0]	Out	Read Data
s_axi_rresp[1:0]	Out	Read Response (Always = “00” = OK)
s_axi_rvalid	Out	Read Data Valid
s_axi_rready	In	Read Data Ready

AXI4-Lite Configuration Address Map

The AXI4-Lite CPU interface is used to configure the core. As the core can be configured as a transmitter or receiver, two separate address maps are used.

Transmit Configuration Address Map

See *JESD204* [Ref 4], Section 5.3.5.3 for register details.

Table 6-6: **Transmit Configuration Address Map**

Address	R/W	Description
0x00	R/W	Bit 0: Enable link synchronisation Bit 1: Reserved Bits 3-2: Test Mode Select Bit 4: Sysref Always (JESD204B Subclass 1 Only) When set to 1 the core responds to any SYSREF pulses; when set to 0 the core only resets the LMFC counters on the first sysref pulse detected and ignores subsequent sysref pulses. Bit 5: Sysref Request (JESD204B Subclass 1 Only). The FPGA can act as a SYSREF generator. A write of '1' to this bit sends a one-core clock cycle long high pulse on the sysref_out pin of the block level. This bit is self-clearing. Bit 6: Reserved Bit 7: Reset (write 1 to reset, read: 1: in progress, 0: complete) Bits 31-8 Reserved
0x01	R/W	Bits 7-0: Multi frames in the initial lane alignment sequence Bits 31-8: Reserved
0x02	R/W	Bits 7-0: DID[7:0] – Device ID Bits 11-8: BID[4:0] – Bank ID Bits 15-12: Adjust Count (JESD204B Subclass 2 Only) Bit 16: Phase Adjust Request (JESD204B Subclass 2 Only) Bit 17: Adjust Count Direction (JESD204B Subclass 2 Only) Bits 22-18: L[4:0] – Lanes Per Link (read only, returns core parameter) Bit 23: SCR[0] – Enable Scrambling Bits 31-24: F[7:0] – Octets per Frame
0x03	R/W	Bits 4-0 K[4:0] – Frames per Multi frame Bits 12-5: M[7:0] – Converters per Device Bits 17-13 N[4:0] – Converter Resolution Bits 19-18: CS[1:0] – Control bits per Sample Bits 24-20: N'[4:0] – Total Bits per Sample Bits 29-25 S[4:0] – Samples per converter per Frame Bit 30: HD[0] – High Density Format Bits 31: Reserved (read as 0)
0x04	R/W	Bits 7-0: RES1[7:0] – Reserved Field Bits 15-8: RES2[7:0] – Reserved Field Bits 23-16: Ignored – Checksum calculated automatically, read as 0 Bits 28-24: CF[4:0] – Control Words per Frame per Link Bits 31-29: Reserved (read as 0)

Receive Configuration Address Map

See *JESD204* [Ref 4], Section 5.3.5.3 for register details.

Table 6-7: **Receiver Configuration Address Map**

Address	R/W	Description
0x00	R/W	Bit 0: Enable scrambling Bit 1: Enable link synchronisation Bits 3-2: Test Mode Select Bit 4: Sysref Always (JESD204B Subclass 1 Only). When set to 1, the core responds to any SYSREF pulses; when set to 0, the core only resets the LMFC counters on the first sysref pulse detected and ignores subsequent sysref pulses. Bit 5: Sysref Request (JESD204B Subclass 1 Only). The FPGA can act as a SYSREF generator. A write of '1' to this bit sends a one core clock cycle long high pulse on the sysref_out pin of the block level. This bit is self-clearing. Bit 6: Reserved Bit 7: Reset. Writing 1 resets the receiver. This bit is self clearing. Bits 31-8: Reserved
0x01	R/W	Bits 7-0: Octets per Frame (F) Bits 31-8: Reserved
0x02	R/W	Bits 31-0: Reserved
0x03	R/W	JESD204 B Only. All bits ignored for JESD204A Bits 4-0: K, frames per multiframe. Bits 17-5: RX buffer delay input. Bit 18: Disable error reporting using the SYNC~ interface. Bits 31-19: Reserved.
0x04 – 0x06	R	Lane 0 Configuration fields extracted from init sequence
0x04	R	Bits 7-0: DID[7:0] Bits 12-8: BID[4:0] Bits 17-13: LID[4:0] Bits 22-18: L[4:0] Bit 23: SCR[0] Bits 31-24: F[7:0]
0x05	R	Bits 4-0: K[4:0] Bits 12-5: M[7:0] Bits 17-13: N[4:0] Bits 19-18: CS[1:0] Bits 24-20: N'[4:0] Bits 29-25: S[4:0] Bit 30: HD[0] Bits 31: Reserved (read as 0)
0x06	R	Bits 7-0: RES1[7:0] Bits 15-8: RES2[7:0] Bits 23-16: FCHK[7:0] Bits 28-24: CF[4:0] Bits 31-29: Reserved (read as 0)

Table 6-7: **Receiver Configuration Address Map *(Cont'd)***

Address	R/W	Description
0x07	R	Bits 3-0: ADJCNT[3:0] Bit 4: PHADJ Bit 5: ADJDIR Bit 8-6: JESDV[2:0] Bit 11-9: SUBCV[2:0] Bits 31:12: Reserved (read as 0)
0x08– 0x0B	R	Lane 1 Configuration fields extracted from init sequence (bit positions as per Lane 0 (registers 0x04-0x08)
0x0C– 0x0F	R	Lane 2 Configuration fields extracted from init sequence (bit positions as per Lane 0 (registers 0x04-0x08)
0x10– 0x13	R	Lane 3 Configuration fields extracted from init sequence (bit positions as per Lane 0 (registers 0x04-0x08)
0x14-0x17	R	Lane 4 Configuration fields extracted from the init sequence (bit positions as per Lane 0 (registers 0x04-0x08)
0x18-0x1B	R	Lane 5 Configuration fields extracted from the init sequence (bit positions as per Lane 0 (registers 0x04-0x08)
0x1C-0x1F	R	Lane 6 Configuration fields extracted from the init sequence (bit positions as per Lane 0 (registers 0x04-0x08)
0x20-0x23	R	Lane 7 Configuration fields extracted from the init sequence (bit positions as per Lane 0 (registers 0x04-0x08)
0x24	R	Test Mode Error Count (Lane 0). This is a 32 bit count which is incremented under the following conditions: Test Mode 0: When a single non 0xBC Comma is detected at the receiver. Test Mode 1: If a comma is detected after syncronisation has been acheived (as defined by rx_sync) and the ILA sequence has started and when an unexpected control byte is received. The counter is reset when test mode is exited or the core is reset.
0x25	R	Test Mode ILA Sequence Count (Lane 0). This is a 32 bit count which is incremented under the following conditions: Test Mode 1 is selected and a /Q/ character is detected marking the ILA boundary. The counter is reset when test mode is exited or the core is reset.
0x26	R	Test Mode Multi-Frame Count (Lane 0). This is a 32 bit count which is incremented under the following conditions: Test Mode 1 is selected and a /R/ character is detected marking the Multiframe boundary. The counter is reset when test mode is exited or the core is reset.
0x27	R	Test Mode Error Count (Lane 1)
0x28	R	Test Mode ILA Sequence Count (Lane 1)
0x29	R	Test Mode Multi-Frame Count (Lane 1)
0x2A	R	Test Mode Error Count (Lane 2)

Table 6-7: **Receiver Configuration Address Map *(Cont'd)***

Address	R/W	Description
0x2B	R	Test Mode ILA Sequence Count (Lane 2)
0x2C	R	Test Mode Multi-Frame Count (Lane 2)
0x2D	R	Test Mode Error Count (Lane 3)
0x2E	R	Test Mode ILA Sequence Count (Lane 3)
0x2F	R	Test Mode Multi-Frame Count (Lane 3)
0x30	R	Test Mode Error Count (Lane 4)
0x31	R	Test Mode ILA Sequence Count (Lane 4)
0x32	R	Test Mode Multi-Frame Count (Lane 4)
0x33	R	Test Mode Error Count (Lane 5)
0x34	R	Test Mode ILA Sequence Count (Lane 5)
0x35	R	Test Mode Multi-Frame Count (Lane 5)
0x36	R	Test Mode Error Count (Lane 6)
0x37	R	Test Mode ILA Sequence Count (Lane 6)
0x38	R	Test Mode Multi-Frame Count (Lane 6)
0x39	R	Test Mode Error Count (Lane 7)
0x3A	R	Test Mode ILA Sequence Count (Lane 7)
0x3B	R	Test Mode Multi-Frame Count (Lane 7)
0x 3C	R	rx_buffer_adjust. 8 bits per lane, bits 7:0 correspond to lane 0, bits 16:8 correspond to lane 2 etc.
0x3D	R	rx_buffer_adjust overflow for 8 lane configurations.

Core Interfaces

This section describes the core netlist interfaces. It is recommended users instantiate the block-level wrapper rather than use the netlist directly. Users can choose to modify the Block Level Wrapper in which case the following sections describe the core interfaces. This can be done for example if you want to remove the AXI4-Lite IPIF configuration interface.

