DesignWare DW_apb_i2s Databook

DW_apb_i2s – Product Code

1.11a
July 2018

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com

Contents

Revision History

This table shows the revision history for the databook from release to release. This is being tracked from version 1.08a onward.

Version	Date	Description
1.11a	July 2018	**Added:** ■ "DMA Handshaking Interface" on page 40 ■ "DW_apb_i2s as Transmitter—With DMA Handshake Interface" on page 149 ■ "DW_apb_i2s as Receiver—With DMA Handshake Interface" on page 150 **Updated:** ■ Version changed for 2018.07a release ■ "Performance" on page 173 ■ Figure 1-2 and Figure 2-1 ■ "Parameter Descriptions" on page 55, "Signal Descriptions" on page 65, "Register Descriptions" on page 81 and "Internal Parameter Descriptions" on page 181 are auto extracted with change bars from the RTL **Removed:** ■ Chapter 2, "Building and Verifying a Component or Subsystem" and added the contents in the newly created user guide.
1.10a	October 2016	■ Version number change to 2016.10a ■ "Parameter Descriptions" on page 55 and "Register Descriptions" on page 81 auto-extracted from the RTL ■ Removed the "Running Leda on Generated Code with coreConsultant" section, and reference to Leda directory in Table 2-1 ■ Removed the "Running Leda on Generated Code with coreAssembler" section, and reference to Leda directory in Table 2-4 ■ Added "Running VCS XPROP Analyzer" ■ Replaced Figure 2-2 and Figure 2-3 to remove references to Leda ■ Moved Internal Parameter Descriptions to Appendix ■ Added an entry for the xprop directory in Table 2-1 and Table 2-4. ■ Added "DW_apb_i2s Registers" on page 39

(Continued)

Version	Date	Description
1.09a	June 2015	■ Added “Running SpyGlass® Lint and SpyGlass® CDC” ■ Added “Running SpyGlass on Generated Code with coreAssembler” ■ “Signal Descriptions” on page 65 auto-extracted from the RTL ■ Added “Internal Parameter Descriptions” on page 181 ■ Added Appendix A, “Synchronizer Methods”
1.08a	June 2014	■ Version change for 2014.06a release ■ Added "Transaction Example" section in the "Functional Description" chapter ■ Added “Performance” section in the “Integration Considerations” chapter
1.07a	May 2013	■ Made minor corrections in the description of the ws_out, sw_slv, and sdox signals ■ Removed a note regarding DesignWare Verification IP (VIP) in “Verification” chapter, as DW_apb_i2s does not use DesignWare VIP ■ Updated the template.
1.06e	Sep 2012	Added the product code on the cover and in Table 1-1
1.06e	Mar 2012	Version change for 2012.03a release
1.06d	Nov 2011	Version change for 2011.11a release
1.06c	Oct 2011	Version change for 2011.10a release
1.06b	Jun 2011	■ Updated system diagram in Figure 1-1 ■ Enhanced “Related Documents” section in Preface
1.06b	Apr 2011	■ Clarification added on the use of the sclk_en and sclk_gate outputs ■ Removed signals starting with dma_*, which are not implemented in RTL
1.06a	Jan 2011	Corrected conditions for programmed gating value in SCLKG field of CCR register
1.06a	Dec 2010	Removed signals starting with dma_*, which are not implemented in RTL
1.06a	Sep 2010	Corrected names of include files and vcs command used for simulation
1.05a	May 2010	Corrected FIFO depths from 2, 4, 8, and 16 bits to I2S_RX_WORDSIZE_x or I2S_TX_WORDSIZE_x
1.05a	Mar 2010	Version change for 2010.03a release
1.04a	Dec 2009	■ Corrected usage flow diagrams for DW_apb_i2s as a receiver and as a transmitter ■ Enhanced information on writing to a transmit channel and reading from a receive channel ■ Modified procedures in the “Programming the DW_apb_i2s” chapter ■ Modified parameter descriptions ■ Updated databook to new template for consistency with other IIP/VIP/PHY databooks

(Continued)

Version	Date	Description
1.04a	May 2009	Removed references to QuickStarts, as they are no longer supported
1.04a	Oct 2008	Version change for 2008.10a release
1.03a	Jun 2008	Version change for 2008.06a release
1.02b	Apr 2008	Corrected gating relationship of sclk and sclk_gate
1.02b	Dec 2007	■ Updated for revised installation guide and consolidated release notes titles ■ Changed references of “Designware AMBA” to simply “DesignWare”
1.02b	Jun 2007	Version change for 2007.06a release

Preface

This databook provides information that you need to interface the DW_apb_i2s to the Advanced Peripheral Bus (APB). The DW_apb_i2s conforms to the *AMBA Specification, Revision 2.0* from Arm®.

The information in this databook includes an overview, pin and parameter descriptions, a memory map, and functional behavior of the component. An overview of the testbench, a description of the tests that are run to verify the coreKit, and synthesis information for the component are also provided.

Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Functional Description" describes the functional operation of the DW_apb_i2s.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_apb_i2s.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_apb_i2s signals.
- Chapter 5, "Register Descriptions" describes the programmable registers of the DW_apb_i2s.
- Chapter 6, "Programming the DW_apb_i2s" provides information needed to program the configured DW_apb_i2s.
- Chapter 7, "Verification" provides information on verifying the configured DW_apb_i2s.
- Chapter 8, "Integration Considerations" includes information you need to integrate the configured DW_apb_i2s into your design.
- Appendix A, "Synchronizer Methods", documents the synchronizer methods (blocks of synchronizer functionality) used in DW_apb_i2s to cross clock boundaries.
- Appendix B, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals chapter.
- Appendix C, "Glossary" provides a glossary of general terms.

Related Documentation

- *Using DesignWare Library IP in coreAssembler* – Contains information on getting started with using DesignWare SIP components for AMBA 2 and AMBA 3 AXI components within coreTools
- *coreAssembler User Guide* – Contains information on using coreAssembler
- *coreConsultant User Guide* – Contains information on using coreConsultant

To see a complete listing of documentation within the DesignWare Synthesizable Components for AMBA 2, see the *Guide to Documentation for DesignWare Synthesizable Components for AMBA 2 and AMBA 3 AXI*.

The DW_apb_i2s component interfaces to the I^{2}S bus, which is protected by patents held by Philips (NXP). Synopsys does not convey permission for I^{2}S use; you must obtain this permission directly from Philips (NXP). Additionally, DW_apb_i2s conforms to the *I2S Bus Specification* from Philips (NXP). For licensing information and to obtain this specification, see the Philips (NXP) web site.

Web Resources

- DesignWare IP product information: http://www.designware.com
- Your custom DesignWare IP page: http://www.mydesignware.com
- Documentation through SolvNet: http://solvnet.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Customer Support

To obtain support for your product:

- First, prepare the following debug information, if applicable:
  - For environment setup problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, use the following menu entry:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This menu entry gathers all the Synopsys product data needed to begin debugging an issue and writes it to the file <*core tool startup directory*>/debug.tar.gz.
  - For simulation issues outside of coreConsultant or coreAssembler:
    - Create a waveforms file (such as VPD or VCD)
    - Identify the hierarchy path to the DesignWare instance
    - Identify the timestamp of any signals or locations in the waveforms that are not understood
- Then, contact Support Center, with a description of your question and supplying the requested information, using one of the following methods:
  - *For fastest response*, use the SolvNet website. If you fill in your information as explained, your issue is automatically routed to a support engineer who is experienced with your product. The **Sub Product** entry is critical for correct routing.

