LogiCORE IP SPDIF v1.2

Product Guide

PG045 December 18, 2012

XILINX®

XILINX

Table of Contents

Appendix B: Migrating

Appendix C: Debugging

Appendix D: Additional Resources

SECTION I: SUMMARY

Introduction

The Sony/Philips Digital Interconnect Format (SPDIF) core is a digital audio interface controller that implements the International Electronic Commission (IEC) 60958-3 interface for transmitting and receiving audio data.

This includes standard bus interfaces to the AMBA® AXI4-Lite and AXI4-Stream interfaces, allowing for integration to the IP core with a master system for further processing of audio data. Data collected by the LogiCORE™ IP SPDIF core is stored in the core internal FIFO, allowing the system to process a relatively slow audio stream.

Features

- Configurable as an SPDIF audio data transmitter or an SPDIF audio data receiver
- Configurable FIFO buffer stores the audio sample data

SPDIF Interface

- IEC 60958-3 standard SPDIF digital audio bus interface
- Two audio channels
- Audio sample lengths of 16/20/24 bits
- Data recovery from the bi-phase mark encoded SPDIF data when the IP core is in receive mode
- Variable sampling rates (32/44.1/48/88.2/96/176.4/192 kHz)
- SPDIF transmitter sends the invalid null audio frames over the SPDIF line in case of a FIFO under-run condition

AXI4-Stream Interface

- Based on AXI4-Stream specification
- Master/slave on AXI4 streaming interface
- 32-bit data width support
- Continuous aligned streams only (no null or positional bytes transmission support)

AXI4-Lite Interface

- Register access support through the AXI4-Lite interface
- 32-bit data width support

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq™-7000[2], Virtex®-7, Kintex™-7, Artix™-7, Virtex-6, Spartan®-6
Supported User Interfaces	SPDIF, AXI4-Stream, AXI4-Lite
Resources	See Table 2-1 and Table 2-2
Provided with Core	
Design Files	NGC Netlist
Example Design	ISE®: Verilog, VHDL Vivado™: Verilog, VHDL
Test Bench	Verilog
Constraints File	ISE: User Constraints File (UCF) Vivado: XDC
Simulation Model	Verilog and VHDL Structural Models
Supported S/W Driver	Not Provided
Tested Design Flows[3]	
Design Entry	ISE Design Suite 14.4 Vivado Design Suite 2012.4[4]
Simulation	Mentor Graphics ModelSim
Synthesis	Xilinx Synthesis Technology (XST) Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete listing of supported devices, see the release notes for this core.
2. Supported in ISE Design Suite implementations only.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
4. Supports only 7 series devices.

Overview

Functional Description

The LogiCORE™ IP SPDIF core is compatible with the SPDIF protocol. It can be used in a receive or transmit mode and delivers or accepts audio data from an AXI4-Stream input. The SPDIF core is designed for use in audio systems, and is used with the LogiCORE IP DisplayPort core for audio data transfers. Figure 1-1 shows the SPDIF block diagram.

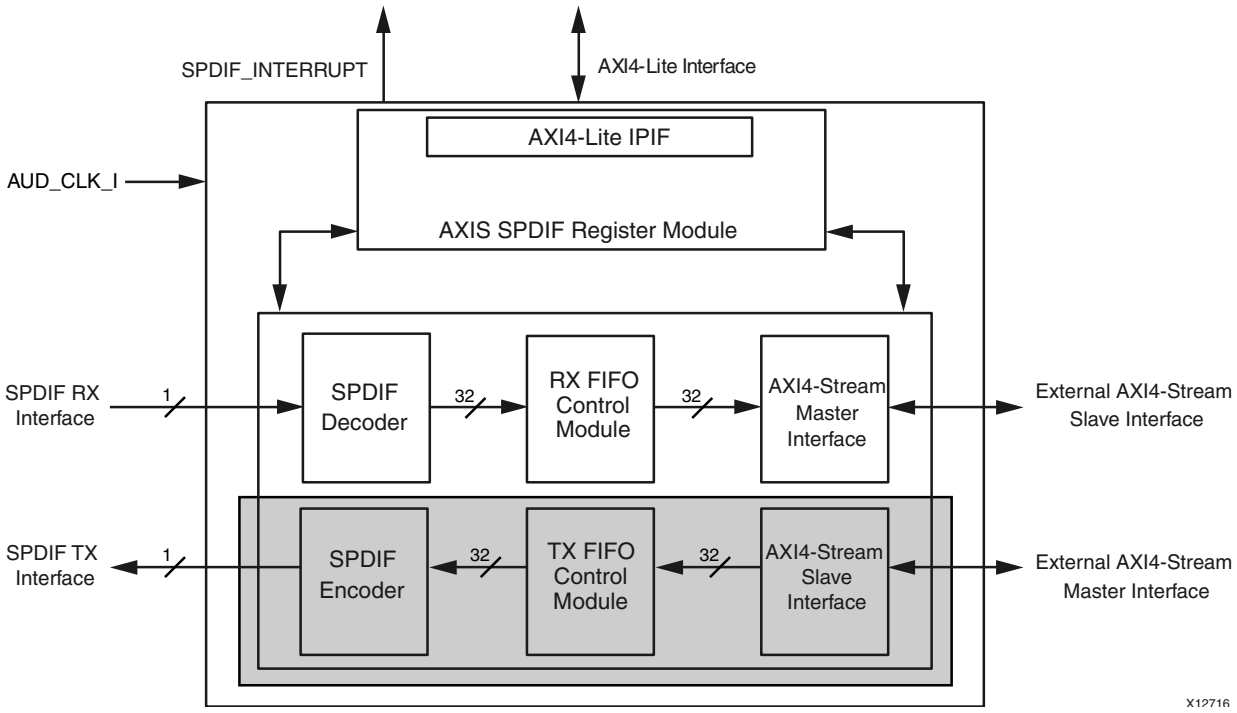

Figure 1-1: **SPDIF Block Diagram**

The SPDIF core can operate in two modes:

- SPDIF receiver – Receives SPDIF audio data and sends it through the AXI4-Stream interface
- SPDIF transmitter – Receives audio data through the AXI4-Stream interface and transmits it through the SPDIF interface

The core contains the following blocks in SPDIF receive mode:

- AXI4-Lite IPIF
- AXI SPDIF Register Module
- SPDIF Decoder
- RX FIFO Control Module
- AXI4-Stream Master Interface

The core contains the following blocks in SPDIF transmit mode:

- AXI4-Lite IPIF
- AXI SPDIF Register Module
- AXI4-Stream Slave Interface
- TX FIFO Control Module
- SPDIF Encoder

Feature Summary

The LogiCORE IP SPDIF core is a digital audio interface controller that implements the PCM IEC 60958-3 interface features for transmitting and receiving audio data. The core can be configured as an SPDIF audio data transmitter or an SPDIF audio data receiver. The IEC 60958-3 standard SPDIF digital audio bus interface has two audio channels and audio sample lengths of 16, 20, and 24 bits. Sample rates range from 32 kHz to 192 kHz.

The core includes an AMBA® AXI4-Lite interface for register access and an AXI4-Stream interface for audio data transfers. The AXI4-Stream interface allows integration between the IP core and an AXI system for further processing of audio data. Data collected by the SPDIF core is stored in the core's internal FIFO, allowing the system to process a relatively slow audio stream.

Unsupported Features

The SPDIF core does not support:

- Non-linear PCM encoded audio data streams
- AXI4-Lite and AXI4-stream bus widths other than 32 bits

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite and ISE Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the DisplayPort web page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

The LogiCORE™ IP SPDIF core implements IEC 60958-3 interface features for transmitting and receiving audio data.

Resource Utilization

Resources required for the SPDIF core have been estimated in Transmit mode (Table 2-1) and Receive mode (Table 2-2). These values were generated in an ISE Design Suite implementation using the Xilinx CORE Generator™ tools. They are derived from post-synthesis reports, and may change during MAP and PAR. The device is XC6SLX150T-FGG676-3.

Table 2-1: **SPDIF Resource Utilization in Transmit Mode**

C_AXIS_BUFFER_SIZE	LUTs	FFs	Block RAMs
16	212	212	1
512	267	287	1

Table 2-2: **SPDIF Resource Utilization in Receive Mode**

C_AXIS_BUFFER_SIZE	C_STATUS_REG	C_USERDATA_REG	LUTs	FFs	Block RAMs
16	0	0	372	346	1
512	1	1	1506	1646	1

Port Descriptions

This section details the interfaces on the SPDIF core. Table 2-3 defines the SPDIF I/O signals.

Table 2-3: **I/O Signal Description**

Port	Signal Name	Interface	I/O	Initial State	Description
System Signals					
P1	AUD_CLK_I	System	I	–	Audio clock input used at the SPDIF interface
P2	SPDIF_INTERRUPT	System	O	0	SPDIF core interrupt output. When the interrupt occurs, this signal is continuously 1 until cleared/disabled.
AXI4-Lite Interface System Signals					
P3	S_AXI_ACLK	System	I	–	AXI4-Lite clock
P4	S_AXI_ARESETN	System	I	–	AXI4-Lite reset, active-Low
AXI4-Lite Write Address Channel Signals					
P5	S_AXI_AWADDR[C_S_AXI_ADDR_WIDTH-1:0]	AXI4-Lite	I	–	AXI4-Lite Write address. The write address bus gives the address of the first transfer in a write burst transaction.
P6	S_AXI_AWVALID	AXI4-Lite	I	–	Write address valid. This signal indicates that valid write address and control information are available.
P7	S_AXI_AWREADY	AXI4-Lite	O	0	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
AXI4-Lite Write Data Channel Signals					
P8	S_AXI_WDATA[C_S_AXI_DATA_WIDTH-1:0]	AXI4-Lite	I	–	Write data bus
P9	S_AXI_WSTRB[C_S_AXI_DATA_WIDTH/8-1:0]	AXI4-Lite	I	–	Write strobes. Each signal indicates which byte lanes to update in memory. These are unused in the SPDIF core.
P10	S_AXI_WVALID	AXI4-Lite	I	–	Write valid. This signal indicates that valid write data and strobes are available.
P11	S_AXI_WREADY	AXI4-Lite	O	0	Write ready. This signal indicates that the slave can accept the write data.
AXI4-Lite Write Response Channel Signals					
P12	S_AXI_BRESP[1:0]	AXI4-Lite	O	0	Write response. This signal indicates the status of the write transaction.
P13	S_AXI_BVALID	AXI4-Lite	O	0	Write response valid. This signal indicates that a valid write response is available.
P14	S_AXI_BREADY	AXI4-Lite	I	–	Response ready. This signal indicates that the master can accept the response information.