Clocks and Resets

Table 6-8: **Clocks and Resets**

Signal Name	Direction	Description
clk	In	Clock for all FPGA logic. This clock should be the same signal used to drive the GTX transceiver userclk signals.
rst	In	Synchronous reset. Standard synchronous input to core, single cycle.

Data Interface

Table 6-9: **Data Interface**

Signal Name	Direction	Description
Transmit Only		
start_of_multiframe[3:0]	Out	Start of Multi Frame Indication to user.
txdatain[N*32-1:0]	In	User data in at quarter the byte rate (32 bits per lane). The least significant word is lane0 and the least significant byte of each 32-bit word is transmitted first.
txdata[N*32-1:0]	Out	Data to GTX transceiver. 32 bits per lane. The least significant word is lane0. The least significant byte of each 32-bit word is transmitted first.
txcharisk[N*4-1:0]	Out	Char is K signal to GTX transceiver. 4 bits per lane. The least significant 4 bits are lane0. The least significant bit of each nibble is transmitted first.
tx_sync	In	Sync Signal from receiver(s). Active-Low.
Receive Only		
rxdataout[N*32-1:0]	Out	User data out at quarter the byte clock rate (32 bits per lane). The least significant word is lane0 and the least significant byte of each 32-bit word was received first.
rxdata[N*32-1:0]	In	Data from GTX transceiver. 32 bits per lane. The least significant word is lane0. The least significant byte of each 32-bit word was received first.
rxcharisk[N*4-1:0]	In	Char is K signal from GTX transceiver. 4 bits per lane. The least significant 4 bits are lane0. The least significant bit of each nibble was received first.
rxdisperr[N*4-1:0]	In	Disparity error signal from GTX transceiver. 4 bits per lane. The least significant 4 bits are lane0. The least significant bit of each pair was received first.
rxnotintable[N*4-1:0]	In	Not in table error signal from GTX transceiver. 4 bits per lane. The least significant 4 bits are lane0. The least significant bit of each nibble was received first.
rx_sync[3:0]	Out	Sync signal from receiver (4 bits give byte resolution). Active-Low.
rxencommaalign	Out	Enable comma alignment signal to transceiver.
enchanbond	Out	Enable channel bonding signal to transceiver.
chanbondseq	In	Channel bonding sequence indication from transciever.
frame_error[N*4-1:0]	Out	Error in byte indication to user.
Common Signals		
start_of_frame[3:0]	Out	Start of Frame Indication to user.
end_of_frame[3:0]	Out	End of Frame Indication to user.

Configuration Signals (Transmit Only)

See *JESD204* [Ref 4], Section 5.3.5.3 for details of the encoding used on the signals named `tx_cfg_*`.

Table 6-10: **Configuration Signals (Transmit Only)**

Signal Name	Direction	Description
multi_frames[7:0]	In	Number of Multiframes in Initial Lane Sync Sequence. A value of between 3 and 255 on this bus corresponds to between 4 to 256 Multiframes. Do not set to less than 3.
support_lane_sync	In	Enable Lane Synchronisation. 1 = Enabled; 0 = Disabled.
tx_cfg_f[7:0]	In	Number of Octets per Frame. A value on the bus between 0 and 255 corresponds to between 1 and 256 octets per frame.
tx_cfg_k[4:0]	In	Number of Frames per Multiframe. A value on the bus between 0 and 31 corresponds to between 1 and 32 frames per multiframe.
tx_cfg_scr	In	Enable scrambling. 0 = Disabled; 1 = Enabled
tx_cfg_did[7:0]	In	Device ID (0 to 255)
tx_cfg_bid[3:0]	In	Bank ID (0 to 15)
tx_cfg_m[7:0]	In	Number of Converters per Device. A value on the bus between 0 and 255 corresponds to between 1 and 256 converters per device.
tx_cfg_cs[1:0]	In	Number of Control Bits per Sample. (0 to 3)
tx_cfg_n[4:0]	In	Converter Resolution. A value on the bus between 0 and 31 corresponds to a resolution between 1 and 32.
tx_cfg_np[4:0]	In	Total number of bits per Sample. A value on the bus between 0 and 31 corresponds to between 1 and 32 bits per sample.
tx_cfg_s[4:0]	In	Number of Samples per Converter per Frame Cycle. A value on the bus between 0 and 31 corresponds to between 1 and 32 samples per converter per frame cycle.
tx_cfg_hd	In	High Density Format
tx_cfg_res1[7:0]	In	Reserved Field 1
tx_cfg_res2[7:0]	In	Reserved Field 2
tx_cfg_cf[4:0]	In	Number of Control Words per Frame Cycle per Link.
JESD204B Only		
sysref_in	In	Subclass 1 only. Active-High SYSREF input.
sysref_out	Out	Subclass 1 only. Active-High SYSREF output. See SYSREF Timing.
tx_cfg_adjcnt[3:0]	In	Subclass 2 only. Transmitter adjustment resolution step count. This is transmitted in the second configuration octet in the ILA.

Table 6-10: **Configuration Signals (Transmit Only) *(Cont'd)***

Signal Name	Direction	Description
tx_cfg_adjdir	In	In Subclass 2 only. Transmitter adjustment direction. This is transmitted in the third configuration octet in the ILA.
tx_cfg_phadj	In	Subclass 2 only. Transmitter adjustment request. This is transmitted in the third configuration octet in the ILA.
tx_cfg_cs_all	In	Set to 1 to force the ILA checksum to be performed over all 13 bytes of the configuration data block. Set to 0 to checksum over only the relevant fields in the configuration block as per JESD204A and JESD204B specification.
sysref_always	In	Subclass 1 only. Control bit describing the receiver SYSREF behavior. When set to zero, the JESD204B transmitter ignores all SYSREF pulses after the first one received during link initialization. When set to one the transmitter re-aligns its LMFC counters to all SYSREF pulses received during link initialization.

Configuration Signals (Receive Only)

Table 6-11: **Configuration Signals (Receive Only)**

Signal Name	Direction	Description
octets_per_frame[7:0]	In	Octets Per Frame. A value on the bus between 0 and 255 corresponds to between 1 and 256 octets per frame.
scram_enable	In	Enable Descrambling. 1 = Enabled; 0 = disabled
init0[N*32-1:0]	Out	ILS Data Word(s) 0 (See Table 6-12 for details)
init1[N*32-1:0]	Out	ILS Data Word(s) 1 (See Table 6-13 for details)
init2[N*32-1:0]	Out	ILS Data Word(s) 2 (See Table 6-14 for details)
JESD204B Only		
init3[N*32-1:0]	Out	ILS Data Word(s) 3 (See Table 6-15 for details)
sysref_in	In	Subclass 1 only. Active-High SYSREF input.
sysref_out	Out	Subclass 1 only. Active-High SYSREF output.
frames_per_multiframe[4:0]	In	Number of frames in the JESD204B multiframe.
rx_buffer_delay[7:0]	In	Difference in the latency between the output of data and the multiframe size in octets. See Minimum Deterministic Latency Support for more information.

Table 6-11: **Configuration Signals (Receive Only) *(Cont'd)***

Signal Name	Direction	Description
rx_buffer_adjust [(C_LANES*8)-1:0]	Out	The maximum allowable reduction in the JESD204B latency in octets. Each lane outputs an 8-bit number describing the difference between the read and write pointers in the JESD204B buffers. The lowest value can be input of the rx_buffer_delay port to adjust the latency. See Minimum Deterministic Latency Support for more information.
disable_error_reporting	In	Control bit determining the behavior of the SYNC~ interface error reporting. When this bit is set to one, errors are not communicated using the SYNC~ interface. See Error Signalling Using the SYNC~ Interface for more information.
sysref_always	In	Subclass 1 only. Control bit describing the receiver SYSREF behavior. When set to zero the JESD204B receiver ignores all SYSREF pulses after the first one received during link initialization. When set to one, the receiver re-aligns its LMFC counters to all SYSREF pulses received during link initialization.

Initial Lane Synchronisation Data

Signals init0, init1 and init2 contain all the configuration data extracted from the received Initial Lane Synchronisation Data. Each signal is 32*Lanes wide.

Init0 Bit Description

See *JESD204* [Ref 4], Section 5.3.5.3 for register details.