Go to http://solvnet.synopsys.com/EnterACall and click **Open A Support Case** to enter a call. Provide the requested information, including:

- **Product:** DesignWare Library IP
- **Sub Product:** AMBA
- **Tool Version:** *product version number*
- **Problem Type:**
- **Priority:**
- **Title:** DW_apb_i2s
- **Description:** For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

After creating the case, attach any debug files you created in the previous step.

- Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
  - Include the Product name, Sub Product name, and Tool Version number in your e-mail (as identified earlier) so it can be routed correctly.
  - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
  - Attach any debug files you created in the previous step.
- Or, telephone your local support center:
  - North America:
    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - All other countries:
    https://www.synopsys.com/support/global-support-centers.html

Product Code

Table 1-1 lists all the components associated with the product code for DesignWare APB Advanced Peripherals.

Table 1-1 DesignWare APB Advanced Peripherals – Product Code: 3772-0

Component Name	Description
DW_apb_i2c	A highly configurable, programmable master or slave i2c device with an APB slave interface
DW_apb_i2s	A configurable master or slave device for the three-wire interface (I2S) for streaming stereo audio between devices
DW_apb_ssi	A configurable, programmable, full-duplex, master or slave synchronous serial interface
DW_apb_uart	A programmable and configurable Universal Asynchronous Receiver/Transmitter (UART) for the AMBA 2 APB bus

1 Product Overview

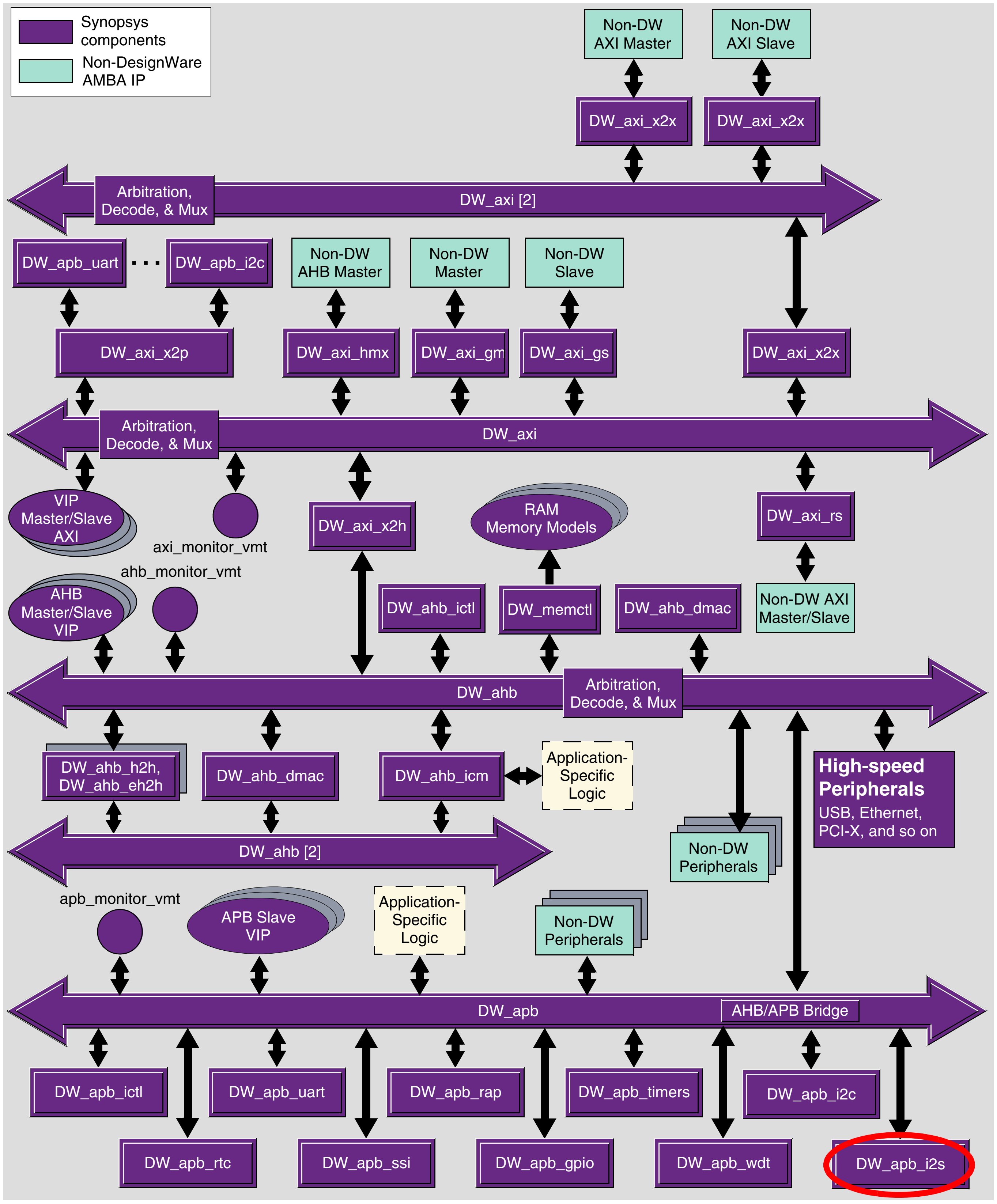
This chapter describes the DesignWare APB Inter-IC Sound (I^2S) Bus (referred to as DW_apb_i2s), which is a serial link designed for digital audio systems. The DW_apb_i2s component is an AMBA 2.0-compliant Advanced Peripheral Bus (APB) slave device and is part of the family of DesignWare Synthesizable Components.

1.1 DesignWare System Overview

The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) components, and AMBA version 3.0-compliant AXI (Advanced eXtensible Interface) components.

Figure 1-1 illustrates one example of this environment, including the AXI bus, the AHB bus, and the APB bus. Included in this subsystem are synthesizable IP for AXI/AHB/APB peripherals, bus bridges, and an AXI interconnect and AHB bus fabric. Also included are verification IP for AXI/AHB/APB master/slave models and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

Figure 1-1 Example of DW_apb_i2s in a Complete System



You can connect, configure, synthesize, and verify the DW_apb_i2s within a DesignWare subsystem using coreAssembler, documentation for which is available on the web in the *coreAssembler User Guide*.

If you want to configure, synthesize, and verify a single component such as the DW_apb_i2s component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

1.2 General Product Description

The DW_apb_i2s is a configurable, synthesizable, and programmable component designed to be used in systems that process digital audio signals, such as:

- A/D and D/A converters
- digital signal processors
- error correction for compact disc and digital recording
- digital filters
- digital input/output interfaces

The Inter-IC Sound (I^2S) Bus is a simple three-wire serial bus protocol developed by Philips to transfer stereo audio data. The bus only handles the transfer of audio data; hence control and subcoding signals need to be transferred separately using a different bus protocol (such as I^2C).