Table 2-3: **I/O Signal Description *(Cont'd)***

Port	Signal Name	Interface	I/O	Initial State	Description
AXI4-Lite Read Address Channel Signals					
P15	S_AXI_ARADDR[C_S_AXI_ADDR_WIDTH -1:0]	AXI4-Lite	I	–	Read address. The read address bus gives the initial address of a read burst transaction.
P16	S_AXI_ARVALID	AXI4-Lite	I	–	Read address valid. When high, this signal indicates that the read address and control information are valid and remain stable until the address acknowledgment signal S_AXI_ARREADY is high.
P17	S_AXI_ARREADY	AXI4-Lite	O	0	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
AXI4-Lite Read Data Channel Signals					
P18	S_AXI_RDATA[C_S_AXI_DATA_WIDTH -1:0]	AXI4-Lite	O	0	Read data bus
P19	S_AXI_RRESP[1:0]	AXI4-Lite	O	0	Read response. This signal indicates the status of the read transfer.
P20	S_AXI_RVALID	AXI4-Lite	O	0	Read valid. This signal indicates that the required read data is available and the read transfer can complete.
P21	S_AXI_RREADY	AXI4-Lite	I	–	Read ready. This signal indicates that the master can accept the read data and response information.
SPDIF RX Interface Signals					
P22	SPDIF_I	SPDIF	I	–	Audio input from the SPDIF interface
SPDIF TX Interface Signals					
P23	SPDIF_O	SPDIF	O	0	Audio output to the SPDIF interface
AXI4-Stream Master Interface Signals					
P24	M_AXIS_ACLK	AXI4-Stream	I	–	The AXI4-Stream global clock signal in receive mode. All streaming signals are sampled on the rising edge of M_AXIS_ACLK.
P25	M_AXIS_ARESETN	AXI4-Stream	I	–	The AXI4-Stream global reset signal in receive mode. M_AXIS_ARESETN is active-Low.
P26	M_AXIS_TVALID	AXI4-Stream	O	0	AXI4-Stream Valid Out. Indicates stream data bus, M_AXIS_TDATA, is valid. 0 = Write data is not valid 1 = Write data is valid
P27	M_AXIS_TREADY	AXI4-Stream	I	–	AXI4-Stream Ready. Indicates to the AXI4-Stream Master interface that the target is ready to receive stream data. 0 = Not ready to receive data 1 = Ready to receive data

Table 2-3: **I/O Signal Description *(Cont'd)***

Port	Signal Name	Interface	I/O	Initial State	Description
P28	M_AXIS_TDATA[C_AXIS_TDATA_WIDTH-1:0]	AXI4-Stream	O	0	AXI4-Stream Data Out
P29	M_AXIS_TID[C_AXIS_TID_WIDTH-1:0]	AXI4-Stream	O	0	M_AXIS_TID is the data stream identifier that indicates channel number of audio data.
AXI4-Stream Slave Interface Signals					
P30	S_AXIS_ACLK	AXI4-Stream	I	–	The AXI4-Stream global clock signal in transmit mode. All signals are sampled on the rising edge of S_AXIS_ACLK.
P31	S_AXIS_ARESETN	AXI4-Stream	I	–	The AXI4-Stream global reset signal in transmit mode. S_AXIS_ARESETN is active-Low.
P32	S_AXIS_TVALID	AXI4-Stream	I	–	AXI4-Stream Valid In. Indicates the stream data bus, S_AXIS_TDATA, is valid. 0 = Write data is not valid 1 = Write data is valid
P33	S_AXIS_TREADY	AXI4-Stream	O	0	AXI4-Stream Ready. Indicates the AXI4-Stream Slave interface is ready to receive stream data. 0 = Not ready to receive data 1 = Ready to receive data
P34	S_AXIS_TDATA[C_AXIS_TDATA_WIDTH-1:0]	AXI4-Stream	I	–	AXI4-Stream Data In
P35	S_AXIS_TID[C_AXIS_TID_WIDTH-1:0]	AXI4-Stream	I	–	S_AXIS_TID is the data stream identifier that indicates channel number of audio data.

Register Space

Table 2-4 specifies the offset address, register name, and accessibility of each firmware addressable register from the three classes of registers within the SPDIF core. User access to each register is from an offset to the base address set the C_BASEADDR parameter. For example, C_BASEADDR + 0x44 represents the address of the Control Register.

For Channel Status registers, capturing channel status bits into channel status registers is configurable. When the C_CSTATUS_REG parameter is set to 1, only these registers are part of SPDIF receiver logic. Channel status registers hold the 192-bit channel status information received over the SPDIF input when the SPDIF core is in receive mode (C_TRANSMIT_RECEIVE is 0). The channel status is assumed to be common for both channel a and channel b. Thus the channels status bits are captured from one of the channels. These registers are updated after one complete audio frame is received. Usually, the channel status register data does not change frame to frame. For more information on these bits including their descriptions, see the IEC-60958-3 specification.

For the Channel a/b User Data registers, capturing SPDIF user data bits into user data registers is configurable. When the C_USERDATA_REG parameter is set to 1, only these registers are part of SPDIF receiver logic. User data registers hold the 192-bit user data received over the SPDIF input when the SPDIF core is in receive mode (C_TRANSMIT_RECEIVE is 0). The user data is captured for both channel a and channel b in the corresponding registers. These registers are updated after one complete audio frame is received.

Table 2-4: **Register Map**

Offset Address	Register Name	Access Type	Default Value	Description
Interrupt Registers				
0x1C	Global Interrupt Enable (GIE)(1)	Read/Write	0x0000	Device Global interrupt enable register
0x20	Interrupt Status Register (ISR)(1)	Read/Toggle on Writing 1(1)	0x0000	IP interrupt status register
0x28	Interrupt Enable Register (IER)(1)	Read/Write	0x0000	IP interrupt enable register
Soft Reset Register				
0x40	Soft Reset Register(2)	Write	N/A	Soft Reset Register
SPDIF Configuration, Control, and Data Registers				
0x44	SPDIF Control Register	Read/Write	0x0000	Control register
0x48	SPDIF Status Register	Read	0x0000	Status register
0x4C	Channel Status Register0	Read	0x0000	Audio Channel status bits 0 to 31
0x50	Channel Status Register1	Read	0x0000	Audio Channel status bits 32 to 63
0x54	Channel Status Register2	Read	0x0000	Audio Channel status bits 64 to 95
0x58	Channel Status Register3	Read	0x0000	Audio Channel status bits 96 to 127
0x5C	Channel Status Register4	Read	0x0000	Audio Channel status bits 128 to 159
0x60	Channel Status Register5	Read	0x0000	Audio Channel status bits 160 to 191
0x64	Channela User Data Register0	Read	0x0000	Channela User Data bits 0 to 31
0x68	Channela User Data Register1	Read	0x0000	Channela User Data bits 32 to 63
0x6C	Channela User Data Register2	Read	0x0000	Channela User Data bits 64 to 95
0x70	Channela User Data Register3	Read	0x0000	Channela User Data bits 96 to 127
0x74	Channela User Data Register4	Read	0x0000	Channela User Data bits 128 to 159
0x78	Channela User Data Register5	Read	0x0000	Channela User Data bits 160 to 191
0x7C	Channelb User Data Register0	Read	0x0000	Channelb User Data bits 0 to 31
0x80	Channelb User Data Register1	Read	0x0000	Channelb User Data bits 32 to 63
0x84	Channelb User Data Register2	Read	0x0000	Channelb User Data bits 64 to 95
0x88	Channelb User Data Register3	Read	0x0000	Channelb User Data bits 96 to 127
0x8C	Channelb User Data Register4	Read	0x0000	Channelb User Data bits 128 to 159

Table 2-4: **Register Map *(Cont'd)***

Offset Address	Register Name	Access Type	Default Value	Description
0x90	Channelb User Data Register5	Read	0x0000	Channelb User Data bits 160 to 191

Notes:

1. See the *Xilinx Interrupt Control Data Sheet* [Ref 5].
2. The soft reset functionality is implemented by the soft_reset module.

Global Interrupt Enable (GIE)

The Global Interrupt Enable register, described in Table 2-5, has a single defined bit that globally enables the final interrupt out to the system.

Table 2-5: **Global Interrupt Enable Register (Offset 0x1C)**

Bits	Name	Core Access	Reset Value	Description
31	GIE	Read/Write	0	Global Interrupt Enable 0 = All interrupts disabled. No interrupts from SPDIF 1 = Unmasked SPDIF interrupts are passed to the processor
30:0	Unused	N/A	N/A	Reserved

Interrupt Status Register (ISR)

Firmware uses the ISR to determine which interrupt events from the SPDIF core need servicing. Writing a 1 to a bit position within the register causes the corresponding bit to toggle. All register bits are cleared upon reset. The register uses a toggle on write method to allow the firmware to clear selected interrupts by writing a 1 to the desired interrupt bit field position.

This mechanism avoids the requirement on the User Interrupt Service routine to perform a Read/Modify/Write operation to clear a single bit within the register. An interrupt value of 1 indicates the interrupt has occurred. A value of 0 indicates that no interrupt occurred or it was cleared. The Interrupt Status register bit fields are described in Table 2-6.

Table 2-6: **Interrupt Status Register (Offset 0x20)**

Bits	Name	Core Access	Reset Value	Description
31:5	Unused	N/A	0	Reserved
4	Preamble Error	Read/Toggle on writing 1	0	This bit is set when the incorrect preamble format is received over the SPDIF core in receive mode, for example, if the channela preamble is received after the start of block.
3	BMC Error	Read/Toggle on writing 1	0	This bit is set when there is a bi-phase mark code (BMC) violation over the SPDIF audio data bits in receive mode (except for the preamble).