Table 6-12: **Init0 Bit Description**

Bits	Name	Description
Lane 0		
7:0	DID	Device ID
12:8	BID	Bank ID
17:13	LID	Lane ID
22:18	L	Number of Lanes per Device
23	SCR	Scrambling Enabled
31:24	F	Octets per Frame
Lanes 1 to 7		
32:63		Lane 1 configuration, bit positions as lane 0
64:95		Lane 2 configuration, bit positions as lane 0
96:127		Lane 3 configuration, bit positions as lane 0
128:159		Lane 4 configuration, bit positions as lane 0
160:191		Lane 5 configuration, bit positions as lane 0

Table 6-12: **Init0 Bit Description *(Cont'd)***

Bits	Name	Description
192:223		Lane 6 configuration, bit positions as lane 0
224:255		Lane 7 configuration, bit positions as lane 0

Init1 Bit Description

Table 6-13: **Init1 Bit Description**

Bits	Name	Description
Lane 0		
4:0	K	Frames per Multiframe
12:5	M	Converters per Device
17:13	N	Converter Resolution
19:18	CS	Control Bits per Sample
24:20	N′	Total Bits per Sample
29:25	S	Samples per Converter per Frame Cycle
30	HD	High Density Format
31		Not used, Read as zero
Lanes 1 to 7		
32:63		Lane 1 configuration, bit positions as lane 0
64:95		Lane 2 configuration, bit positions as lane 0
96:127		Lane 3 configuration, bit positions as lane 0
128:159		Lane 4 configuration, bit positions as lane 0
160:191		Lane 5 configuration, bit positions as lane 0
192:223		Lane 6 configuration, bit positions as lane 0
224:255		Lane 7 configuration, bit positions as lane 0

Init2 Bit Description

Table 6-14: **Init2 Bit Description**

Bits	Name	Description
Lane 0		
7:0	RES1	Reserved Field 1
15:8	RES2	Reserved Field 2
23:16	FCHK	Checksum
28:24	CF	Control Words per Frame Cycle per Link
31:29		Not Used, Read 0
Lanes 1 to 7		
32:63		Lane 1 configuration, bit positions as lane 0
64:95		Lane 2 configuration, bit positions as lane 0
96:127		Lane 3 configuration, bit positions as lane 0
128:159		Lane 4 configuration, bit positions as lane 0
160:191		Lane 5 configuration, bit positions as lane 0
192:223		Lane 6 configuration, bit positions as lane 0
224:255		Lane 7 configuration, bit positions as lane 0

Table 6-15: **Init3 Bit Description**

Bits	Name	Description
Lane 0		
3:0	ADJCNT	Adjustment step count
4	PHYADJ	Adjustment request
5	ADJDIR	Adjustment direction
8:6	JESDV	JESD204 version
11:9	SUBCLASSV	JESD204 subclass version
31:12	Reserved	
Lanes 1 to 7		
32:63		Lane 1 configuration, bit positions as lane 0
64:95		Lane 2 configuration, bit positions as lane 0
96:127		Lane 3 configuration, bit positions as lane 0
128:159		Lane 4 configuration, bit positions as lane 0
160:191		Lane 5 configuration, bit positions as lane 0
192:223		Lane 6 configuration, bit positions as lane 0

XILINX®

Chapter 7

Constraining the Core

This chapter describes how to constrain a design containing the JESD204 core. This is accomplished by using the UCF delivered with the core at generation time. See Chapter 10, Detailed Example Design, for a complete description of the Xilinx® CORE Generator™ output files.

Caution! Not all constraints are relevant to specific implementations of the core; consult the UCF created with the core instance to see exactly what constraints are relevant.

Device, Package, and Speed Grade Selection

This line of the UCF selects the part to be used in the implementation run. Edit the text in this line so that it matches the part intended for the final application.

```
# Select the part to be used in the implementation run
CONFIG PART = 6vlx75tff784-1;
```

The JESD204 core can be implemented in these Xilinx devices:

- Kintex™-7 devices, speed grade -1 or higher
- Virtex®-7 devices, speed grade -1 or higher
- Virtex-6 LXT devices, speed grade of -1 or higher
- Virtex-6 SXT devices, speed grade of -1 or higher
- Virtex-6 HXT devices, speed grade of -1 or higher
- Virtex-6 CXT devices, speed grade of -1 or higher

Clock Frequencies, Clock Management, and Placement

The reference clock and core clock frequency constraints vary depending on the selected line rate and reference clock when generating the core. See the generated UCF for details.

Other Constraints

There are also several paths where clock domains are crossed. These include the management interface. These are all constrained in the example UCF. See the example `design.ucf` for a comprehensive list of all timing constraints.

Chapter 8

Design Considerations

The JESD204 IP core supports both JESD204A and JESD204B with JESD204B being recommended for all new designs. JESD204B has three subclasses; subclass 0 is backwards compatible with JESD204A; subclass 1 and 2 add support for deterministic latency over the link. Subclass 1 is recommended for designs requiring deterministic latency.

GTX Transceiver Clock Configuration

JESD204B

Default Clocking

There are several options for clocking the transceiver and core in JESD204B. The JESD204B specification defines a device clock which can be faster or slower than the byte clock. The JESD204 core has a fixed 32-bit internal datapath so the FPGA logic always operates with a clock which is one quarter of the byte clock. The GTX transceiver cannot operate with a very slow reference clock so, for line rates less than 4 Gb/s, the reference clock must be faster than the FPGA logic clock. The transceiver wrapper provided with the core uses the TXOUTCLK or RXOUTCLK from the GTX transceiver to supply the correct clock to the logic. See Figure 8-1.

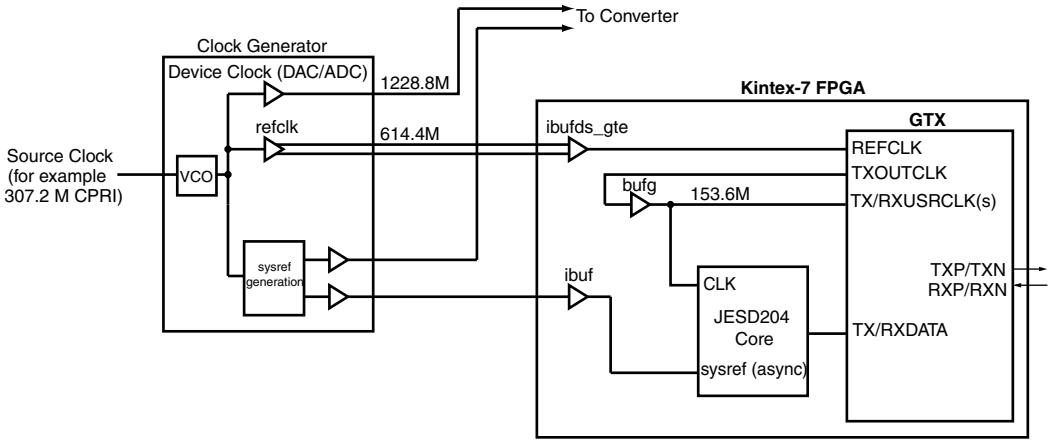

Figure 8-1: **Default JESD204B Clocking Scheme for 6.25 Gb/s Line Rate**

Advanced Clocking

Advanced clocking allows improved latency measurement accuracy over the default clocking scheme. These modifications need to be made manually to the block level wrapper and transceiver wrapper provided with the core. Contact Xilinx before making use of advanced clocking.

If accurate latency adjustment is required using JESD204B subclass 1 or 2 then an advanced clocking scheme should be used to ensure that `sysref` (or `sync` in the case of subclass 2) is captured synchronously to the core clock. Figure 8-2 shows an alternative clocking scheme which allows `sysref` to be captured synchronously. This scheme also allows any line rate to be used because the reference clock can be faster than the FPGA logic clock.

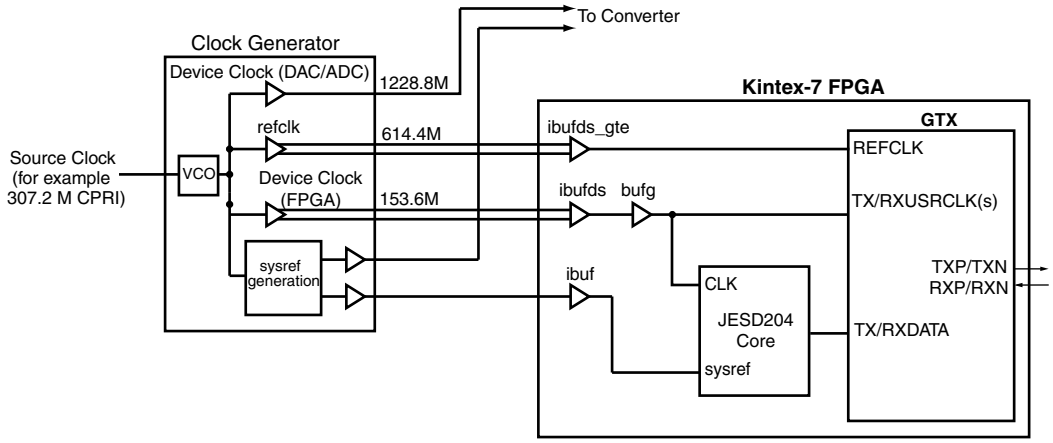

Figure 8-2: **Multiple Clocks for Capturing Sysref Synchronously**

The clock generator in Figure 8-2 must generate multiple clocks of different frequency which are edge-aligned. This clock configuration does not support frame formats where the multiframe boundary is not an exact multiple of 4 bytes because of the 4 byte internal datapath of the JEDS204 core.

If the line rate is over 4 Gb/s then the reference clock can be used directly by the logic as shown in Figure 8-3. Contact Xilinx for more information.