1.2.1 DW_apb_i2s Features

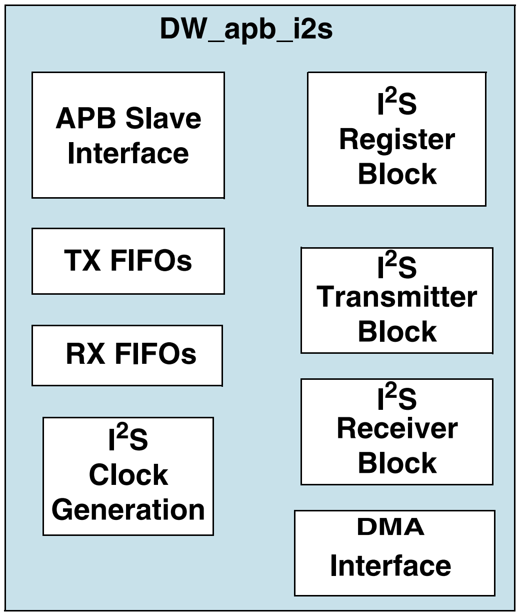
DW_apb_i2s has the following features:

- APB data bus widths of 8, 16, and 32 bits
- I^2S transmitter and/or receiver based on the Philips I^2S serial protocol
- Configurable number of stereo channels (up to 4) for both transmitter and receiver
- Full duplex communication due to the independence of transmitter and receiver
- Asynchronous clocking of APB bus and I^2S sclk
- Master or slave mode of operation
- Audio data resolutions of 12, 16, 20, 24, and 32 bits
- External sclk gating and enable signals
- Configurable FIFO depth of 2, 4, 8, and 16 words, where the wordsize is determined by *I2S_RX_WORDSIZE_x* or *I2S_TX_WORDSIZE_x*
- Configurable support for programmable DMA registers
- Programmable FIFO thresholds
- Component parameters for configurable software driver support
- Support for DMA Hardware Handshaking Interface

1.2.2 DW_apb_i2s Block Diagram

Figure 1-2 illustrates a block diagram of the DW_apb_i2s.

Figure 1-2 Block Diagram of DW_apb_i2s



1.2.3 I^{2}S Terminology

The following terms are used throughout this manual and are defined as follows

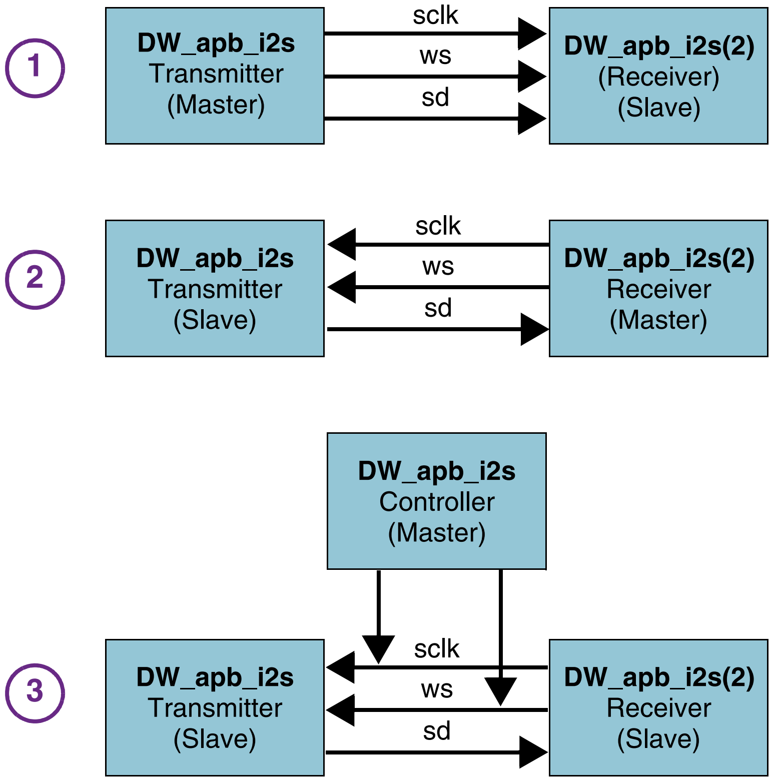
- sclk – serial clock
- ws – word select
- sd – serial data
- Transmitter – device that places data on the sd line and is clocked by sclk and ws
- Receiver – device that receives data from the sd line and is clocked by sclk and ws
- Master – when configured as a master, DW_apb_i2s initializes the ws signal and supplies the clock gating and clock enabling signals
- Slave – when configured as a slave, DW_apb_i2s responds to externally generated sclk and ws signals

1.2.4 Overview of DW_apb_i2s

The bus consists of a serial data line (sd), a word select line (ws), and a serial clock (sclk). The serial data line is time multiplexed to allow the transfer of two data streams (such as, left and right stereo data). DW_apb_i2s can be configured to have up to four data channels for both transmit and receive operations. In essence, if you configured DW_apb_i2s with the maximum channels for transmitter and receiver transactions, there would be a total of eight data lines in addition to the serial clock and word select—10 wires total.

Figure 1-3 illustrates three simple system configurations for the DW_apb_i2s component. Note that the examples show a second instantiation of the DW_apb_i2s component, which acts as the receiver configured as either a slave or master.

Figure 1-3 Simple System Configurations for DW_apb_i2s

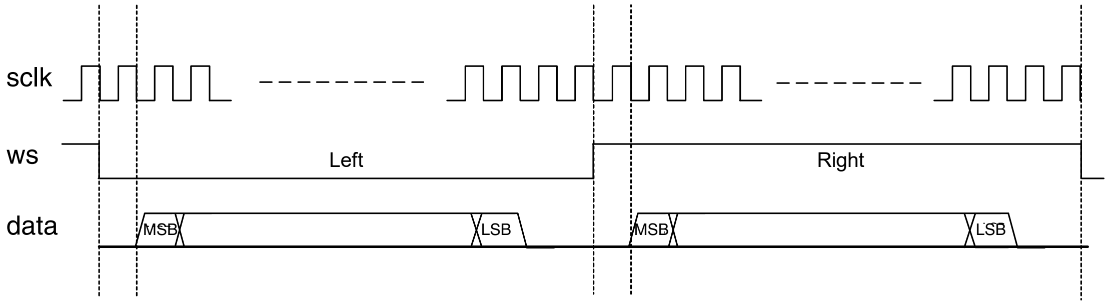

1
DW_apb_i2s Transmitter (Master)
sclk
ws
sd
DW_apb_i2s(2) (Receiver) (Slave)
2
DW_apb_i2s Transmitter (Slave)
sclk
ws
sd
DW_apb_i2s(2) Receiver (Master)
DW_apb_i2s Controller (Master)
3
DW_apb_i2s Transmitter (Slave)
sclk
ws
sd
DW_apb_i2s(2) Receiver (Slave)

Examples 1 and 2 in the figure show that either the transmitter or the receiver can act as the bus master. The master is responsible for generating the shared sclk and ws clocking signals. In complex systems where there may be several transmitters and receivers, a separate system master can be used. As illustrated in example 3 in the figure, this system master can also be combined with one of the transmitters or receivers in the system. The “controller” in this example is enabled and disabled by configuring the component to act as a master and by programming the clock enable and clock configuration registers.

The serial data is transmitted in two’s complement format with the most significant bit (MSB) first. This means that the transmitter and receiver can have different word lengths, and neither the transmitter nor receiver needs to know what size words the other can handle. If the word being transferred is too large for the receiver, the least significant bits (LSB) are truncated. Similarly, if the word size is less than what the receiver can handle, the data is zero padded.

The word select line is used to time the multiplexed data streams. For instance, when ws is low, the word being transferred is left stereo data; when ws is high, the word being transferred is right stereo data. This format is illustrated in Figure 1-4. For standard I^{2}S formats, the MSB of a word is sent one sclk cycle after a ws change. Serial data sent by the transmitter can be synchronized with either the negative edge or positive edge of the sclk signal. However, the receiver must latch the serial data on the rising edge of sclk.

Figure 1-4 I^{2}S Stereo Frame Format



For more details about the operation of the DW_apb_i2s, see "Functional Description" on page 23.