Table 2-6: **Interrupt Status Register (Offset 0x20) *(Cont'd)***

Bits	Name	Core Access	Reset Value	Description
2	Start Of Block	Read/Toggle on writing 1	0	This bit is set when the SPDIF core is in receive mode and when it detects the start of block preamble over the SPDIF_I input.
1	TX/RX FIFO Empty	Read/Toggle on writing 1	0	This bit is set when the TX FIFO changes from non-empty to empty in Transmit mode and when the RX FIFO changes from non-empty to empty.
0	TX/RX FIFO Full	Read/Toggle on writing 1	0	This bit is set when the TX FIFO becomes full in transmit mode and when the RX FIFO becomes full in receive mode.

Interrupt Enable Register

The Interrupt Enable register is a read and write register that enables the SPDIF interrupts. The Interrupt Enable register bit fields are described in Table 2-7.

Table 2-7: **Interrupt Enable Register (Offset 0x28)**

Bits	Name	Core Access	Reset Value	Description
31:5	Unused	N/A	N/A	Reserved
4	Preamble Error	Read/Write	0	This bit must be set to generate the preamble error interrupt. In transmit mode, this bit is unused.
3	BMC Error Interrupt Enable	Read/Write	0	This bit must be set to generate the BMC error interrupt. In transmit mode, this bit is unused.
2	Start Of Block Interrupt Enable	Read/Write	0	This bit must be set to generate the start of block interrupt in receive mode. In transmit mode, this bit is unused.
1	TX/RX FIFO Empty Interrupt Enable	Read/Write	0	This bit must be set to generate the TX FIFO empty interrupt when the SPDIF core is in transmit mode and the same bit must be set when the SPDIF core is in transmit mode to enable the RX FIFO empty interrupt.
0	TX/RX FIFO Full Interrupt Enable	Read/Write	0	This bit must be set to generate the TX FIFO full interrupt when the SPDIF core is in transmit mode and the same bit has to be set when the SPDIF core is in receive mode to enable the RX FIFO full interrupt.

Soft Reset Register

The firmware writes to the Soft Reset register to initialize all of the SPDIF registers to their default states. To accomplish this, the firmware must write the value of `0xA` to the least-significant nibble of the 32-bit word. After recognizing a write of `0xA`, the soft_reset module issues a pulse four clocks long to reset the SPDIF core. At the end of the pulse, the Soft Reset register acknowledges the AXI4 transaction, which prevents anything further

from happening while the reset occurs. Writing any value to Bits[3:0] other than 0xA results in an AXI4 transaction acknowledge with an error status. This register is not readable. The Soft Reset register bit fields are described in Table 2-8.

Table 2-8: **Soft Reset Register (Offset 0x40)**

Bits	Name	Core Access	Reset Value	Description
31:4	Unused	N/A	N/A	Reserved
3:0	Reset Key	Write	0	The firmware must write a value of 0xA to this field to cause a soft reset of the Interrupt registers of the SPDIF controller. Writing any other value results in an AXI4 transaction acknowledgment with SLVERR and no reset occurs.

SPDIF Control Register

The SPDIF Control register is read and write register that configures the SPDIF core. This register has an SPDIF enable bit, a TX/RX FIFO flush bit, and clock configuration bits. The SPDIF Control register bit fields are described in Table 2-9.

Table 2-9: **SPDIF Control Register (Offset 0x44)**

Bits	Name	Core Access	Reset Value	Description
31:6	Unused	N/A	N/A	Reserved
5:2	TX clock configuration bits	Read/Write	0	These bits give the audio clock division number to transmit the SPDIF bits. The bit frequency is generated based on these bits. For example, to generate a 32 kHz audio sampling frequency, the bit rate is 2.048 MHz (that is, 32 kHz × 64 because the audio sample width is of 64 bits, 32 bits for channel a and 32 bits for channel b). If the supplied AUD_CLK_I is 16.384, the Bits Division Number has to be 0001. Bits Division Number: 0000 = 4 0001 = 8 0010 = 16 0011 = 24 0100 = 32 0101 = 48 0110 = 64 Others = Reserved
1	SPDIF TX FIFO/RX FIFO Flush	Read/Write	0	This bit has to be set to 1 to reset the TX FIFO in transmit mode and to reset the RX FIFO in receive mode.
0	SPDIF TX/RX Enable	Read/Write	0	This bit has must be set to 1 to enable the SPDIF core.

SPDIF Status Register

The SPDIF Status register is a read-only register that contains the status of the SPDIF core. The SPDIF Status register bit fields are described in Table 2-10.

Table 2-10: **SPDIF Status Register (Offset 0x48)**

Bits	Name	Core Access	Reset Value	Description
31:10	Unused	N/A	N/A	Reserved
9:0	Sample clock count	Read	0	These bits are updated with the number of audio clocks for the SPDIF data bit period. This audio clock count is recovered by the SPDIF decoder module. This count gives the approximate count for the SPDIF bit period when the audio clock is not the harmonic of core frequency. These bits are used in receive mode only. In transmit mode, these bits are unused.

Channel Status Registers

A set of six configurable registers store the 192-bit SPDIF Audio Channel Status information. These registers are active when the SPDIF core is in receive mode and when the C_CSTATUS_REG parameter is 1. This channel status information is captured from one of the channels, assuming both channela and channelb carry the same channel status information over SPDIF. The Channel Status register bit fields are described in Table 2-11. For complete descriptions of these bit fields, see the IEC-60958-3 specification.

Table 2-11: **Channel Status Registers (Offsets 0x4C to 0x60)**

Bits	Name	Core Access	Reset Value	Description
31:0	Channel Status register0	Read	0	This register holds bits 0 to 31 of the audio channel status information received over SPDIF.
31:0	Channel Status register1	Read	0	This register holds bits 32 to 63 of the audio channel status information received over SPDIF.
31:0	Channel Status register2	Read	0	This register holds bits 64 to 95 of the audio channel status information received over SPDIF.
31:0	Channel Status register3	Read	0	This register holds bits 96 to 127 of the audio channel status information received over SPDIF.
31:0	Channel Status register4	Read	0	This register holds bits 128 to 159 of the audio channel status information received over SPDIF.
31:0	Channel Status register5	Read	0	This register holds bits 160 to 191 of the audio channel status information received over SPDIF

Channela User Data Registers

A set of six configurable registers store the 192-bit SPDIF Channela User data information. These registers are active when the SPDIF core is in receive mode and when the

C_USERDATA_REG parameter is 1. This user data information is captured from channel a. The Channela User Data registers bit fields are described in Table 2-12.

Table 2-12: **Channela User Data Registers (Offsets 0x64-0x78)**

Bits	Name	Core Access	Reset Value	Description
31:0	Channela User Data Register0	Read	0	This register holds bits 0 to 31 of the user data information received over SPDIF channela.
31:0	Channela User Data Register1	Read	0	This register holds bits 32 to 63 of the audio user data information received over SPDIF channela.
31:0	Channela User Data Register2	Read	0	This register holds bits 64 to 95 of the user data information received over SPDIF channela.
31:0	Channela User Data Register3	Read	0	This register holds bits 96 to 127 of the user data information received over SPDIF channela.
31:0	Channela User Data Register4	Read	0	This register holds bits 128 to 159 of the user data information received over SPDIF channela.
31:0	Channela User Data Register5	Read	0	This register holds bits 160 to 191 of the user data information received over SPDIF channela.

Channelb User Data Registers

A set of six configurable registers store the 192-bit SPDIF Channelb User data information. These registers are active when the SPDIF core is in receive mode and when the C_USERDATA_REG parameter is 1. This user data information is captured from channel b. The Channelb User data registers bit fields are described in Table 2-13.

Table 2-13: **Channelb User Data Registers (Offsets 0x7C to 0x90)**

Bits	Name	Core Access	Reset Value	Description
31:0	Channelb User data register0	Read	0	This register holds bits 0 to 31 of the user data information received over SPDIF channelb.
31:0	Channelb User data register1	Read	0	This register holds bits 32 to 63 of the audio user data information received over SPDIF channelb.
31:0	Channelb User data register2	Read	0	This register holds bits 64 to 95 of the user data information received over SPDIF channelb.
31:0	Channelb User data register3	Read	0	This register holds bits 96 to 127 of the user data information received over SPDIF channelb.
31:0	Channelb User data register4	Read	0	This register holds bits 128 to 159 of the user data information received over SPDIF channelb.
31:0	Channelb User data register5	Read	0	This register holds bits 160 to 191 of the user data information received over SPDIF channelb.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

SPDIF Register Module

This section describes the block within the SPDIF register module.

AXI4-Lite IPIF

The SPDIF register module provides the read/write control logic for the SPDIF core register set. The registers are accessible by the AXI4-Lite master interface through the AXI4-Lite IPIF module, which is part of the SPDIF register module.

The data width of the AXI4-Lite interface is fixed at 32 bits. The registers are defined in Table 2-4, page 14.

The interrupt control and soft reset functionality are also implemented as part of the SPDIF register module. The SPDIF core can be reset by writing `0xA` to the soft reset register. For an SPDIF transmitter, an SPDIF interrupt can be generated based on the FIFO Full/FIFO empty conditions. For an SPDIF receiver, in addition to the FIFO Full/FIFO Empty conditions, the interrupt is triggered if any preamble error/bi-phase mark code (BMC) error is detected over the SPDIF line or if the SPDIF core receives the start of block over the SPDIF line.

SPDIF Decoder

The enable bit in the Control register of the SPDIF register module has to be set to enable the SPDIF decoder module. The SPDIF decoder recovers data from the bi-phase mark coded SPDIF data stream (see Bi-Phase Mark Code). The audio clock frequency should be at least eight times the bit rate 64 x FS. FS is the sampling frequency where each sample has 64 bits. In bi-phase mark code, each bit changes twice in a bit period.