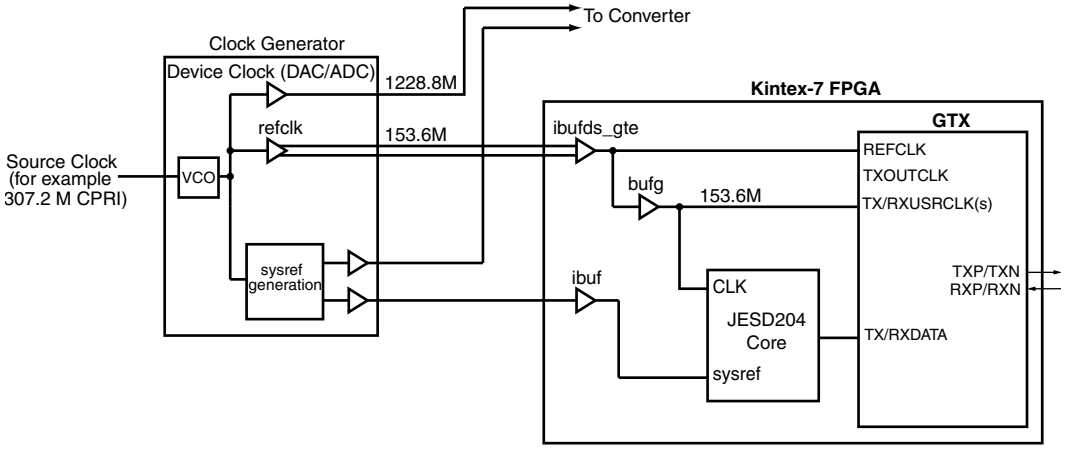

Figure 8-3: **Single Clock for Capturing Sysref Synchronously**

JESD204A

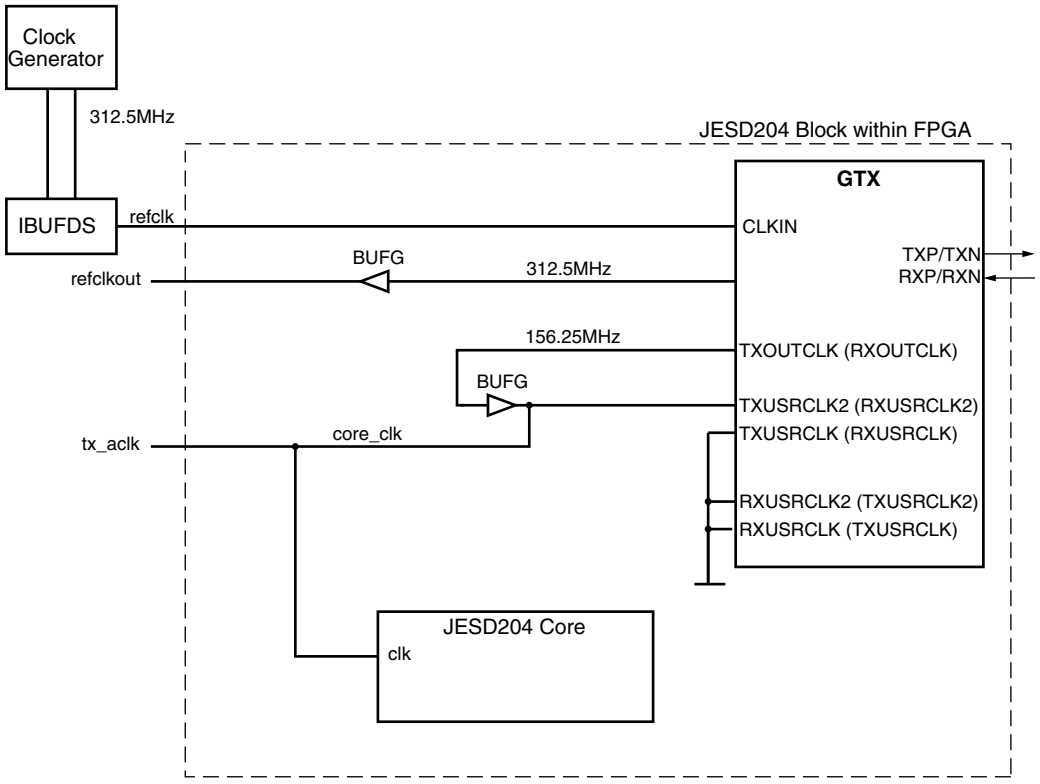

Figure 8-4: **GTX Tx Clocks and Core (Rx in brackets)**

Reference Clock Selection

JESD204A specifies that the Frame Clock is distributed to each device. The Frame Clock can be much slower than the Reference Clock requirements of the GTX transceiver which requires a High Frequency, Low Jitter Reference for best performance.

If the number of octets per frame is low, the Frame Clock can be used as the reference clock directly. If the number of octets per frame is high and the JESD204A is followed exactly, a clean-up PLL is required to generate a suitable reference clock from the available Frame Clock. Figure 8-5 shows some possible clock configurations.

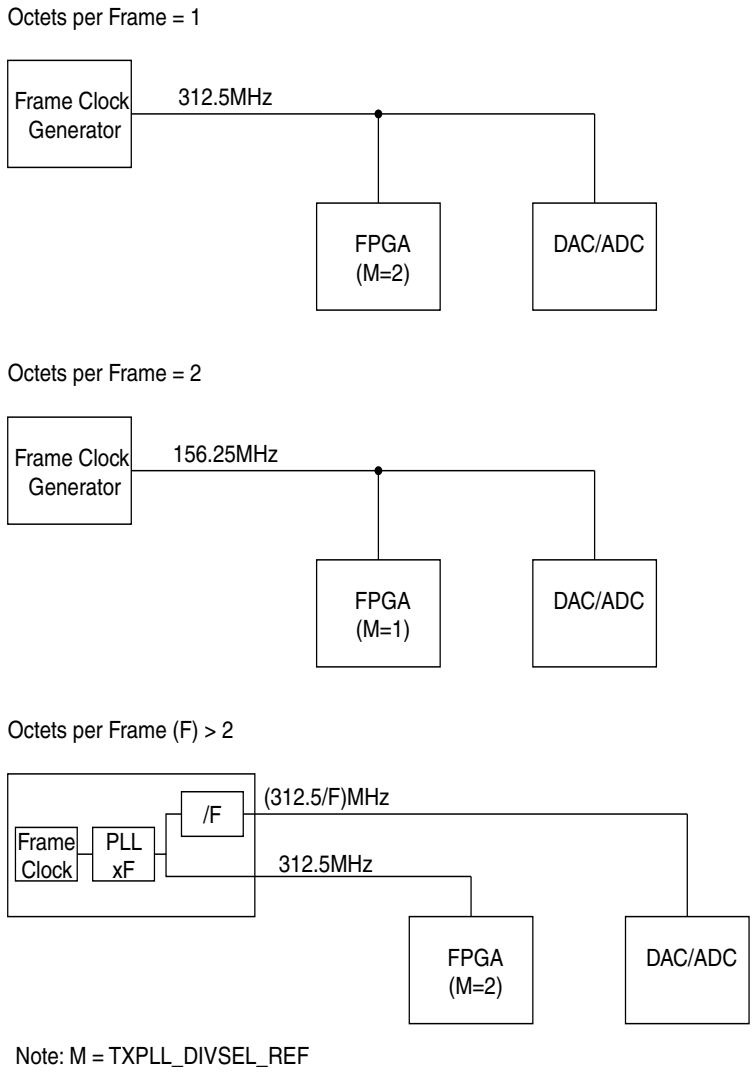

Figure 8-5: **JESD204A Clocks System**

Example:

Line Rate: 3.125 Gb/s

Device Clock: 78.125 MHz (The JESD204 IP has a 32-bit internal datapath.)

Optimal GTX transceiver Reference Clock: 312.5 MHz (See the *7 Series* or *Virtex-6 FPGA GTX Transceiver User Guide* for alternatives [Ref 2],[Ref 3].

Changing the setting of F in the core does not automatically adjust the GTX transceiver configuration. The GTX transceiver must be configured for the correct reference clock using the correct attributes before building the bit file. The default configuration uses a reference clock equal to the frame clock (312.5 MHz in this example).

The core can be configured for other values of F as long as the reference clock is multiplied up as per the bottom configuration shown in Figure 8-5. Contact Xilinx for help on configuring the GTX transceiver for other reference clock frequencies.

Table 8-1: **Example**

Octets per Frame	Frame Clock (MHz)	PLL Requirement
1	312.5	No PLL required. The Frame Clock can be used directly as a Reference Clock.
2	156.25	No PLL required. 156.25 MHz is sub-optimal but still meets the requirements of the GTX transceiver for operation at 3.125 Gb/s [Ref 2][Ref 3].
3	104.167	104.167 MHz is a non-integer multiple of the required reference clock and an external PLL is required to generate a 312.5 MHz Reference Clock.
4+	≤78.125	The Frame Clock frequency is too low to be used as a reference clock and an external PLL is required.

In practice the system designer is likely to implement the clean-up PLL function in one of the following ways:

- Use a DAC or ADC device capable of generating a Reference Clock directly. Devices need to generate an internal Sample Clock from the Frame Clock which is suitable for use as a reference Clock for the GTX transceiver if it is available externally.
- By using a Clock Generator device capable of generating both a Reference Clock and Frame Clock derived from the same source.