Source code for this component is available on a per-project basis as a DesignWare core; contact your local sales office for the details.

1.3 Standards Compliance

The DW_apb_i2s component conforms to the *AMBA Specification, Revision 2.0* from Arm®. Readers are assumed to be familiar with this specification.

The DW_apb_i2s component interfaces to the I2S bus, which is protected by patents held by Philips (NXP). Synopsys does not convey permission for I2S use; you must obtain this permission directly from Philips (NXP). Additionally, DW_apb_i2s conforms to the I2S Bus Specification from Philips (NXP). For licensing information and to obtain this specification, see the Philips (NXP) web site.

1.4 Verification Environment Overview

The DW_apb_i2s includes an extensive verification environment, which sets up and invokes your selected simulation tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation.

The "Verification" on page 155 chapter discusses the testbench of DW_apb_i2s.

1.5 Licenses

Before you begin using the DW_apb_i2s, you must have a valid license. For more information, see "Licenses" section in the *DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI Installation Guide*.

1.6 Where To Go From Here

At this point, you may want to get started working with the DW_apb_i2s component within a subsystem or by itself. Synopsys provides several tools within its coreTools suite of products for the purposes of configuration, synthesis, and verification of single or multiple synthesizable IP components—coreConsultant and coreAssembler. For information on the different coreTools, see *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_apb_i2s component, see *DesignWare Synthesizable Components for AMBA 2 User Guide*.

For more information about implementing your DW_apb_i2s component within a DesignWare subsystem using coreAssembler, see *DesignWare Synthesizable Components for AMBA 2 User Guide*.

2 Functional Description

This chapter describes the functional behavior of DW_apb_i2s in more detail.

2.1 Overview

The I^2S bus can only handle audio data transmissions; subcoding and controls are handled by another device, such as an I^2C. The I^2S protocol requires a minimum of three wires—data (sd), word select (ws), and serial clock (sclk)—keeping the design simple and the pin count minimal. However, DW_apb_i2s can be configured to have up to four channels for transmit and receive operations, making the maximum configured wire count 10.

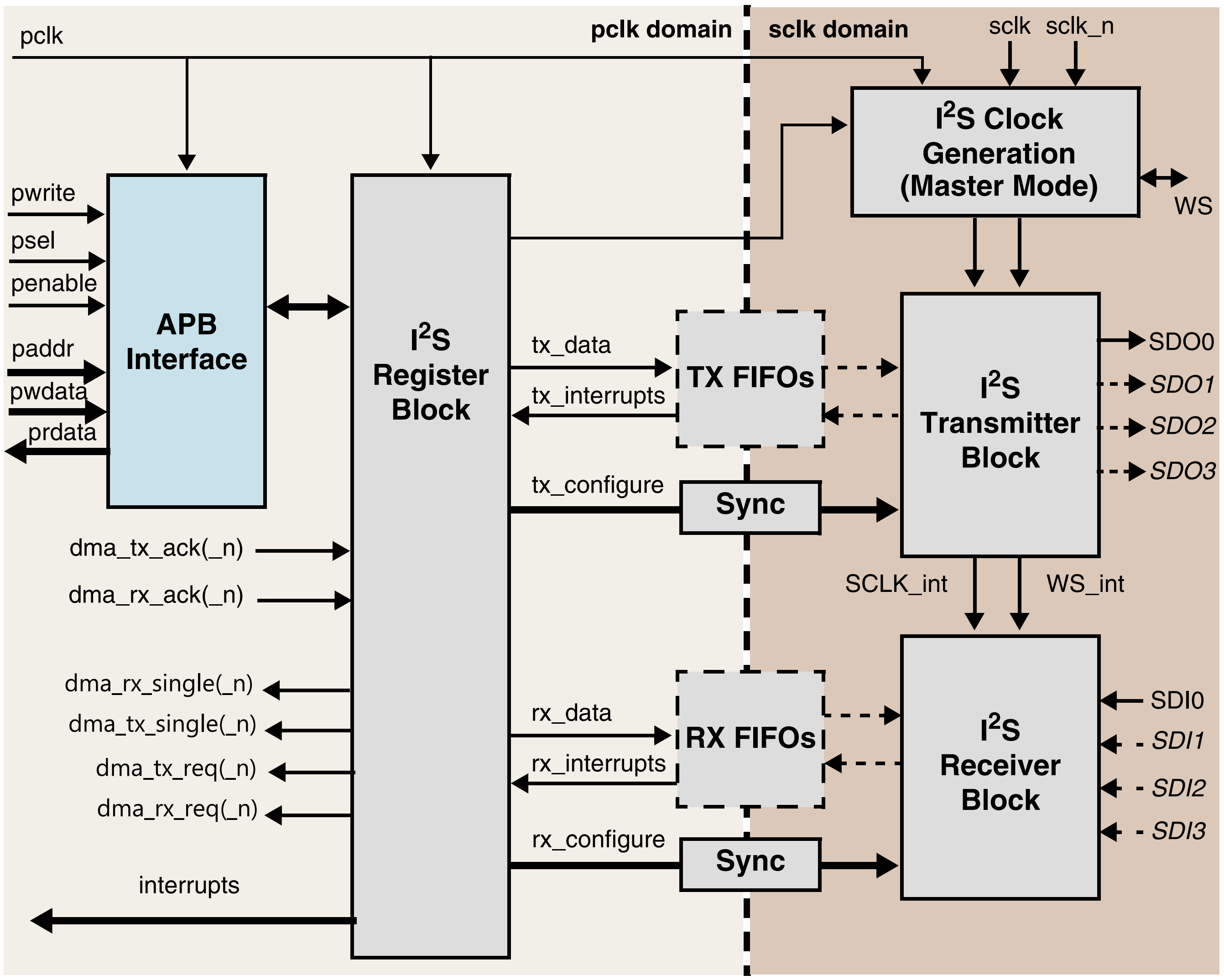
The component also can be configured to operate as either a master or a slave (the default mode). When configured as a master, DW_apb_i2s initializes the word select (ws_out signal) and supplies the clock gating (sclk_gate) and clock enabling (sclk_en) signals. When operating as a slave, DW_apb_i2s responds to externally generated sclk and ws signals.

Whether configured as a master or slave, an external sclk and an inverted version of sclk need to be supplied to the device through input signals sclk and sclk_n.

DW_apb_i2s supports the standard I^2S frame format for transmitting and receiving data —the MSB of a word is sent one sclk cycle after a word select change.

Figure 2-1 shows the block diagram of DW_apb_i2s.

Figure 2-1 DW_apb_i2s Block Diagram



2.2 DW_apb_i2s Enable

You must enable the DW_apb_i2s component before any data can be received or transmitted into the FIFOs. To enable the component, set the I^{2}S Enable (IEN) bit of the I^{2}S Enable Register (IER) to 1. When you disable the device, it acts as a global disable. To disable DW_apb_i2s, set IER[0] to 0.

After disable, the following events occur:

- TX and RX FIFOs are cleared, and read/write pointers are reset;
- Any data in the process of being transmitted or received is lost;
- All other programmable enables (such as transmitter/receiver block enables and individual TX/RX channel enables) in the component are overridden;

- Generation of master mode clock signals sclk_en, ws_out and sclk_gate are disabled (for instance, they are held low).