For example, to recover data from a 192 kHz sampling rate, the minimum audio clock frequency should be 8 × 64 × 192 kHz = 98.304 MHz to recover the data samples. As per

SPDIF Protocol Preamble violates the bi-phase mark code format, SPDIF decoder module identifies the channel number and the start of the audio block from the Preamble pattern.

The serial-to-parallel data conversion also takes place in the SPDIF Decoder module and then generates the FIFO write enables with the 32-bit FIFO input data. The sampling frequency information (that is, the count of audio clocks during the bit period) is updated in the Status register of the SPDIF register module.

This module detects the BMC/Preamble errors over the SPDIF line and reports to the SPDIF register module.

RX FIFO Control Module

The Asynchronous RX FIFO is used to store the 32-bit audio data received from the SPDIF decoder. The FIFO size is configurable and based on the C_AXIS_BUFFER_SIZE parameter generated. The data width of the FIFO is fixed to 32 bits. This module receives the FIFO write input control and write data from the SPDIF decoder. When the FIFO reaches the full condition, an interrupt is generated and the RX FIFO Full status is updated through the Status register of the SPDIF register module. Similarly, the RX FIFO empty interrupt and corresponding status are updated through the SPDIF Interrupt Status register.

AXI4-Stream Interface

The data width over the AXI4-Stream interface is fixed at 32 bits. Table 3-1 shows the 32-bit data format over the AXI4-Stream interface during audio data transmission and reception. All bit positions are as per the IEC60958-3 standard except for the preamble bit format.

Table 3-1: **AXI4-Stream Audio Data Format**

Bit[31]	Bit[30]	Bit[29]	Bit[28]	Bits[27:4]	Bits[3:0]
Parity	Channel Status Bit	User Data Bit	Validity Bit	Audio Sample Data	Preamble

The preamble provides the start of the audio block and audio channel information. The preamble patterns for the start of block, channel1 audio data, and channel2 audio data are listed in Table 3-2.

Table 3-2: **Preamble**

Bits[3:0]	Description
0001	Start of audio block
0010	Channel1 audio data
0011	Channel2 audio data

Bits[27:4] carry the audio data MSB bit at the 27th position and the LSB position is based on the audio sample length. Bit[28] provides the audio validity information. Bit[29] carries the user data information, and Bit[30] carries the channel status bit. Bit[31] is the even parity over 32 bits except for the preamble bits.

AXI4-Stream Master Interface

The AXI4-Stream Master interface transfers the 32-bit parallel data read from the Non-empty FIFO to the AXI4-Stream interface. The corresponding data valid signal (M_AXIS_TVALID) is set and the channel identifier signal (M_AXIS_TID) is driven with the corresponding channel number. The channel number information is available to the AXI4-Stream Master interface through the SPDIF decoder. This module depends on the handshaking signal M_AXIS_TREADY issued from the AXI4-Stream interface target slave for completion of the transfer.

AXI4-Stream Slave Interface

The AXI4-Stream Slave interface receives the 32-bit streaming data from the target connected to the AXI4-Stream interface. This module generates the handshaking signal S_AXIS_TREADY after receiving the streaming data (S_AXIS_TDATA), data valid signal (S_AXIS_TVALID), and channel number identification (S_AXIS_TID). This also generates the TX FIFO write control signals and transfers the data received from the AXI4-Stream interface to TX FIFO Control Module. If the SPDIF TX FIFO is full, this module stops receiving the audio samples by driving the handshake signal S_AXIS_TREADY Low. This avoids the FIFO overrun condition in the SPDIF transmitter.

TX FIFO Control Module

The Asynchronous TX FIFO is used to store the 32-bit streaming data received from the AXI4-Stream slave interface. The FIFO size is configurable and based on the configuration parameter C_AXIS_BUFFER_SIZE generated. The data width of the FIFO is fixed at 32 bits.

This module receives the FIFO write input control and write data from AXI4-Stream slave interface. When the FIFO reaches the full condition, an interrupt is generated and the TX FIFO Full status is updated through the Interrupt Status register of the SPDIF register module. This condition can occur when the SPDIF enable bit is not set in the Control register of the SPDIF register module, and the target is sending continuous streaming data or when the value set for the C_AXIS_BUFFER_SIZE parameter is not sufficient.

The TX FIFO Control Module generates an TX FIFO_EMPTY interrupt when the TX FIFO becomes empty. This condition can occur if enough samples are not received by the SPDIF transmitter to send over the SPDIF line.

SPDIF Encoder

This module has to get the clock configuration bits through the Control register to know the bit rate before SPDIF data transmission starts. The enable bit in Control register of the SPDIF registers module has to be set to enable the SPDIF encoder module. The SPDIF encoder converts the 32-bit parallel data received from the TX FIFO to serial data. The serial data is transferred over the SPDIF interface in Bi-Phase Mark Code format with respect to the received bit rate information. The audio clock input has to be the harmonic of the sampling rate and should be higher than the bit rate. For example, for a 192 kHz sampling rate, 49.152 MHz or 98.3 MHz has to be provided as the core frequency (`AUD_CLK_I`) and the corresponding clock divisor information has to be given through the clock configuration register bits).

In case of a TX FIFO under-run condition, this module sends the null audio frames over the SPDIF line with the validity bit set. When the validity bit is set to 1, it means per the SPDIF protocol that the audio sample is invalid and the codec has to ignore the sample.

IMPORTANT: *The audio clock generation and the setting of clock configuration bits in the Control register have to be done with care to support these sampling rates: 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, 176.4 kHz, and 192 kHz. The minimum audio clock frequency of 49.152 MHz or harmonic frequency of the same which is higher than this frequency supports the sampling rates of 32 kHz, 48 kHz, 96 kHz, and 192 kHz. The minimum audio clock frequency of 45.1584 MHz or harmonic frequency of the same which is higher than this frequency supports the sampling rates of 44.1 kHz, 88.2 kHz, and 176.4 kHz.*

Bi-Phase Mark Code

The SPDIF interface (IEC-60958) is a consumer version of the AES/EBU-interface. The SPDIF digital signal is coded using the bi-phase mark code (BMC), which is a type of phase modulation. The bit clock, data bits, and BMC signals are shown in Figure 3-1.

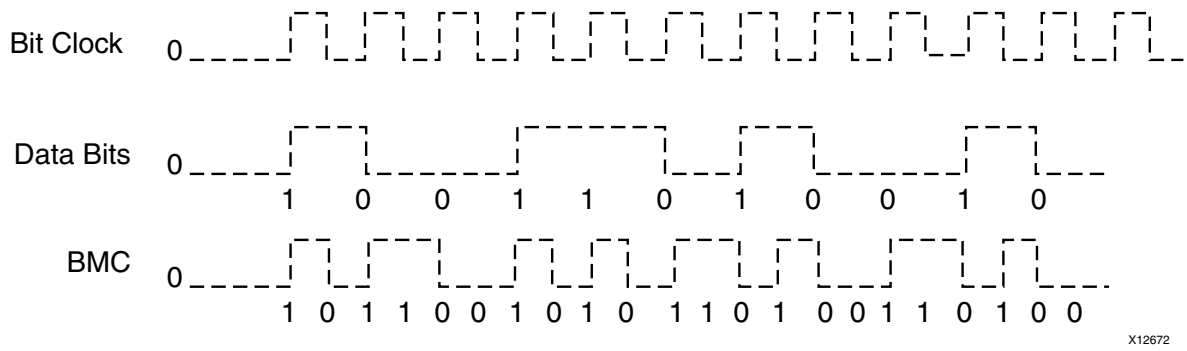

Figure 3-1: **Bi-Phase Mark Code**

The frequency of the clock is twice the bit rate. Every bit of the original data is represented as two logical states, which, together, form a cell. The length of a cell (time slot) is equal to the length of a data bit. The logical level at the start of a bit is always inverted to the level at the end of the previous bit. The level at the end of a bit is equal (a 0 is transmitted) or inverted (a 1 is transmitted) to the start of that bit. The BMC has either one or two transitions for every bit. Data bit 1 has two transitions during a bit period, and Data bit 0 has one transition during a bit period. As per protocol, except for the preambles, SPDIF audio data is transferred in the BMC format. Preambles violate the BMC to identify the channel information by the SPDIF receivers.

Clocking

The core uses three clock domains. An asynchronous FIFO is used for cross clocking domain transfers.

- **AUD_CLK_I** – The SPDIF audio interface works with this clock. The frequency of the clock should be greater than or equal to 512 times the audio sampling rate for SPDIF in receive mode. It has been tested to run as fast as 100 MHz.

  For SPDIF in transmit mode, the frequency of this clock should be a harmonic of the audio sampling rate and the corresponding divisor should be set in the SPDIF control register. For a 32 kHz sampling rate, 16.384 MHz is the minimum AUD_CLK_I rate required.

- **S_AXI_ACLK** – This is the processor domain. The AXI4-Lite interface to the SPDIF register access works with this clock. It has been tested to run as fast as 135 MHz.
- **S_AXIS_ACLK** – This is the audio streaming interface clock. This clock should be greater than or equal to 512 times the sampling rate to match the audio rates. It has been tested to run as fast as 100 MHz.

Table 3-3: **SPDIF Clock Frequencies**

Clock	Min	Max	Description
AUD_CLK_I	16 MHz	100 MHz	Audio clock input
S_AXI_ACLK	25 MHz	135 MHz	Host processor clock
S_AXIS_ACLK	16 MHz	100 MHz	Audio streamlining interface clock

Resets

The SPDIF core uses the following resets. Using any of these options resets the entire design.

- AXI4-Lite interface – `s_axi_aresetn`
- AXI Streaming interface – `s_axis_aresetn/m_axis_aresetn`
- Soft reset through the register interface

There is also a register bit to flush the FIFO alone when the reset internal ASYNC FIFO gets reset.

Design Parameters

Table 3-4 defines the design parameters of the SPDIF.