Line Rate and REFCLK (JESD204A & JESD204B)

The JESD204 IP supports any line rate supported by the chosen part in the range 1 Gb/s to 10 Gb/s. When the core is generated the transceivers are configured for operation at 6.25 Gb/s using a REFCLK of 156.25 MHz. If your target line rate is different from the default then the GTX transceiver settings must be adjusted accordingly. The preferred method to update the transceiver settings is to use the Transceiver Wizard as follows:

1. Using the CORE Generator™ tool create a new project, selecting the correct part, package and speed grade for your design. Select 'Verilog' generation.
2. Select Project>Import Existing Customised IP and open the XCO provided with the JESD204 IP in the `example_design` directory.
3. Select 'Recustomize and Regenerate' (under Current Project Settings)
4. Update the Line Rate and Reference Clock only on the first page.
5. Update the number of transceivers on the first page (second page in the Virtex-6 Wizard) to match the value selected in your JESD204 core. The XY position of the transceiver does not matter because the JESD204 core does not use the constraints generated by the Transceiver Wizard.
6. Click Finish to generate the Transceiver Wrappers.
7. Copy `<component_name>_gtwizard_v2_1.v` and `<component_name>_gtwizard_gt_v2_1` for 7 series designs or `<component_name>_v6_gtxwizard_1_12.v` and `<component_name>_v6_gtxwizard_1_12_gt.v` to the `example_design` directory of your JESD204 IP, replacing the existing files of the same name.
8. Rebuild the design.

JESD204B Implementation

The JESD204 core supports the JESD204B revision to the JESD204A specification as described in [Ref 5]. Device subclasses 0, 1 and 2 are supported.

Device Subclass 0

Devices of subclass zero offer no support for deterministic latency. The device maintains backwards compatibility with the JESD204A standard while adding support for lane operation up to 6.25 Gb/s.

Device Subclass 1

In addition to supporting lane operation at up to 6.25 Gb/s, devices of subclass 1 support deterministic latency using the SYSREF interface. Figure 8-6 shows the timing relationships between the signals on a JESD204B link.

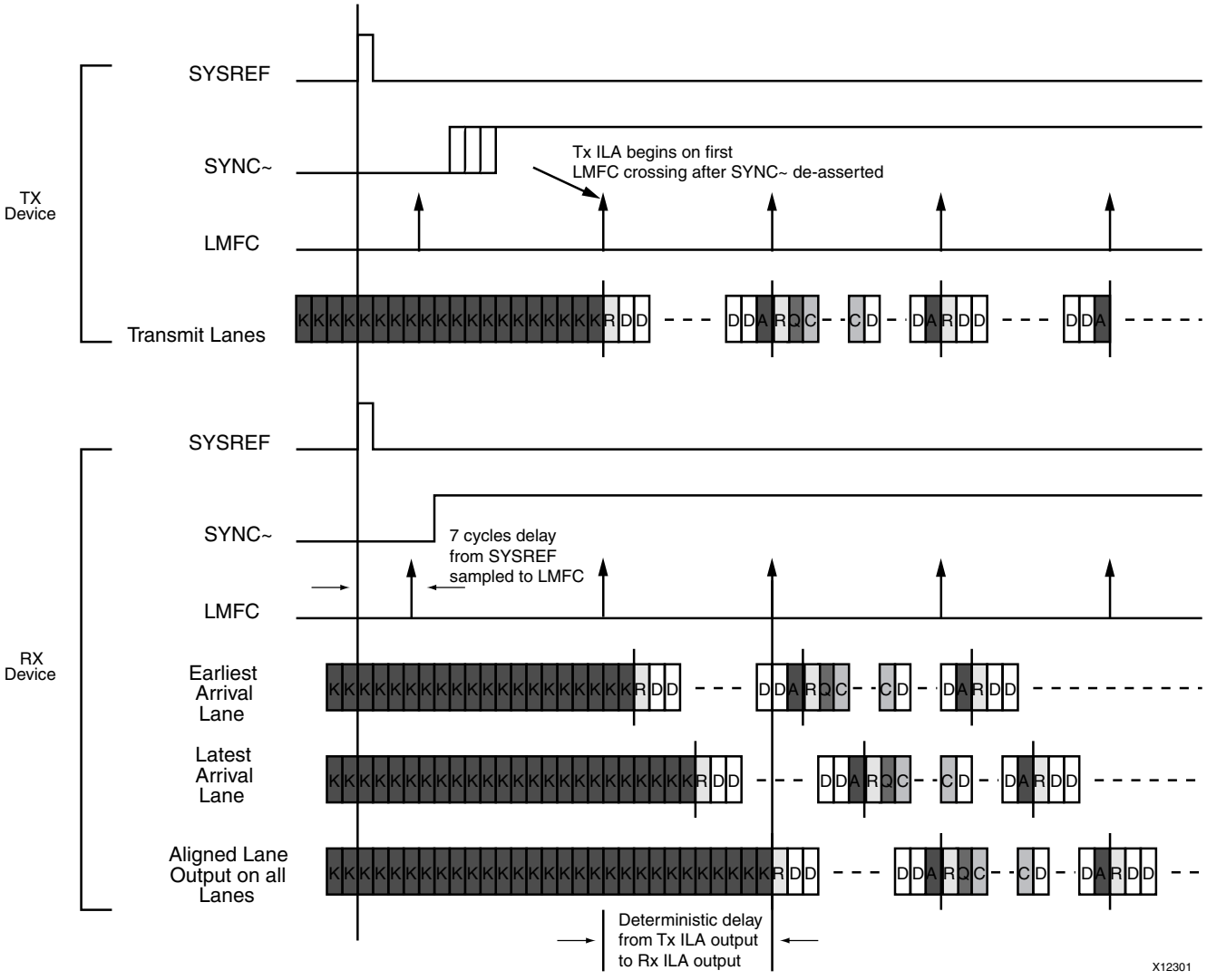

Figure 8-6: **Deterministic Latency across the JESD204B Link**

The SYSREF signal is distributed to all devices in a system. On assertion, the JESD204 receiver aligns its internal local multiframe clock (LMFC) to the incoming SYSREF signal and deasserts SYNC~. The latency between the sampling of SYSREF high and the deassertion of SYNC~ in the JESD204B core receiver is seven device clock cycles. The LMFC counts through the octets in a multiframe. A LMFC crossing occurs at the end of each multiframe. The device clock for the FPGA is the fabric clock (tx_aclk or rx_aclk).

When the transmit device detects the SYNC~ signal going low, it waits until the next LMFC crossing and starts transmitting data. If ILA generation is supported, then this is the ILA sequence; otherwise it is a normal frame data. When the receiver detects that all lanes have valid data, it buffers the data until the next LMFC crossing. Now the buffers are released and the data sent to the client interface. In this manner the latency between the data output and the SYSREF signal can be accurately ascertained.

The length of the multiframe must be larger than the longest possible delay across any lane in the JESD204B link. In Virtex®-6 FPGA implementations the transmitter contributes 18 core clock cycles to the latency of the link and the receiver contributes 20 core clock cycles. In Virtex-7 and Kintex™-7 device implementations the transmitter latency is 22 core clock cycles and the receiver latency is 29 core clock cycles. Each core clock cycle is equivalent to 4 octets.

SYSREF Timing

When JESD204B is used in Subclass 1, the SYSREF signal is the master timing reference for the system. There are two ways to implement a Subclass 1 system using the JESD204 core.

- FPGA Generates SYSREF

  The most accurate timing can be achieved if the FPGA is used to generate the SYSREF signal. sysref_out is provided for this purpose and can be triggered under control of the AXI4-Lite configuration interface. sysref_out is synchronous to the core clock (tx_aclk or rx_aclk) domain so it removes the uncertainty shown in Figure 8-7. If sysref_out is used then it should be connected to an output pad on the FPGA and also looped back to the sysref input.

  When SYSREF is generated by the FPGA, the internal core clock (tx_aclk or rx_aclk) can also be output to the system clock generator to ensure the ADC or DAC receives a frame clock synchronized to the internal core clock.

- External SYSREF generation

  The FPGA uses the GTX transceiver reference clock as the master clock (phase locked to the frame clock) for the system and generates a core clock internally. There is a phase uncertainty between the reference clock and the core clock. When SYSREF is generated externally, it is likely to be synchronous to the reference clock and not the internal core clock but the SYSREF signal must be captured and passed from the reference clock domain to the core clock (tx_aclk or rx_aclk) domain. Figure 8-7 shows the possible uncertainty between the SYSREF input and the captured internal SYSREF signal.