  When DW_apb_i2s is enabled and configured as a master, the device always starts in the left stereo data cycle (ws = 0), and one sclk cycle later transitions to the right stereo data cycle (ws = 1). This allows for half a frame of sclks to write data to the TX FIFOs and to ensure that any connected slave receivers do not miss the start of the data frame (for instance, the ws 1-to-0 transition) once the sclk restarts. (When DW_apb_i2s is configured as a slave, ws is externally supplied.)

On reset, the IER[0] is set to 0 (disable).

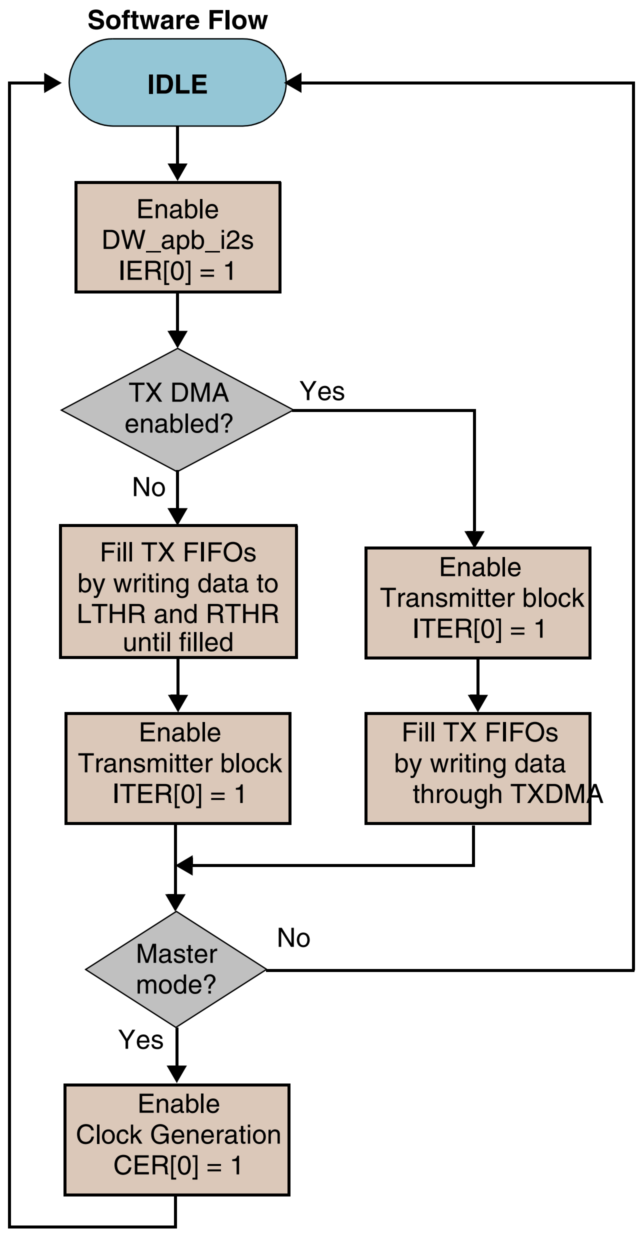
2.3 DW_apb_i2s as Transmitter

The DW_apb_i2s component can be configured to support up to four stereo I^{2}S transmit (TX) channels. These channels can operate in either master or slave mode. By default, DW_apb_i2s is configured in slave mode. Stereo data pairs (such as, left and right audio data) written to a TX channel through the APB bus are shifted out serially on the appropriate serial data out line (sdo0, sdo1, sdo2, sdo3). The shifting is timed with respect to the serial clock (sclk) and the word select line (ws).

The instantiation of the I^{2}S transmitter block and the number of TX channels is determined by the two configuration parameters: Transmitter Block Enabled (I2S_TRANSMITTER_BLOCK) and Number of Transmit Channels (I2S_TX_CHANNELS), respectively. By default, DW_apb_i2s is configured with one transmit channel. For more information about these parameters, see “Parameter Descriptions” on page 55.

Figure 2-2 illustrates the basic usage flow for DW_apb_i2s when it acts as a transmitter.

Figure 2-2 Basic Usage Flow – DW_apb_i2s as Transmitter


Software Flow
IDLE
Enable DW_apb_i2s IER[0] = 1
TX DMA enabled?
Yes
No
Fill TX FIFOs by writing data to LTHR and RTHR until filled
Enable Transmitter block ITER[0] = 1
Enable Transmitter block ITER[0] = 1
Fill TX FIFOs by writing data through TXDMA
Master mode?
No
Yes
Enable Clock Generation CER[0] = 1

2.3.1 Transmitter Block Enable

The Transmitter Block Enable (TXEN) bit of the I^2S Transmitter Enable Register (ITER) globally turns on and off all of the configured TX channels. To enable the transmitter block, set ITER[0] to 1. To disable the block, set ITER[0] to 0.

When the transmitter block is disabled, the following events occur:

- Outgoing data is lost and the channel outputs are held low;
- Data in the TX FIFOs are preserved and the FIFOs can be written to;

- Any previous programming (like changes in word size, threshold levels, and so on) of the TX channels is preserved; and
- Any individual TX channel enables are overridden.

  When the transmitter block is enabled, if there is data in the TX FIFOs, the channel resumes transmission on the next left stereo data cycle (such as when the ws line goes low).

When the block is disabled, you can perform any of the following procedures:

- Program (or further program) TX channel registers
- Flush the TX FIFOs by programming the Transmitter FIFOs Reset bit of the Transmitter FIFO Flush Register (TXFFR[0] = 1)
- Flush an individual channel's TX FIFO by programming the Transmit Channel FIFO Reset (TXCHFR) bit of the Transmit FIFO Flush Register (TFFx[0] = 1, where *x* is the channel number)

On reset, the ITER[0] is set to 0 (disable).

2.3.2 Transmit Channel Enable

Each transmit channel has its own enable/disable that can be set independently of the other channels to allow the reprogramming of a channel and to flush the channel's TX FIFOs while other TX channels are transmitting. This enable/disable is controlled by bit 0 of the Transmitter Enable Register (TERx, where *x* is the channel number). For example, to enable TX Channel 1, write 1 to TER1[0]. To disable this channel, write 0 to TER1[0].

When a TX channel is disabled, the following occurs:

- Outgoing stereo data is lost;
- Channel output is held low;
- Data in the TX FIFO is preserved, and the FIFO can be written to; and
- Any previous programming of the TX channel's registers is preserved, and the registers can be further reprogrammed.

When a TX channel is disabled, you can flush the channel's TX FIFO by programming the Transmit Channel FIFO Reset (TXCHFR) bit of the Transmit FIFO Flush (TFFx[0] = 1, where *x* is the channel number). When the TX channel is enabled, if there is data in the TX FIFO, the channel resumes transmission on the next left stereo data cycle (such as, when the ws line goes low).

On reset, the TFFx[0] is set to 1 (enable).

2.3.3 Transmit Channel Audio Data Resolution

Each TX channel is initially configured with a maximum audio data resolution as set by the Maximum Audio Resolution parameter (I2S_TX_WORDSIZE_*x*, where x is the channel number). A TX channel can be reprogrammed during operation to any supported audio data resolution that is less than I2S_TX_WORDSIZE_*x*.

For example, if the TX Channel 2 is initially configured with a 32-bit audio resolution, it can be programmed to support a resolution of 12, 16, 20, 24, or 32 bits. However, if TX Channel 3 is initially configured with a 20-bit audio resolution, it can only be programmed to support resolution of 12, 16, or 20 bits. Any other resolution values are considered invalid. Furthermore, if the channel is programmed with an invalid audio resolution, the TX channel defaults to I2S_TX_WORDSIZE_*x*.