For SPDIF core parameters, the C_TRANSMIT_RECEIVE parameter value controls the SPDIF as a transmitter or receiver. When C_TRANSMIT_RECEIVE is 1, the SPDIF core receives the AXI4-Stream data and transmits over the SPDIF interface. When C_TRANSMIT_RECEIVE is 0, the SPDIF core receives the SPDIF audio data samples and transmits over the AXI4-Stream interface.

Also, the C_AXIS_BUFFER_SIZE parameter value decides the size of an async TX FIFO/RX FIFO being generated. The SPDIF core in transmit mode generates the TX FIFO and generates the RX FIFO in receive mode. The default value of the buffer size is 512. An Asynchronous FIFO is used, whose read depth is C_AXIS_BUFFER_SIZE - 1. The FIFO full condition is also generated when the depth is C_AXIS_BUFFER_SIZE - 1. Set the buffer size based on the streaming frequency and idle cycles. There is a five clock cycle latency associated with the Asynchronous FIFO to generate the FIFO full and FIFO empty flags. The FIFO used in the SPDIF core is block RAM based.

Table 3-4: **Design Parameters**

Generic	Feature/Description	Parameter Name	Allowable Values	Default Values	VHDL Type
System Parameter					
G1	Target FPGA family	C_FAMILY	virtex6, spartan6, Virtex7, Kintex7	virtex6	string
AXI4 Interconnect Parameters					
G2	AXI4 base address	C_BASEADDR	Valid Address (1)	0xFFFFFFFF (2)	std_logic_vector
G3	AXI4 high address	C_HIGHADDR	Valid Address (3)	0x00000000 (2)	std_logic_vector
AXI4-Lite Interface Parameters					
G4	AXI4-Lite address bus width	C_S_AXI_ADDR_WIDTH	9	9	integer
G5	AXI4-Lite data bus width	C_S_AXI_DATA_WIDTH	32 (4)	32 (4)	integer

Table 3-4: **Design Parameters *(Cont'd)***

Generic	Feature/Description	Parameter Name	Allowable Values	Default Values	VHDL Type
AXI4-Stream Interface Parameters					
G6	AXI4-Stream data width	C_AXIS_TDATA_WIDTH	32(5)	32(5)	integer
G7	AXI4-Stream interface ID width	C_AXIS_TID_WIDTH	3	3	integer
SPDIF Core Parameters					
G8	SPDIF Transmitter/Receiver Configuration parameter	C_TRANSMIT_RECEIVE	0, 1	0	integer
G9	SPDIF Buffer size	C_AXIS_BUFFER_SIZE	16, 32, 64, 128, 256, 512, and 1024	512	integer
G10	SPDIF Channel Status registers enable parameter	C_CSTATUS_REG	0 to 1	0	integer
G11	SPDIF User data registers enable parameter	C_USERDATA_REG	0 to 1	0	integer

Notes:

1. Set these parameters when there are multiple slaves connected to the AXI master through the AXI interconnect. If the AXI4-Lite interface of the SPDIF core is connected through `axi_ext_slave_conn`, these parameters need not be set. The C_BASEADDR parameter must be a multiple of the range, where the range is C_HIGHADDR - C_BASEADDR + 1.
2. An invalid default value is specified to ensure that the actual value is set.
3. The range specified by C_HIGHADDR - C_BASEADDR must be a power of 2 and greater than or equal to 0xFFF.
4. The AXI4-Lite interface data width is fixed to 32 bits.
5. The AXI4-Stream Master and Slave interface data widths are fixed to 32 bits.

Parameter – I/O Signal Dependencies

The dependencies between the SPDIF core design parameters and I/O signals are described in Table 3-5. In addition, when certain features are parameterized out of the design, the related logic is no longer a part of the design. In the SPDIF core, the transmit or receive functionality is configurable based on the parameter C_TRANSMIT_RECEIVE value.

Table 3-5: **Parameter – I/O Signal Dependencies**

Generic or Port	Name	Affects	Depends	Relationship Description
Design Parameters				
G4	C_S_AXI_ADDR_WIDTH	P3, P13	–	Affects the address width of the AXI4-Lite interface address signals.
G5	C_S_AXI_DATA_WIDTH	P6, P7, P16	–	Affects the data and Strobe width of the AXI4-Lite interface signals.

Table 3-5: **Parameter – I/O Signal Dependencies *(Cont'd)***

Generic or Port	Name	Affects	Depends	Relationship Description
G6	C_AXIS_TDATA_WIDTH	P26, P32	–	Affects the data width of the AXI4-Stream Master/Slave interface signals.
G7	C_AXIS_TID_WIDTH	P27, P33	–	Affects the ID width of the AXI4 Streaming Master/Slave Interface signals.
G8	C_TRANSMIT_RECEIVE	P20, P21,P22, P23, P24, P25, P26, P27, P28, P29, P30, P31, P32, P33	–	Affects the AXI4-Stream interface signals and SPDIF interface signals.
I/O Signals				
P3	S_AXI_AWADDR[C_S_AXI_ADDR_WIDTH-1:0]	–	G4	Port width depends on C_S_AXI_ADDR_WIDTH.
P13	S_AXI_ARADDR[C_S_AXI_ADDR_WIDTH-1:0]	–	G4	Port width depends on C_S_AXI_ADDR_WIDTH.
P6	S_AXI_WDATA[C_S_AXI_DATA_WIDTH-1:0]	–	G5	Port width depends on C_S_AXI_DATA_WIDTH.
P7	S_AXI_WSTRB[C_S_AXI_DATA_WIDTH/8-1:0]	–	G5	Port width depends on C_S_AXI_DATA_WIDTH.
P16	S_AXI_RDATA[C_S_AXI_DATA_WIDTH-1:0]	–	G5	Port width depends on C_S_AXI_DATA_WIDTH.
P20	SPDIF_I	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 0.
P21	SPDIF_O	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 1.
P22	M_AXIS_ACLK	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 0.
P23	M_AXIS_ARESETN	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 0.
P24	M_AXIS_TVALID	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 0.
P25	M_AXIS_TREADY	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 0.
P26	M_AXIS_TDATA[C_AXIS_TDATA_WIDTH-1:0]	–	G8, G6	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 0. The width of the port depends on C_AXIS_TDATA_WIDTH
P27	M_AXIS_TID[C_AXIS_TID_WIDTH-1:0]	–	G8, G7	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 0. The width of the port depends on C_AXIS_TID_WIDTH.
P28	S_AXIS_ACLK	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 1.

Table 3-5: **Parameter – I/O Signal Dependencies *(Cont'd)***

Generic or Port	Name	Affects	Depends	Relationship Description
P29	S_AXIS_ARESETN	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 1.
P30	S_AXIS_TVALID	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 1.
P31	S_AXIS_TREADY	–	G8	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 1.
P32	S_AXIS_TDATA[C_AXIS_TDATA_WIDTH-1:0]	–	G8, G6	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 1. The width of the parameter depends on C_AXIS_TDATA_WIDTH.
P33	S_AXIS_TID[C_AXIS_TID_WIDTH -1:0]	–	G8, G7	Port is valid only when the parameter C_TRANSMIT_RECEIVE is 1. The width of the port depends on C_AXIS_TID_WIDTH.

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

This chapter includes information about using Xilinx Vivado™ Design Suite to customize and generate the core.

GUI

Generating the Core

This section describes how to generate an SPDIF core with default values using the Xilinx Vivado IP Catalog.

To generate the core:

1. Start the Vivado IP Catalog.
2. Choose **File > New Project**.
3. Type a directory name.

   This example uses the directory name *design*.
4. To set project options:
   - Part Options

     From Target Architecture, select the desired family. For a list of supported families, see IP Facts, page 6.

     Note: If an unsupported silicon family is selected, the SPDIF core does not appear in the taxonomy tree.
   - Generation Options

     For Design Entry, select either **VHDL** or **Verilog**.
5. After creating the project, locate the SPDIF core in the taxonomy tree under **Standard Bus Interfaces > Spdif**.
6. Double-click the core to display the main SPDIF configuration screen (see Figure 4-1).

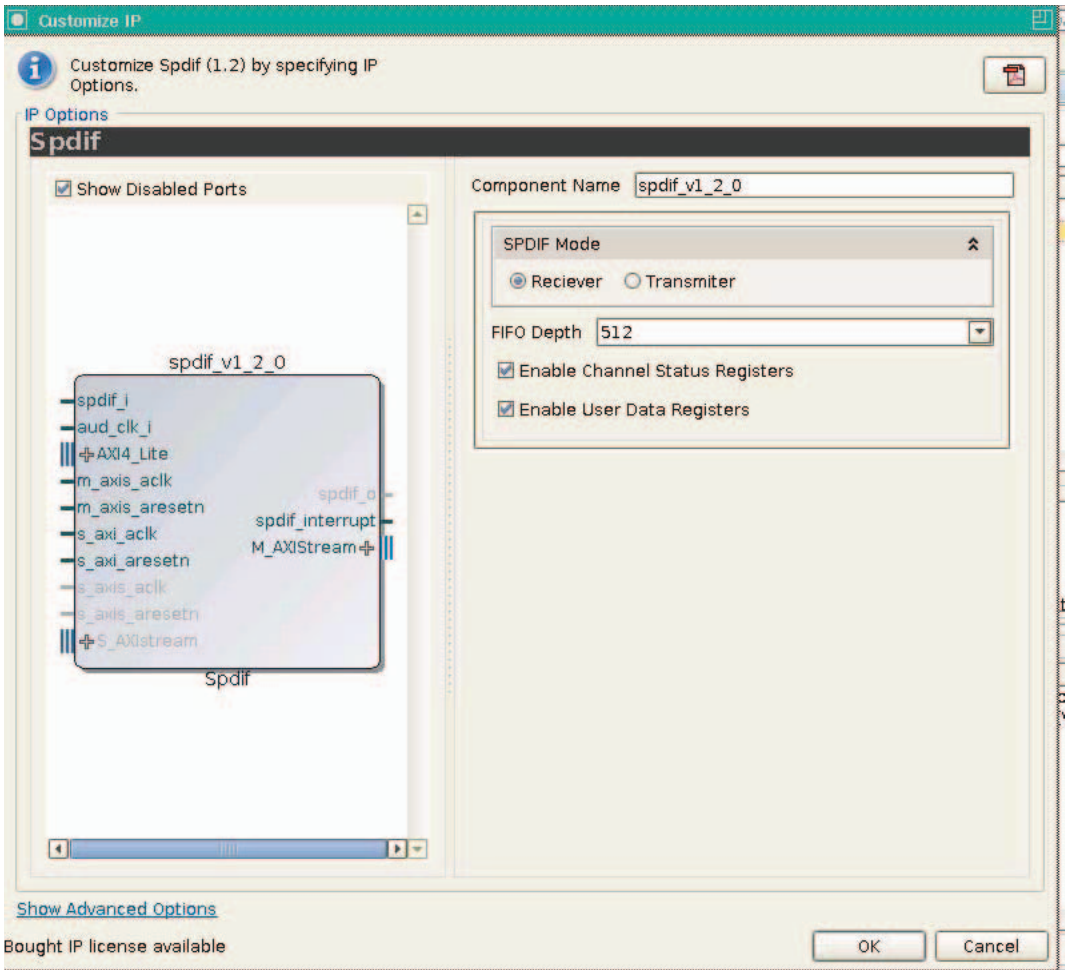

Figure 4-1: **SPDIF Main Screen**

In the Component Name field, enter a name for the core instance. This example uses the name *spdif_v1_2*.