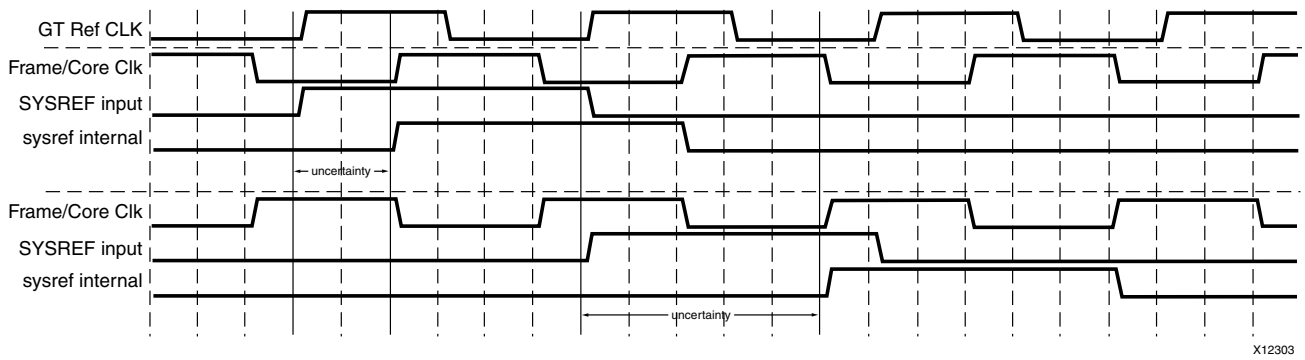

Figure 8-7: **SYSREF Generation**

In addition ensure that SYSREF is held active long enough to be captured by the slow core clock if it is only active for a single cycle of a fast reference clock. This can be achieved by ensuring that SYSREF is held active for an extended period by the external clock generator or by stretching the SYSREF after capturing in the reference clock domain; refclkout is made available by the core for this purpose. The logic required to pass SYSREF cleanly to the core clock domain is not included in the core or Block Level logic.

Device Subclass 2

The operation of the JESD204B circuit in subclass 2 mode is similar to a device in subclass 1 mode. In this case deterministic latency across the link is achieved using the SYNC~ interface. When the receiver has aligned to the incoming idle characters from the transmitter, it deasserts the SYNC~ signal. The transmitter detects this and waits until the next LMFC crossing before transmitting data or the ILA sequence (depending on the setting of the enable link synchronization control bit). The receiver buffers the data at its input until the next LMFC crossing at its side of the link, before sending the received data to the client.

In subclass 2 devices the transmitter and receiver latencies are as listed in the previous subclass 1 section.

JESD204B Receiver

Elastic Buffer Implementation

In subclass 1 and subclass 2 devices the receiver elastic buffer can be implemented in block memory or distributed memory. To use the block RAM implementation the 'Use BRAM' GUI checkbox should be enabled. In addition, you can select the size of the elastic buffer. The 'Octets in Multiframe' GUI option should be set to a size that holds a full multiframe of data.

Minimum Deterministic Latency Support

When the JESD204B link is first established in subclass 1 and subclass 2 devices, the receiver outputs data as shown in Figure 8-8. Data is output on the LMFC crossing after valid data is detected in all lanes. It is possible to support minimum latency by adjusting the number of octets input on the rx_buffer_delay control port.

An indication of the maximum allowable reduction of the latency is output on the rx_buffer_adjust port. This provides an indication of the difference between the write

and read pointers of the receiver elastic buffer in each lane. The number of octets output in bits 7:0 give an indication of the buffer fill level in lane 0; bits 15:8 refer to lane 1; bits 23:16 refer to lane 2 and bits 31:24 refer to lane 3. The lowest number given can be input to the `rx_buffer_delay` port to reduce the overall latency by that number of octets.

A full link resynchronization cycle must take place before the modified latency setting is acted upon. A minimum latency example is shown in Figure 8-8.

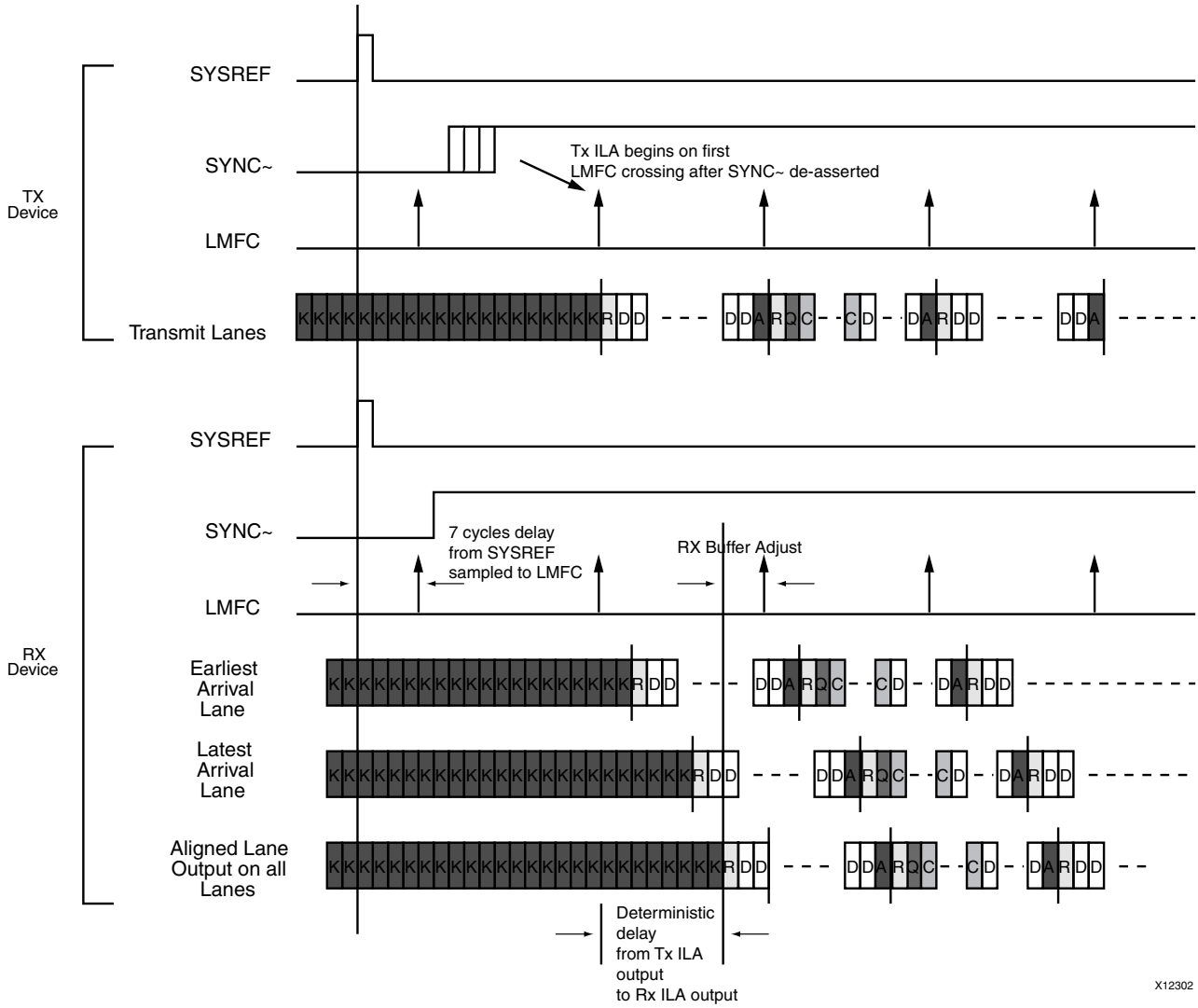

Figure 8-8: **Minimum Deterministic Latency**

Error Signalling Using the SYNC~ Interface

In the JESD204 core the `SYNC~` signal can be deasserted for two frame periods at the end of each multiframe to indicate an error. This behavior is enabled by the deassertion of the `disable_error_reporting` control bit.

The `SYNC~` signal is a 4-bit output; this enables the correct error output for a frame size of down to one octet. The signal is low when the receiver has detected an error. An error occurs when an idle or an unexpected control character is received during normal frame

transmission. Error signalling using the SYNC~ interface is not supported in JESD204A devices.

Figure 8-9 shows an example of error reporting on the SYNC~ interface for a frame size of three octets and where the multiframe size is a multiple of four octets. When one or more lanes receive an unexpected control character (in this case /I/) the SYNC interface is driven low for the last two frame periods in the subsequent multiframe.

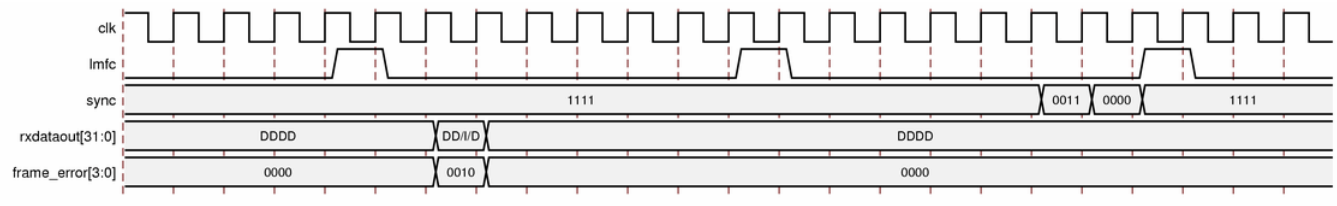

Figure 8-9: **Error reporting Using the SYNC~ Interface**

To give a frame-accurate SYNC~ output the 4-bit SYNC~ signal should be serialized on the frame clock before being output to the ADC. If this is not necessary, the bits can be combined in the block level of the example design to giving a single bit SYNC~ output that is synchronous to the core clock.