Reprogramming of the audio resolution ensures that the MSB of the data is still transmitted first if the resolution of the data to be sent is reduced. Changes to the resolution are programmed through the Word Length (WLEN) bits of the Transmitter Configuration Registers (TCRx[2:0], where *x* is the channel number). The channel must be disabled prior to any resolution changes.

On reset or if an invalid resolution is selected, the TX channel's audio data resolution defaults back to the initial parameter setting of I2S_TX_WORDSIZE_*x*. For more information about this parameter, see "Parameter Descriptions" on page 55.

2.3.4 Transmit Channel FIFOs

Each Transmit Channel has two FIFO banks for left and right stereo data. The FIFOs can be configured as determined by the I2S_FIFO_DEPTH_GLOBAL parameter. The FIFO width is determined by the "Maximum Audio Resolution – Transmit Channel X" parameter (I2S_TX_WORDSIZE_*x*, where *x* is the channel number).

There are several ways to clear the TX FIFOs and reset the read/write pointers as described as follows;

- on reset
- by disabling DW_apb_i2s (IER[0] = 0)
- by flushing the transmitter block (TXFFR[0] = 1)
- by flushing an individual TX channel (TFFx[0] = 1, where *x* is the channel number)

You must disable the transmitter block/channel before the transmitter block and individual channel FIFO can be flushed.

The TX FIFO Empty Threshold Trigger Level parameter (I2S_TX_FIFO_THRE_*x*, where *x* is the channel number) sets the default trigger threshold level for the TX FIFO. The trigger level can be set to any value in the range of 0 to I2S_TX_FIFO_*x* – 1. When this level is reached, a transmit channel empty interrupt is generated. This level can be reprogrammed during operation by writing to the Transmit Channel Empty Trigger (TXCHET) bits of the Transmit FIFO Configuration Register (TFCRx[3:0], where *x* is the channel number).

You must disable the TX channel prior to changing the trigger level.

For more information about the I2S_FIFO_DEPTH_GLOBAL and I2S_TX_FIFO_THRE_*x* parameters, see "Parameter Descriptions" on page 55.

2.3.5 Transmit Channel Interrupts

All interrupts in DW_apb_i2s can be configured as active low or active high by setting the "Polarity of Interrupt Signals is Active High?" parameter (I2S_INTR_POL). Each TX channel generates two interrupts: TX FIFO Empty and Data Overrun.

- TX FIFO Empty interrupt – This interrupt is asserted when the empty trigger threshold level for the TX FIFO is reached. When this interrupt is included on the I/O, it appears on the outputs tx_emp_*x*_intr (where *x* is the channel number). A TX FIFO Empty interrupt is cleared by writing data to the TX FIFO to bring its level above the empty trigger threshold level for the channel.

- Data Overrun interrupt –This interrupt is asserted when an attempt is made to write to a full TX FIFO (any data being written is lost while data in the FIFO is preserved). When this interrupt is included on the I/O, it appears on the outputs tx_or_*x*_intr (where *x* is the channel number). A Data Overrun interrupt is cleared by reading the Transmit Channel Overrun (TXCHO) bit [0] of the Transmit Overrun Register (TORx, where *x* is the channel number).

The interrupt status of any TX channel can be determined by polling the Interrupt Status Register (ISRx, where *x* is the channel number). The TXFE bit [4] indicates the status of the TX FIFO Empty interrupt, while the TXFO bit [5] indicates the status of the Data Overrun interrupt.

Both the TX FIFO Empty and Data Overrun interrupts can be masked off by writing 1 in the Transmit Empty Mask (TXFEM) and Transmit Overrun Mask (TXFOM) bits of the Interrupt Mask Register (IMRx, where *x* is the channel number), respectively. This prevents the interrupts from driving their output lines, however, the ISR*x* always shows the current status of the interrupts regardless of any masking.

The setting of the "Multiple Interrupt Output Ports Present?" configuration parameter (I2S_INTERRUPT_SIGNALS) affects whether these two interrupts are included on DW_apb_i2s's interface. Table 2-1 shows the specific interrupt signal to appear on the I/O according to the setting of I2S_INTERRUPT_SIGNALS.

Table 2-1 TX Channel Interrupt Signals on I/O

Interrupt	Signal on I/O
I2S_INTERRUPT_SIGNALS = True (multiple interrupts on I/O)	
TX FIFO Empty	tx_emp_*x*_intr
Data Overrun	tx_or_*x*_intr
I2S_INTERRUPT_SIGNALS = False (shared single interrupt on I/O)	
combined interrupt signal	intr

For more information about the I2S_INTERRUPT_SIGNALS parameter, see "Parameter Descriptions" on page 55.

2.3.6 Writing to a Transmit Channel

The stereo data pairs to be transmitted by a TX channel are written to the TX FIFOs through the Left Transmit Holding Register (LTHRx, where *x* is the channel number) and the Right Transmit Holding Register (RTHRx, where *x* is the channel number). All stereo data pairs must be written using the following two-stage process:

1. Write left stereo data to LTHR*x*
2. Write right stereo data to RTHR*x*.

Note You must write stereo data to the device in this order, otherwise, the interrupt and status lines values are invalid, and the left/right stereo pairs might be transmitted out of sync.

When TX DMA is enabled (I2S_TX_DMA = 1), data to be transmitted by TX channels are written to the TX FIFOs through the TXDMA register rather than through LTHR*x* and RTHR*x*. Data is written cyclically through all enabled TX channels starting from the lowest-numbered enabled channel. After a stereo data pair is transmitted, the component points to the next enabled channel.

The following example describes the behavior of the TXDMA register for a component that has been configured with four Transmit channels, where Channels 0 and 2 are enabled.

Order of transmitted data:

1. Ch0 — Left Data
2. Ch0 — Right Data
3. Ch2 — Left Data
4. Ch2 — Right Data
5. Ch0 — Left Data
6. Ch0 — Right Data, and so on

The RTXDMA register resets TXMDA to the lowest-enabled Channel. The RTXDMA register can be written to at any stage of the TXDMA transmit cycle; however, it has no effect when the component is in the middle of a stereo pair transmit.

The following example describes the operation of this register for a system with four Transmit channels, where all the channels are enabled.

Order of transmitted data:

1. Ch0 — Left Data
2. Ch0 — Right Data
3. RTXDMA Reset
4. Ch0 — Left Data
5. Ch0 — Right Data
6. Ch1 — Left Data
7. RTXDMA Reset — No effect (read not complete)
8. Ch1 — Right Data, and so on.
9. Ch2 — Left Data
10. Ch2 — Right Data
11. RTXDMA Reset
12. Ch0 — Left Data
13. Ch0 — Right Data

When DW_apb_i2s is enabled, if the TX FIFO is empty and data is not written to the FIFOs before the next left cycle, the channel outputs zeros for a full frame (left and right cycle). Transmission only commences if there is data in the TX FIFO prior to the transition to the left data cycle. In other words, if the start of the frame is missed, the channel output idles until the next available frame.

If the APB bus width is less than the configured or programmed audio resolution, multiple writes are required to write each stereo word. For example, if the Maximum Audio Resolution for Transmit Channel 1 is 20 (I2S_TX_WORDSIZE_1 = 20) and APB_DATA_WIDTH = 8, then three writes per register are required to write data to LTHR1 and RTHR1. However, if the audio resolution of the channel is reprogrammed to be 12 bits, then only two writes per register are required (the third write is ignored by the device). Thus, if the audio resolution is reduced, there is no need to perform the extra write to pad the data with leading zeros. For more information about these parameters, see "Parameter Descriptions" on page 55.