7. After selecting the parameters from the GUI screens, click **OK**.

   The core and its supporting files, including the example design, are generated in the `project` directory.

Output Generation

This section provides detailed information about the files and the directory structure generated by the Xilinx Vivado tool.

In the IP Catalog project, click **Open IP Example Design in GUI** or typing the command `open_example_project [get_ips <component_name>]` in the TCL console. This invokes a separate example design project.

In this new project, <component_name>_exdes is the top module for synthesis and <component_name>_tb is the top module for simulation. The implementation or simulation of the example design runs from the example project.

The SPDIF core directories and their associated files are defined in this section. The directory structure for a Vivado design tools project under <project_name>/<project_name>.srcs/sources_1/ip/<component name> is the same as <project_directory>/<component name> for a CORE Generator™ tools project.

The directory structure is shown here:

- <project_name>/<project_name>.srcs/sources_1/ip/
  Top-level project directory. The name is user-defined.
- <project_name>/<project_name>.srcs/sources_1/ip/<component name>
  Core release note files
  - <component_name>example design
    Verilog and VHDL design files
  - <component_name>/implement
    Implementation script files
    - <component_name>/implement/results
      Results directory, created after implementation scripts are run, and contains implement script results
  - <component_name>/simulation
    Simulation scripts
    - <component_name>/simulation/functional
      Functional simulation files

<project_name>/<project_name>.srcs/sources_1/ip/<component name>

The directory contains all the Vivado tools project files.

Table 4-1: **Project Directory**

Name	Description
project_name>/<project_name>.srcs/sources_1/ip/<component name>	
<component_name>.xci	Vivado tools project-specific option file, can be used as an input to the Vivado tools.
<component_name>.{veo|vho}	VHDL or Verilog instantiation template.
<component_name>.xdc	Constraints file for core.

Back to Top

Constraining the Core

This chapter defines the constraint requirements of the SPDIF endpoint example design. An example Xilinx Design Constraints (XDC) is provided with the example design, which implements the constraints defined in this chapter.

When a Spartan®-6 FPGA is selected as the target device, the XDC is generated for an XC6SLX150T-FGG676-3 device as an example. The example designs and XDCs can be retargeted for other devices.

Information provided in this chapter indicates which constraints to modify when targeting devices other than those shown in the example designs.

Device, Package, and Speed Grade Selections

The SPDIF cores can be implemented in the devices listed in the IP Facts table on page 6 with the following attributes:

- Large enough to accommodate the cores
- With a fast enough speed grade to meet the frequency requirements

Clock Frequencies

To operate the core at the highest performance rating, the following constraints must be present. Prorate these numbers if slower performance is desired.

```
NET "S_AXI_ACLK" TNM_NET = S_AXI_ACLK;
TIMESPEC TS_S_AXI_ACLK = PERIOD "S_AXI_ACLK" 7.408 ns HIGH 50 %;

NET "AUD_CLK_I" TNM_NET = AUD_CLK_I;
TIMESPEC TS_AUD_CLK_I = PERIOD "AUD_CLK_I" 10 ns HIGH 50 %;

NET "S_AXIS_ACLK" TNM_NET = S_AXIS_ACLK;
TIMESPEC TS_S_AXIS_ACLK = PERIOD "S_AXIS_ACLK" 10 ns HIGH 50 %;
```

Clock Placement

There are no clock placement constraints.

Banking

There are no banking constraints.

I/O Standard and Placement

There are no I/O constraints.

Detailed Example Design

The Vivado™ Design Suite *Detailed Example Design* chapter is the same as the ISE® Design Suite version. For more information, see Chapter 9, Example Design.

SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core.

GUI

Generating the Core

This section describes how to generate an SPDIF core with default values using the Xilinx CORE Generator™ tool.

To generate the core:

1. Start the CORE Generator tool.

   For help starting and using the CORE Generator tool, see the *Xilinx CORE Generator Guide*, available from the ISE documentation web page.

2. Choose **File > New Project**.
3. Type a directory name.

   This example uses the directory name *design*.

4. To set project options:
   - Part Options

     From Target Architecture, select the desired family. For a list of supported families, see IP Facts, page 6.

     Note: If an unsupported silicon family is selected, the SPDIF core does not appear in the taxonomy tree.
   - Generation Options

     For Design Entry, select either **VHDL** or **Verilog**.

5. After creating the project, locate the SPDIF core in the taxonomy tree under **Standard Bus Interfaces > Spdif**.

6. Double-click the core to display the main SPDIF configuration screen (see Figure 7-1).

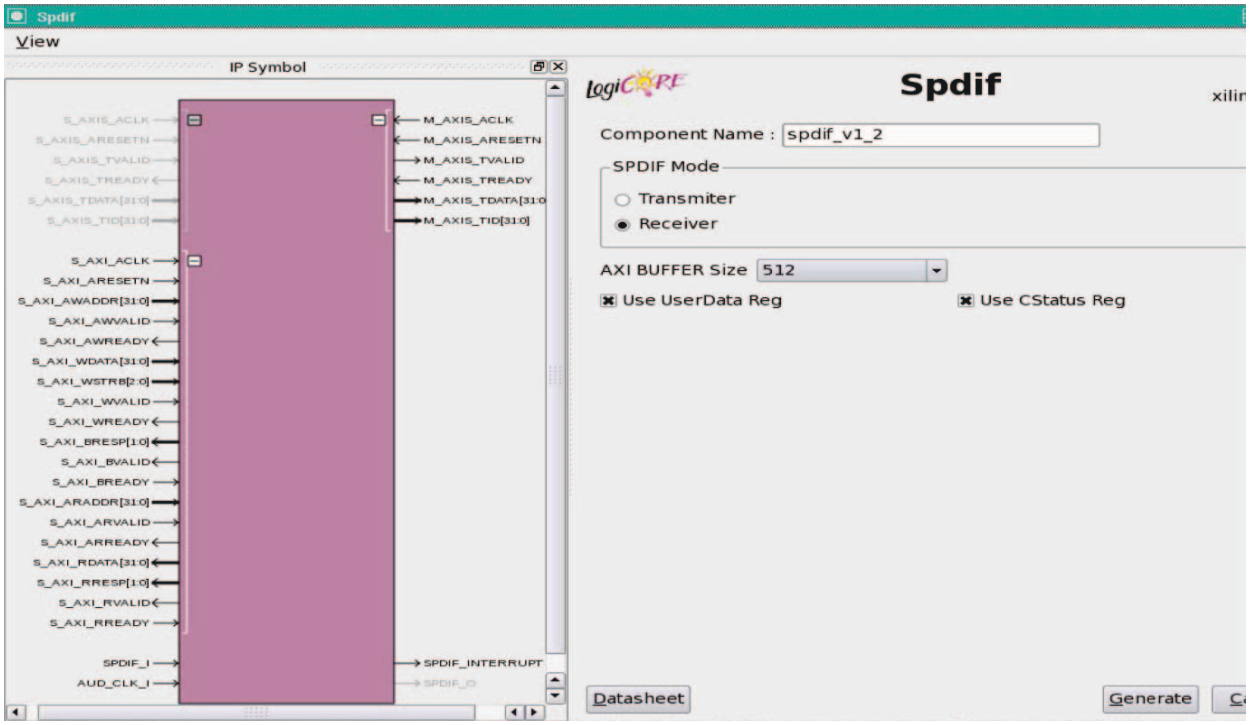

Figure 7-1: **SPDIF Main Screen**

In the Component Name field, enter a name for the core instance. This example uses the name *spdif_v1_2*.

7. After selecting the parameters from the GUI screens, click **Finish**.

   The core and its supporting files, including the example design, are generated in the `project` directory.

Output Generation

The SPDIF core directories and their associated files are defined in this section. The directory structure is shown here:

- `<project directory>`
  Top-level project directory. The name is user-defined.
  - `<project_directory/<component_name>`
  - `<component_name>/doc`
    Product documentation
  - `<component_name>example design`
    Verilog and VHDL design files
  - `<component_name>/implement`
    Implementation script files
    - `<component_name>/implement/results`
      Functional simulation files
  - `<component_name>/simulation`
    Simulation scripts
    - `<component_name>/simulation/functional`
      Functional simulation files

The SPDIF core directories and their associated files are defined in this section.

<project directory>

The `<project directory>` contains all the CORE Generator tool project files.

Table 7-1: **Project Directory**

Name	Description
<project directory>	
<component_name>.ngc	Top-level netlist
<component_name>.v[hd]	Verilog or VHDL simulation model
<component_name>.xco	CORE Generator tool 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|vho}	VHDL or Verilog instantiation template.

Table 7-1: **Project Directory *(Cont'd)***

Name	Description
<component_name>_readme.txt	Core name release notes file.