Link Re-initialization Using the SYNC~ Interface

To signal a link re-initialization request, the receiver deasserts SYNC~ for a period of five frame periods and nine octets. When this occurs, the transmitter should fall back out of data transmission mode and re-negotiate the link as described in the previous device subclass sections.

JESD204B Transmitter

Transmitter Phase Adjustment

The JESD204B subclass 2 transmitter LogiCORE™ IP provides ports to enable the alignment of the frame clock and LMFC internal to the FPGA to those of the DAC receiver. The required clock and LMFC phase adjustments are communicated to the DAC using the tx_cfg_adjcnt, tx_cfg_adjdir and tx_cfg_phadj ports listed in Table 6-10. These ports can also be accessed through a JESD204 configuration register as detailed in Table 6-7.

In subclass 2 devices the DAC receiver deasserts the SYNC~ signal on its internal LMFC boundary. At the FPGA transmitter it is the responsibility of the client logic to detect the deassertion of the SYNC~ signal and to compare its timing to that of the transmitter LMFC. If adjustment of the DAC LMFC phase is required, the client inputs the required adjustment step count to the tx_cfg_adjcnt port. The direction of the phase adjustment is input on tx_cfg_adjdir and an adjustment request is signalled by the assertion of the tx_cfg_phadj input.

The values on the phase adjustment ports are embedded in bytes 1 and 2 of the ILA sequence that is sent to the DAC during link initialization. On the reception of the ILA the DAC adjusts its LMFC phase by the step count value and sends back an error report with the new LMFC phase information. This process can be repeated until the LMFC at the DAC and the FPGA are aligned.

Sharing Transceivers between Transmit and Receive

Description

The JESD204 interface is unidirectional but the JESD204 IP does allow the user to share a single transceiver or group of transceivers between a transmit interface and a receive interface. The user is required to use the Transceiver Wizard and make the RTL modifications to the Block Level Wrapper provided with the JESD204 IP core to accomplish this. Also, to create a shared transceiver core, run the CORE Generator tool twice to create a core for the transmit path and a second core for the receive path.

Creating a Shared Transceiver Core

When 'Share Transceiver' is selected in the JESD204 GUI only a transmit core is created but the Example Design and Block Level include all the transceiver and clocking logic for both a transmit and receive core and a placeholder to instantiate a receive core. The following steps must be carried out to create a shared transceiver core:

1. Generate a core with 'Share Transceiver' selected and all other options set correctly for a transmitter.
2. Generate a core with Receive selected with the correct settings for the receiver. Choose a different component name from the first core (the placeholder instantiation expects the name to be the same as the first core with '_rx' appended).
3. Modify the `<component_name>_block.v` file generated by the shared transceiver core to instantiate the receive core.
4. Copy `<component_name>_rx_mod.v` from the receiver core `example_design` directory to the `example_design` directory of the shared transceiver core.
5. Update the constraints provided with the shared transceiver core. Uncomment the constraints for the receive portion of the design (see the comments in the UCF).
6. Build the Duplex design. The example scripts expect the NGC file for the receive core to be in the same directory as the shared transceiver core. If this is not the case then the receive core NGC file should be copied manually.

Creating a core with a different number of transmit and receive lanes is possible but requires modification of the Block Level which is beyond the scope of these instructions.

PLL Description

The following sections describe advanced features of the transceiver. The user is expected to understand the clocking features of the transceiver (see the *7 Series* or *Virtex-6 FPGA GTX Transceiver User Guide* [Ref 2][Ref 3] for details), and be familiar with the Transceiver Wizard.

Virtex-6

The separate transmit and receive PLLs allow independent operation of the transmitter and receiver. The transceiver wrapper provided with the JESD204 IP core has REFCLK0 connected to the transmit PLL and REFCLK1 connected to the receive PLL by default.

The preferred way of updating the transceiver settings is to use the Transceiver Wizard in the same way as described in Line Rate and REFCLK (JESD204A & JESD204B) but select different line rates and reference clocks for transmit and receive.

7 Series

The GTX transceiver in 7 series devices has a single channel PLL for each transceiver which is shared between the transmitter and receiver and also a common quad PLL shared between a bank of 4 transceivers. The channel PLL and quad PLL have different operating ranges which limit the line rate selection between transmit and receive when sharing a transceiver. See the *7 series Data Sheets* [Ref 6] for the exact operating ranges for a particular device/package/speedgrade.

When Duplex is selected when the JESD204 IP is generated, the channel PLL is used for transmit and the quad PLL for receive. This can be changed by modifying `gtxwizard_v2_1_top.v` in the `example_design` directory. See the comments in the sections of RTL tagged with 'Modify this section to change PLL settings.' The TXSYSCLKSEL and RXSYSCLKSEL attributes also need to be modified either manually by editing `gtwizard_v2_1_gt.v` or using the Transceiver Wizard and replacing `gtwizard_v2_1_gt.v` after using the GUI to select the correct PLL and REFCLK options. Ensure the clock selected in `gtxwizard_v2_1_top.v` and in the Wizard match.

When using the 7 series Transceiver Wizard to change line rates, the choice of REFCLK input and transceiver XY position are not critical because these are overwritten by the JESD204 constraints, transceiver top level and block level.

AXI4-Lite Management Interface

For AXI information, go to www.xilinx.com/ipcenter/axi4.htm.

The JESD204 core netlist is configured by individual signals that can be tied off if required. For example, you may not need to have full software control over all parameters. See Core Interfaces for details of the configuration signals.

The Block Level wrapper includes a Xilinx® IPIF interface to convert from AXI4-Lite to a simple register bank. Registers are provided for all the configuration and status signals to and from the core. The AXI4-Lite address map is detailed in AXI4-Lite Configuration Address Map.

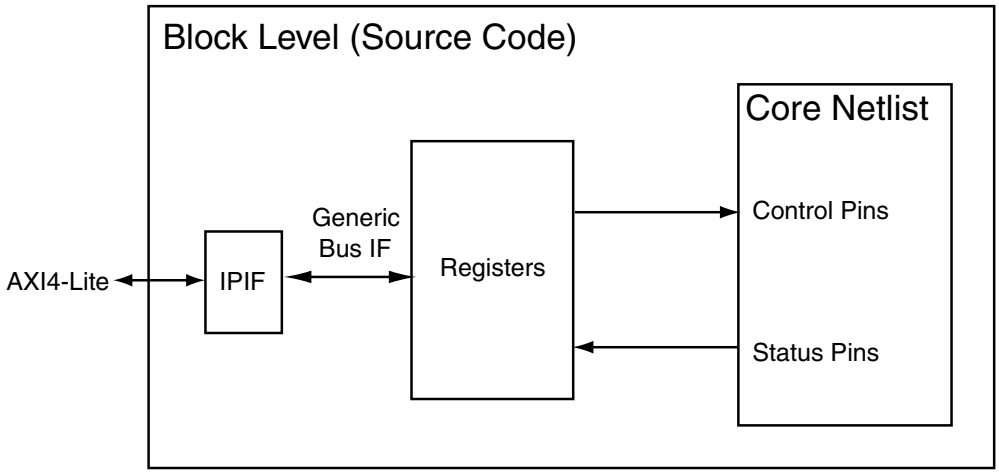

Figure 8-10: **AXI4-Lite - IPIF Interface**

Chapter 9

Quick Start Example Design

This chapter provides detailed information about the example designs and includes instructions for generating and customizing the core, the purpose and contents of the provided scripts, and the contents of the example Verilog wrappers.

Overview

The JESD204 Example Design includes the following:

- An instance of the JESD204 core, either Transmit or Receive depending on the option selected when generating the core
- The Verilog Block Level that contains the serial transceiver and Clocking Logic required by the core
- A Verilog Example top-level design that contains device-level I/O
- A demonstration test bench to exercise the core

Figure 9-1 illustrates the JESD204 example design, which principally connects the core I/O to the top-level FPGA I/O to allow the core to be implemented as an FPGA design. The target part is 7k325tffg900 if a Kintex™-7 FPGA is selected in the CORE Generator™ tool and 6vlx75tff784 if a Virtex®-6 FPGA is selected.

The top-level example design is intended as a template for creating your own top-level FPGA design. The Block Level is designed to be instantiated in your own design. The example design is not intended to be targeted to a development board but is a generic example.