Note Data should only be written to the FIFO when it is not full. Any attempt to write to a full FIFO results in that data being lost and a Data Overrun interrupt being generated.

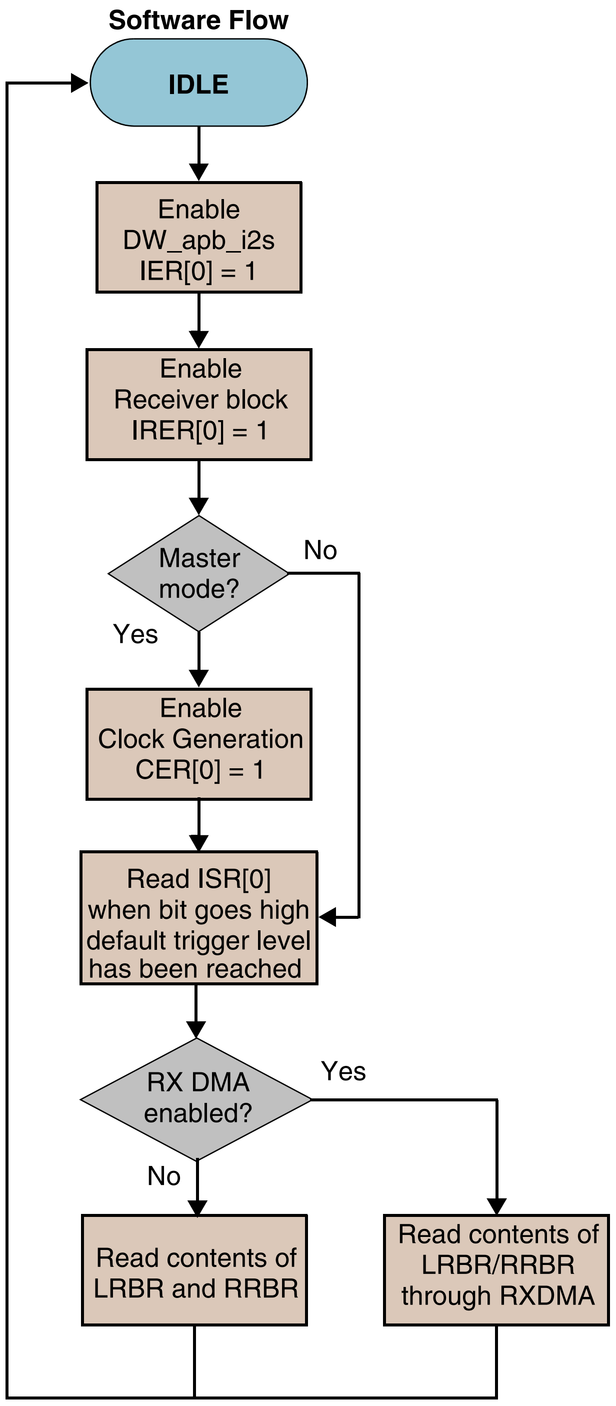
2.4 DW_apb_i2s as Receiver

DW_apb_i2s can be configured to support up to four stereo I^2S receive (RX) channels. These channels can operate in either master or slave mode. By default, DW_apb_i2s is configured in slave mode. Stereo data pairs (such as, left and right audio data) are received serially from a data input line (sdi0, sdi1, sdi2, sdi3). These data words are stored in RX FIFOs until they are read through the APB bus. The receiving is timed with respect to the serial clock (sclk) and the word select line (ws).

The instantiation of the receiver block and the number of RX channels is determined by two configuration parameters: Receiver Block Enabled (I2S_RECEIVER_BLOCK) and Number of Receive Channels (I2S_RX_CHANNELS), respectively. By default, DW_apb_i2s is configured with one receive channel. For more information about configuration parameters, see "Parameter Descriptions" on page 55.

Figure 2-3 illustrates the basic usage flow for DW_apb_i2s when it acts as a receiver.

Figure 2-3 Basic Usage Flow – DW_apb_i2s as Receiver


Software Flow
IDLE
Enable DW_apb_i2s IER[0] = 1
Enable Receiver block IRER[0] = 1
Master mode?
No
Yes
Enable Clock Generation CER[0] = 1
Read ISR[0] when bit goes high default trigger level has been reached
RX DMA enabled?
Yes
No
Read contents of LRBR and RRBR
Read contents of LRBR/RRBR through RXDMA

2.4.1 Receiver Block Enable

The Receiver Block Enable (RXEN) bit of the I^{2}S Receiver Enable Register (IRER) enables/disables all configured RX channels. To enable the receiver block, set IRER[0] to '1.' To disable the block, set this bit to '0.'

When the receiver block is disabled, the following events occur:

- Incoming data is lost;
- Data in the RX FIFOs is preserved and the FIFOs can be read;
- Any previous programming (such as changes in word size, threshold levels, and so on) of the RX channels is preserved; and
- Any individual RX channel enable is overridden. Enabling the channel resumes receiving on the next left stereo data cycle (for instance, when ws goes low).

When the block is disabled, you can perform any of the following procedures:

- Program (or further program) the RX channel registers;
- Flush the RX FIFOs by programming the Receiver FIFOs Reset (RXFR) bit of the Receiver FIFO Flush Register (RXFFR[0] = 1).
- Flush an individual channel's RX FIFO by programming the Receive Channel FIFO Reset (RXCHFR) bit of the Receive FIFO Flush Register (RFFx [0] = 1, where *x* is the channel number).

On reset, IRER[0] is set to 0 (disable).

2.4.2 Receive Channel Enable

Each RX channel has its own enable/disable that can be set independently of the other channels to allow programming of the channel and to clear the channel's RX FIFO while other RX channels are still receiving data. This enable/disable is controlled by bit 0 of the Receiver Enable Register (RERx[0], where *x* is the channel number). For example, to enable RX Channel 1, write 1 to RER1[0]. To disable this channel, write 0 to RER1[0].

When the RX channel is disabled, the following occurs:

- Incoming data is lost;
- Data in the RX FIFO is preserved;
- FIFO can be read;
- Previous programming of the RX channel is preserved; and
- RX channel can be further programmed.

When the RX channel or block is disabled, you can flush the channel's RX FIFO by writing 1 in bit 0 of the Receive FIFO Flush Register (RFFx, where *x* is the channel number). When the channel is enabled, it resumes receiving on the next left stereo data cycle (for instance. when ws line goes low).

On reset, the RFFx[0] is set to 1 (enable).

2.4.3 Receive Channel Audio Data Resolution

Each RX channel is initially configured with a maximum audio data resolution as set by the "Max Audio Resolution – Receive Channel X" parameter (I2S_RX_WORDSIZE_*x*, where *x* is the channel number). An RX channel can be programmed during operation to any supported audio data resolution that is less than I2S_RX_WORDSIZE_*x*.

For example, if the RX Channel 2 is initially configured with a 32-bit audio resolution, it can be programmed to support resolutions of 12 16, 20, 24, or 32 bits. However, if RX Channel 3 is initially configured with a 20-bit audio resolution, it can only be programmed to support resolutions of 12, 16, or 20 bits. Any other resolution values are considered invalid. Additionally, if the channel is programmed with an invalid audio resolution, the RX channel defaults to I2S_RX_WORDSIZE_*x*.