Back to Top

<component_name>example design

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

Table 7-2: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_exdes.ucf	Provides example constraints necessary for processing the SPDIF core using the Xilinx implementation tools.
<component_name>_exdes.v[hd]	The VHDL or Verilog top-level file for the example design. It instantiates the SPDIF core.

Back to Top

<component_name>/doc

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

Table 7-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
pg045_spdif.pdf	*LogiCORE IP SPDIF v1.2 Product Guide*

Back to Top

<component_name>/implement

The `implement` directory contains the core implementation script files. Generated for Full-System Hardware Evaluation and Full license types.

Table 7-4: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
implement.{bat|sh}	A Windows (BAT) or Linux (SH) script that processes the example design.
xst.prj	The XST project file for the example design that lists all of the source files to be synthesized. Only available when the CORE Generator tool project option is set to **ISE** or **Other**.

Table 7-4: **Implement Directory *(Cont'd)***

Name	Description
xst.scr	The XST script file for the example design used to synthesize the core. Only available when the CORE Generator tool Vendor project option is set to **ISE** or **Other**.

Back to Top

<component_name>/implement/results

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

Table 7-5: **Results Directory**

Name	Description
<project_dir>/<component_name>/implement/results	
Implement script result files.	

Back to Top

<component_name>/simulation

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

Table 7-6: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
spdif_v1_2_tb.v	Verilog test bench file that instantiates all verification models and generates the necessary clocks required by the test bench.
axilite_master.v	Verilog test bench file. AXI4-Lite master model that configures the core registers.
axis_rx_checker.v	Verilog test bench file. AXI4-Stream checker model that verifies the data on AXI4-Stream master interface of the SPDIF core.
axistream_master.v	Verilog test bench file. AXI4-Stream master model that drives the data on the AXI4-Stream slave interface of the SPDIF core.
axistream_slave.v	Verilog test bench file. AXI4-Stream slave model that receives the data driven on the AXI4-Stream master interface of the SPDIF core.
spdif_rx_driver.v	Verilog test bench file. SPDIF RX driver model that drives the core input SPDIF line.
spdif_tx_checker.v	Verilog test bench file. SPDIF line checker model that verifies the data on the SPDIF output line.

Back to Top

<component_name>/simulation/functional

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

Table 7-7: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	A macro file for ModelSim that compiles the HDL sources and runs the simulation.
wave_mti.do	A macro file for ModelSim that opens a wave window and adds key signals to the wave viewer. This file is called by the `simulate_mti.do` file and is displayed after the simulation is loaded.

Back to Top

Constraining the Core

This chapter defines the constraint requirements of the SPDIF endpoint example design. An example User Constraints File (UCF) is provided with the example design, which implements the constraints defined in this chapter.

When a Spartan®-6 FPGA is selected as the target device, the UCF is generated for an XC6SLX150T-FGG676-3 device as an example. The example designs and UCFs can be retargeted for other devices.

Information provided in this chapter indicates which constraints to modify when targeting devices other than those shown in the example designs.

Device, Package, and Speed Grade Selections

The SPDIF cores can be implemented in the devices listed in the IP Facts table on page 6 with the following attributes:

- Large enough to accommodate the cores
- With a fast enough speed grade to meet the frequency requirements

Clock Frequencies

To operate the core at the highest performance rating, the following constraints must be present. Prorate these numbers if slower performance is desired.

```
NET "S_AXI_ACLK" TNM_NET = S_AXI_ACLK;
TIMESPEC TS_S_AXI_ACLK = PERIOD "S_AXI_ACLK" 7.408 ns HIGH 50 %;

NET "AUD_CLK_I" TNM_NET = AUD_CLK_I;
TIMESPEC TS_AUD_CLK_I = PERIOD "AUD_CLK_I" 10 ns HIGH 50 %;

NET "S_AXIS_ACLK" TNM_NET = S_AXIS_ACLK;
TIMESPEC TS_S_AXIS_ACLK = PERIOD "S_AXIS_ACLK" 10 ns HIGH 50 %;
```

Clock Placement

There are no clock placement constraints.

Banking

There are no banking constraints.

I/O Standard and Placement

There are no I/O constraints.

Detailed Example Design

Example Design

This section provides instructions to generate an SPDIF core quickly, run the design through implementation with the Xilinx tools, and simulate the example design using the provided demonstration test bench.

Overview

Figure 9-1 and Figure 9-2 illustrate the SPDIF receiver and transmitter example designs, respectively. The example designs consist of the following:

- SPDIF netlist
- HDL wrapper which instantiates the SPDIF netlist
- Customizable demonstration test bench to simulate the example design

The SPDIF example designs have been tested with Xilinx ISE® Design Suite and the Mentor Graphics ModelSim simulator.

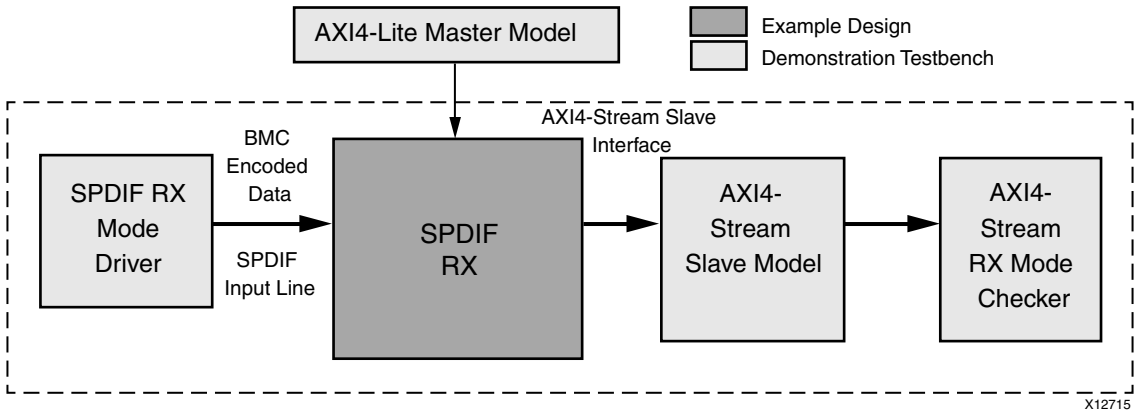

Figure 9-1: **SPDIF Receiver Example Design**

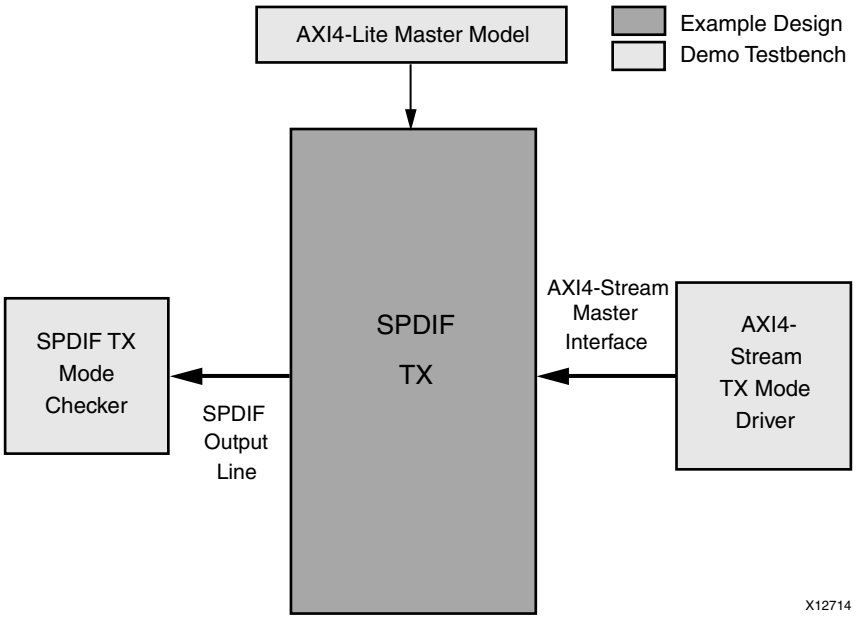

Figure 9-2: **SPDIF Transmitter Example Design**

Implementing an Example Design

After generating a core with either a Full-System Hardware Evaluation or Full license, the netlists and example design can be processed by the Xilinx implementation tools. The generated output files include scripts to assist you in running the Xilinx tools.

To implement an SPDIF example design, open a command prompt or terminal window and type these commands:

- For Windows:

```
ms-dos> cd <proj_dir>\quickstart\implement
ms-dos> implement.bat
```

- For Linux:

```
Linux-shell% cd <proj_dir>/quickstart/implement
Linux-shell% ./implement.sh
```

  These commands execute a script that synthesizes, builds, maps, and places-and-routes the example design. The script then generates a post-par simulation model for use in timing simulation. The resulting files are placed in the `results` directory.

Simulating an Example Design

The SPDIF core provides a quick way to simulate and observe the behavior of the core by using the provided example designs. The simulation model provided is in Verilog.

Setting Up for Simulation

The Xilinx UNISIM and SIMPRIM libraries must be mapped into the simulator. If the UNISIM or SIMPRIM libraries are not set for your environment, see the *Synthesis and Simulation Guide* [Ref 10] in the Xilinx Software Manuals for assistance compiling Xilinx simulation models. Simulation scripts are provided for ModelSim.

Functional Simulation

This section provides instructions for running a functional simulation of the SPDIF core using Verilog. Functional simulation models are provided when the core is generated. Implementing the core before simulating the functional models is not required.

To run a Verilog functional simulation of the example designs:

1. Set the current directory to:

   ```
   <quickstart>/simulation/functional/
   ```

2. Launch the simulation script.

   ModelSim: `vsim -do simulate_mti.do`

   The simulation script compiles the functional simulation models and demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Demonstration Test Bench

Test Bench Functionality

The demonstration test bench is a Verilog file that exercises the example designs and the core itself. The test bench is provided *as-is*. Feedback is welcome but enhancements are not guaranteed.