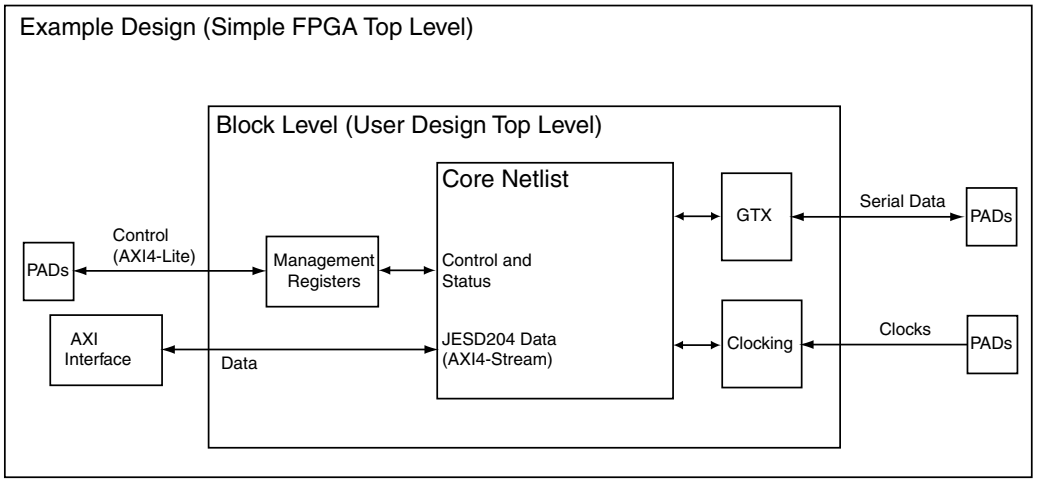

Figure 9-1: **JESD204 Example Design**

Generating the Core

This section describes how to generate a JESD204 core using the Xilinx® CORE Generator tool. The generated core uses the default values.

1. Start the CORE Generator tool.
   For help starting and using the CORE Generator tool, see the documentation supplied with the ISE® Design Suite.
2. Choose **File > New Project**.
3. Type a directory name for the project. In this example, the directory is called `<project_dir>`.
4. Do the following to set project options:
   From the Part tab, select a silicon family, part, speed grade, and package that supports the JESD204 core.

   Note: If an unsupported silicon family is selected, the JESD204 core does not appear in the taxonomy tree. For a list of supported architectures, see the JESD204 Product Specification.
5. After creating the project, locate the core in the taxonomy tree at the left side of the CORE Generator window. The JESD204 core appears under these categories:
   - Communications & Networking/Telecommunications
   - Communications & Networking/Wireless
6. Double-click the core to open it. A message can appear warning you about the limitations of the license, and then the JESD204 customization screen appears.
7. Accept all other default options.
8. Click Finish to generate the core.

The core and its supporting files, including the example design, are generated in the project directory. For a detailed description of the directory structure and files, see Chapter 10, Detailed Example Design.

Simulating the JESD204 Example Design

The example design supplied with the JESD204 core provides a complete environment that allows you to simulate the core and view the outputs using Mentor Graphics ModelSim. (For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.) Scripts are provided for pre-layout, functional simulation only.

Setting up for Simulation

To run the functional simulation, you must have the Xilinx Simulation Libraries compiled for your system. See COMPXLIB in the *Xilinx ISE Command Line Tools User Guide* [Ref 7], and the *Xilinx ISE Software Manuals and Help* [Ref 8].

The Xilinx simulation libraries must be mapped into the simulator. If the libraries are not set for your environment, go to Answer Record 15338 on www.xilinx.com/support for assistance compiling Xilinx simulation models and setting up the simulator environment. Virtex-6 FPGA designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant, mixed language simulator. For a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator, Mentor Graphics ModelSim v6.6d is currently supported.

Pre-implementation Simulation

To run a functional simulation of the example design:

1. Open a command prompt or shell in your project directory and then set the current directory to: <project_dir>/JESD204_v2_1/simulation/functional.
2. Launch the simulation script: vsim -do simulate_mti.do.

The simulation script compiles the functional model and the demonstration test bench, adds some relevant signals to a wave window, and then runs the simulation to completion. You can then inspect the simulation transcript and waveform to observe the operation of the core. See Chapter 10, Detailed Example Design for detailed information about the simulation environment.

Implementing the JESD204 Example Designs

After successfully generating the core, the core and example design Verilog wrapper can be processed using the Xilinx implementation tools. The generated outputs include several scripts to assist in processing.

Open a command prompt or shell in your project directory and enter these commands:

Linux

% **cd** <component_name>**/implement**

% **./implement.sh**

Windows

> **cd** <component_name>**\implement**

> **implement.bat**

The implement command performs these steps:

1. Starts a script to synthesize the JESD204 example design Verilog
2. Builds, maps, and place-and-routes the example design
3. Generates an FPGA bit file
4. Generates a gate-level Verilog model for timing simulation

All output files are located in the <results> subdirectory, which is automatically created by the script.

XILINX®

Chapter 10

Detailed Example Design

This chapter describes the directory structure and the types of files delivered when the JESD204 core is generated. It also contains information about the provided demonstration test bench.

- <project directory>
  Top-level core files: netlist and instantiation templates
  - <project directory>/<component name>
    - <component name>/example_design
      Block level wrapper, transceiver wrapper and example design files
    - <component name>/doc
      Product documentation
    - <component name>/implement
      Implementation scripts
      - implement/results
        Results directory, created after implementation scripts are run, and contains implement script results
    - <component name>/simulation
      Test Bench
      - /simulation/functional
        Scripts for functional simulation

<project directory>

The <project directory> contains all the Xilinx® CORE Generator™ project files.

Table 10-1: **Project Directory**

Name	Description
<component_name>.ngc	A binary Xilinx implementation netlist. Used as an input to the Xilinx implementation tools.
<component_name>.v	Verilog structural simulation model. File used to support functional simulation of a core.
<component_name>.xco	CORE Generator project-specific option file; can be used as an input to the CORE Generator tool.
<component_name>_flist.txt	List of files delivered with the core.
<component_name>.veo	A Verilog instantiation template for the core netlist.

<component name>/example_design

The example design directory contains the example design files provided with the core.

Table 10-2: **Example Design Directory**

Name	Description
<component_name>_block.vhd	JESD204 core combined with transceiver and clocking blocks. This file is intended to be instantiated in user designs directly.
<component_name>_example_design.v	Simple device-level design intended to be used for running the example test bench and synthesis scripts only. This is not intended to be used in user designs.
<component_name>_example_design.ucf	Example design constraints file. The constraints for the core can be extracted from this file and included in a design using the core or block level.
<component_name>_address_decoder.v	AXI4-Lite to IPIF logic used for configuration interface.
<component_name>_axi_lite_ipif.v	AXI4-Lite to IPIF logic used for configuration interface.
<component_name>_counter_f.v	AXI4-Lite to IPIF logic used for configuration interface.
<component_name>_pselect_f.v	AXI4-Lite to IPIF logic used for configuration interface.
<component_name>_slave_attachment.v	AXI4-Lite to IPIF logic used for configuration interface.
<component_name>_v6_gtxwizard_v1_12_gtx.v	Virtex®-6 FPGA GTX transceiver wrapper logic.

Table 10-2: **Example Design Directory *(Cont'd)***

Name	Description
<component_name>_v6_gtxwizard_v1_12_top.v	Virtex-6 FPGA GTX transceiver wrapper logic top level.
<component_name>_v6_gtxwizard_v1_12.v	Virtex-6 FPGA GTX transceiver wrapper logic.
<component_name>_gtwizard_v2_1_gt.v	Kintex™-7 FPGA GTX transceiver wrapper logic.
<component_name>_gtwizard_v2_1_top.v	Kintex-7 FPGA GTX transceiver wrapper logic.
<component_name>_gtwizard_v2_1.v	Kintex-7 FPGA GTX transceiver wrapper logic.
<component_name>_mod.v	Dummy module top level used by synthesis to allow instantiation of core netlist.

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 10-3: **Doc Directory**

Name	Description
jesd204_readme.txt	JESD204 core release notes file
jesd204_v2_1_vinfo.html	JESD204 core release notes file in html format

<component name>/implement

The implement directory contains the core implementation script files.

Table 10-4: **Implement Directory**

Name	Description
implement.sh	Linux shell script for JESD204 example design implementation
implement.bat	Windows batch file for JESD204 example design implementation
xst.prj	XST Project file for the JESD204 example design
xst.scr	XST script for the JESD204 example design

implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 10-5: **Results Directory**

Name	Description
Implement script result files	

<component name>/simulation

The simulation directory contains the simulation scripts provided with the core.

Table 10-6: **Simulation Directory**

Name	Description
demo_tb.v	General-use top-level wrapper for example design, AXI and serial JESD204 data generators.

/simulation/functional

This directory contains functional simulation scripts provided with the core.

Table 10-7: **Functional Directory**

Name	Description
simulate_mti.do	Simulation script
wave_mti.do	Simulation support script

XILINX®

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm

Additional Core Resources

For detailed information and updates about the JESD204 core, see the JESD204 Data Sheet, located on the JESD204 product page.

After generating the core, these documents are available in the document directory:

- JESD204 Release Notes
- JESD204 User Guide

References

1. AMBA® AXI4-Stream Protocol Specification
2. 7 Series FPGAs GTX Transceivers User Guide (UG476)
3. Virtex®-6 FPGA GTX Transceivers User Guide (UG366)
4. Serial Interface for Data Converters (JESD204A)
5. Serial Interface for Data Converters (JESD204B)
6. 7 Series data sheets
7. Xilinx ISE Command Line Tools User Guide (UG628)
8. Xilinx ISE® Manuals and Help
9. Xilinx AXI Reference Guide (UG761)
10. Xilinx 7 Series FPGAs Configurable Logic Block User Guide (UG474)

To search for Xilinx documentation, go to www.xilinx.com/support.

To search for JESD204 documentation, go to www.jedec.org