The test bench consists of the following:

- AXI4-Lite master model to configure the core registers
- SPDIF RX driver model to drive the core input SPDIF line
- AXI4-Stream slave model to receive the data driven on AXI4-Stream master interface of the SPDIF core
- AXI4-Stream master model to drive the data on the AXI4-Stream slave interface of the SPDIF core

- AXI4-Stream checker model to verify the data on the AXI4-Stream master interface of the SPDIF core
- SPDIF line checker model to verify the data on the SPDIF output line
- Test bench top file that instantiates all verification models and generates the necessary clocks required by the test bench

SPDIF Core in RX Mode

Figure 7-1, page 38 shows the demonstration test bench in RX mode.

- **Clock Generator** – The sampling frequency at which the audio data is received on the core SPDIF input line is fixed at 32 kHz. `AUD_CLK_I`, with 16.384 MHz frequency, is generated by the top-level module as required for the 32 kHz sampling frequency. The clock generator generates a reference clock for use by the test bench components.
- **AXI4-Lite Master Model** – In the AXI4-Lite Master Model, the SPDIF control register is set to `32'h00000001` to enable the SPDIF core.
- **SPDIF RX Mode Driver** – A sub-frame is a 32-bit audio sample with a 4-bit preamble and 28-bit audio data. A frame is a combination of 2 sub-frames (channel 0 and channel 1). The SPDIF RX Mode Driver model generates 1000 sub-frames to drive on the core SPDIF input line. This model also generates the preambles as required by the SPDIF protocol.
  The 28-bit audio data generated by this model is an incremental pattern. This data is driven in bi-phase encoded format as specified by the SPDIF protocol.
  The channel number is driven such that alternate channel IDs exist (channel 0 and channel 1).
- **AXI4-Stream Slave Model** – This model follows the AXI4-Stream protocol to receive the 32-bit streaming data driven on the AXI4-Stream master interface of the SPDIF core.
- **AXI4-Stream RX Mode Checker** – This model generates the expected incremental data and checks the data integrity of audio data received on the AXI4-Stream master interface of the SPDIF core. This model also checks the number of sub-frames (TIDs) received on the AXI4-Stream master interface of the SPDIF core.
  This model displays the completion of 50 sub-frames received on the AXI4-Stream master interface.

SPDIF Core in TX Mode

This section describes the functionality of the blocks in the demonstration test bench in TX mode.

- **Clock Generator** – The sampling frequency at which the audio data is transmitted on the core SPDIF output line is fixed at 32 kHz. `AUD_CLK_I` is generated by the top-level

module as required for the 32 kHz sampling frequency. AUD_CLK_I is generated for 49.152 MHz.

- **AXI4-Lite Master Model** – In the AXI4-Lite Master Model, the SPDIF control register is set to 32'h0000000D to enable the SPDIF core. This configuration sets the TX clock configuration bits to a value of 4'h3 (divisor 24).
- **AXI4-Stream TX Mode Driver** – The AXI4-Stream TX Mode Driver generates 1536 sub-frames of 32 bits to drive on the AXI4-Stream slave interface of the SPDIF core as defined in the AXI4-Stream protocol. This model generates the 4-bit preambles as required by the SPDIF protocol. The 28-bit audio data generated by this model is an incremental pattern.
  The channel number is driven such that alternate channel IDs exist (channel 0 and channel 1).
- **SPDIF TX Mode Checker** – This model receives the data driven on the SPDIF output line. Every 64 bits of data are concatenated on the line, because the audio data is bi-phase encoded on the SPDIF output line. This model decodes the 56-bit data and generates the 28-bit audio data.
  The channel ID is determined based on the received preamble on the SPDIF output line. This model generates the expected 32-bit incremental data and compares with the data received on SPDIF output line.
  This model displays the completion of 50 sub-frames received.

SECTION IV: APPENDICES

Verification, Compliance, and Interoperability

Migrating

Debugging

Additional Resources

Appendix A

Verification, Compliance, and Interoperability

Simulation

This section provides information on implementation and simulation scripts.

Implementation Scripts

Note: Present only with a Full license.

The implementation script is either a shell script (SH) or batch file (BAT) that processes the example design through the Xilinx tool flow. It is located at:

- Linux

  `<project_dir>/<component_name>/implement/implement.sh`

- Windows

  `<project_dir>/<component_name>/implement/implement.bat`

When the CORE Generator tool is run with the Full System Hardware Evaluation or Full license types, the implementation script performs these steps:

- Synthesizes the HDL example design files using XST
- Runs NGDBuild to consolidate the core netlist and the example design netlist into the NGD file containing the entire design
- Maps the design to the target technology
- Place-and-routes the design on the target device
- Performs static timing analysis on the routed design using Timing Analyzer (TRCE)
- Generates a bitstream
- Enables Netgen to run on the routed design to generate a VHDL or Verilog netlist (as appropriate for the Design Entry project setting) and timing information in the form of SDF files

The Xilinx tool flow generates several output and report files. These are saved in the following directory which is created by the implementation script:

```
<project_dir>/<component_name>/implement/results
```

Simulation Scripts

Functional Simulation

The test scripts are ModelSim macros that automate the simulation of the test bench. They are available from the following location:

```
<project_dir>/<component_name>/simulation/functional/
```

The test script performs these tasks:

- Compiles the structural UNISIM simulation model
- Compiles HDL Example Design source code
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Opens a Wave window and adds signals of interest (wave_mti.do/wave_ncsim.sv)
- Runs the simulation to completion

Appendix B

Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

For information on migrating to the Vivado™ Design Suite, see *Vivado Design Suite Migration Methodology Guide* (UG911) [Ref 9].

For a complete list of Vivado User and Methodology Guides, see the Vivado Design Suite - 2012.4 User Guides web page.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step process for debugging process and a flow diagram to guide you through debugging the SPDIF core.

The following topics are included in this appendix:

Finding Help on Xilinx.com

To help in the design and debug process when using the SPDIF, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support WebCase.

Documentation

This product guide is the main document associated with the SPDIF. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Release Notes

Known issues for all cores, including the SPDIF are described in the IP Release Notes Guide (XTP025).

Known Issues

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Answer Records for the SPDIF

There are no known issues or Answer Records for the SPDIF core.

Contacting Technical Support

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Debug Tools

There are many tools available to address SPDIF design issues. It is important to know which tools are useful for debugging various situations.

Example Design

The SPDIF is delivered with an example design that can be synthesized, complete with functional test benches. Information about the example design can be found in *Chapter 9, Detailed Example Design*.

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information for using the ChipScope Pro tool, see www.xilinx.com/tools/cspro.htm.

Vivado Lab Tools

Vivado Lab Tools inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. Vivado Lab Tools allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed.

License Checkers

If the IP requires a license key, the key must be verified. The ISE and Vivado tool flows have a number of license check points for gating licensed IP through the flow. If the license check succeeds the IP may continue generation, otherwise generation halts with error. License checkpoints are enforced by the following tools:

- ISE flow: XST, NgdBuild, Bitgen
- Vivado flow: Vivado Synthesis, Vivado Implementation, write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The ChipScope tool is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the ChipScope tool for debugging the specific problems.

Many of these common issues can also be applied to debugging design simulations. Details are provided in the General Checks section.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the LOCKED port.
- If your outputs go to 0, check your licensing.
- Check all the clocks are toggling or not.
- Check the active state of each reset and make sure no interface is in reset condition.
- Check the AUD_CLK_I requirements are met for SPDIF Transmitter/SPDIF Receiver.
- Check whether the core is configured and then enabled or not before looking at the functionality of the core.

Interface Debug

AXI4-Lite Interfaces

Write a specific value to any read/write register and it reads back the same register to verify that the interface is functional. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `S_AXI_ACLK and AUD_CLK_I` inputs are connected and toggling.
- The interface is not being held in reset, and `S_AXI_ARESETN` is an active-Low reset.
- If the simulation has been run, verify in simulation and/or a ChipScope tool capture that the waveform is correct for accessing the AXI4-Lite interface.

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If transmit `<interface_name>_tready` is stuck low following the `<interface_name>_tvalid` input being asserted, the core cannot send data.
- If the receive `<interface_name>_tvalid` is stuck low, the core is not receiving data.

- Check that the S_AXIS_ACLK input for SPDIF Transmitter and M_AXIS_ACLK input for SPDIF receiver are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed.
- Check core configuration.

Other Interfaces

In SPDIF Receiver mode:

- Check if the SPDIF interface line is active or not in case of SPDIF Receiver.
- Check the AUD_CLK_I requirement (that is, the clock should be minimum of eight times the SPDIF bit rate).
- Check the Status register whether the core capturing the bit rate as expected or not.

In SPDIF Transmitter mode:

- Check the AUD_CLK_I frequency. It should be harmonic to the Audio sampling rate.
- Check whether the proper divisor value is set in Control register to generate a SPDIF line rate properly.

Appendix D

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Appendix C, Debugging and 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.

References

Unless otherwise noted, IP references are for the product documentation page. These documents provide supplemental material useful with this product guide:

1. *7 Series FPGAs Overview* (DS180)
2. *Virtex-6 Family Overview* (DS150)
3. *Spartan-6 Family Overview* (DS160)
4. *LogiCORE IP AXI Interconnect Data Sheet* (DS768)
5. *LogiCORE IP Interrupt Control Data Sheet* (DS516)
6. *LogiCORE IP AXI Lite IPIF Data Sheet* (DS765)
7. *LogiCORE IP DisplayPort User Guide* (UG767)
8. *AXI Reference Guide* (UG761)
9. Vivado Design Suite documentation: www.xilinx.com/cgi-bin/docs/rdoc?v=2012.4;t=vivado+docs
10. *Synthesis and Simulation Guide* (UG626)
11. *AXI4 AMBA AXI Protocol Version: 2.0 Specification*

12. *IEC-60958-3 Standard Specification*

13. AMBA AXI4-Stream Protocol Specification

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/24/12	1.0	Initial Xilinx release
07/25/12	2.0	Updated for Vivado Design Suite 2012.2, Zynq features, and ISE 14.2.
12/18/12	3.0	• Updated to ISE Design Suite 14.4 and Vivado Design Suite 2012.4. • Updated description to SPDIF Control register Bits[5:2]. • Updated Appendix Debug section.

Notice of Disclaimer