This programming ensures that the LSB of the received data is placed in the LSB position of the RX FIFO if the resolution of the data being received is reduced. Changes to the resolution are programmed through the Word Length (WLEN) bits of the Receive Configuration registers (RCRx[3:0], where *x* is the channel number). The channel must be disabled prior to any resolution changes.

The RX channel also supports unknown data resolutions. If the received word is greater than the configured channel resolution, the least significant bits are ignored. If the received word is less than the configured/programmed channel resolution, the least significant bits are padded with zeros.

On reset or if an invalid resolution is selected, the RX channel's audio data resolution defaults back to the initial parameter setting of I2S_RX_WORDSIZE_*x*.

For more information about the I2S_RX_WORDSIZE_*x* parameter, see "Parameter Descriptions" on page 55.

2.4.4 Receive Channel FIFOs

Each Receive Channel has two FIFO banks for left and right stereo data. The FIFOs can be configured as determined by the "FIFO Depth for RX and TX Channels?" parameter (I2S_FIFO_DEPTH_GLOBAL). The FIFO width is determined by the configured maximum data resolution for the channel (I2S_RX_WORDSIZE_*x*, where *x* is the channel number).

The RX FIFOs can be cleared and the read/write pointers reset in a number ways, as described as follows:

- on reset
- by disabling DW_apb_i2s (IER[0] = 0)
- by flushing the receiver block (RXFFR[0] = 1)
- by flushing an individual RX channel (RFFx[0] = 1, where *x* is the channel number)

Before you flush the receiver block or individual channels, you must disable the receiver block or channel.

The RX FIFO Data Available Level parameter (I2S_RX_FIFO_THRE_*x*, where *x* is the channel number) sets the default data available trigger level for the RX FIFO. When this level is reached, a RX channel data available interrupt is generated. The valid values are 0 to FIFO_DEPTH–1, which correspond to trigger levels of 1 to FIFO_DEPTH (for example, Trigger Level = Configured Value + 1). This level can be reprogrammed during operation through the Receive Channel Data Trigger (RXCHDT) bits of the Receive FIFO Configuration Register (RFCRx[3:0], where *x* is the channel number). The RX channel needs to be disabled prior to any changes in the trigger level.

For more information about I2S_RX_FIFO_THRE_*x* and I2S_RX_WORDSIZE_*x* parameters, see "Parameter Descriptions" on page 55.

2.4.5 Receive Channel Interrupts

All interrupts in DW_apb_i2s can be configured as active low or active high through the "Polarity of Interrupts Signals is Active High?" parameter (I2S_INTR_POL). Each RX channel generates two interrupts: RX FIFO Data Available and Data Overrun.

- RX FIFO Data Available interrupt – This interrupt is asserted when the trigger level for the RX FIFO is reached. When this interrupt is included on the I/O, it appears on the outputs rx_da_*x*_intr (where *x* is the channel number). This interrupt is cleared by reading data from the RX FIFO until its level drops below the data available trigger level for the channel.
- Data Overrun interrupt – This interrupt is asserted when an attempt is made to write received data to a full RX FIFO (any data being written is lost while data in the FIFO is preserved). When this interrupt is included on the I/O, it appears on the outputs rx_or_*x*_intr (where *x* is the channel number). This interrupt is cleared by reading the Receive Channel Overrun (RXCHO) bit [0] of the Receive Overrun Register (RORx, where *x* is the channel number).

The interrupt status of any RX channel can be determined by polling the Interrupt Status Register (ISRx, where *x* is the channel number). The RXDA bit [0] indicates the status of the RX FIFO Data Available interrupt; the RXFO bit [1] indicates the status of the RX FIFO Data Overrun interrupt.

Both the Receive Empty Threshold and Data Overrun interrupts can be masked by writing 1 in the Receive Empty Threshold Mask (RDM) and Receive Overrun Mask (ROM) bits of the Interrupt Mask Register (IMRx, where *x* is the channel number), respectively. This prevents the interrupts from driving their output lines, however, the ISRx always shows the current status of the interrupts regardless of any masking.

The setting of the "Multiple Interrupt Output Ports Present?" configuration parameter (I2S_INTERRUPT_SIGNALS) affects whether these two interrupts are included on DW_apb_i2s's interface. Table 2-2 shows the specific interrupt signal to appear on the I/O according to the setting of I2S_INTERRUPT_SIGNALS

Table 2-2 RX Channel Interrupt Signals on I/O

Interrupt	Signal on I/O
I2S_INTERRUPT_SIGNALS = True (multiple interrupts on I/O)	
RX FIFO Data Available	rx_da_*x*_intr
Data Overrun	rx_or_*x*_intr
I2S_INTERRUPT_SIGNALS = False (shared single interrupt on I/O)	
combined interrupt signal	intr

For more information about the I2S_INTERRUPT_SIGNALS parameter, see "Parameter Descriptions" on page 55.

2.4.6 Reading from a Receive Channel

The stereo data pairs received by a RX channel are written to the left and right RX FIFOs. These FIFOs can be read through the Left Receive Buffer Register (LRBRx, where *x* is the channel number) and the Right Receive Buffer Register (RRBRx, where *x* is the channel number). All stereo data pairs must be read using the following two-stage process:

1. Read the left stereo data from LRBRx.
2. Read the right stereo data from RRBRx.

Note You must read the stereo data in this order to avoid the status and interrupt lines becoming out of sync.

When RX DMA is enabled (I2S_RX_DMA = 1), data can be read from RX FIFOs through the RXDMA register rather than through LRBR*x* and RRBR*x*. The RXDMA register cyclically accesses the RX FIFOs of all enabled RX channels similarly to the TXDMA register.

The RRXDMA register resets the RXDMA read cycle. This register provides the same functionality as the RTXDMA register, but targets RXDMA instead.

2.5 Clock Generation (Master Mode)

The clock generation block is only instantiated when the “Is an I2S Master?” parameter (I2S_MODE_EN) is checked (“True”). When DW_apb_i2s is a master, it initializes the word select signal (ws_out) and supplies the clock gating (sclk_gate) and clock enabling (sclk_en) signals. Additionally, the sclk and sclk_n inputs are included on the I/O regardless of whether the device is a master or a slave.

2.5.1 Clock Generation Enable

The Clock Generation Enable (CLKEN) bit of the Clock Enable Register (CER) enables and disables the master mode clock signals: ws_out, sclk_en, and sclk_gate. To enable these signals, set CER[0] to 1; to disable them, set this bit to 0, in which case ws_out is held low (ws_out = 0).

When the CLKEN bit is disabled, any incoming or outgoing data is lost. However, data already in the RX and TX FIFOs are preserved. After this bit is enabled, transmission recommences at the start of the next stereo frame.

On enabling CER[0], ws_out always starts in the left stereo data cycle (ws_out = 0). One sclk cycle later, it transitions to the right stereo data cycle (ws_out = 1); hence—0-to-1 transition. This allows for half a frame of sclks to write data to the TX FIFOs and ensures that any connected slave receivers do not miss the start of the data frame (the ws 1-to-0 transition) once the sclk restarts.

To further explain this behavior, the ws transitions—0-to-1 and 1-to-0—are used by the device to clock the start of the right or left stereo data cycle. The transition 1-to-0 indicates the start of a new stereo data pair. Because DW_apb_i2s is simple in terms of control, for every 1-to-0 transition, the device sends the next entry in its FIFO (similarly, the Receiver assumes new data is being sent and starts receiving). On enabling CER[0], the device starts with ws = 0 for one cycle and then transitions 0-to-1. This allows connected devices to clock off the transition and determine which cycle they are in. However, because it is not 1-to-0 transition, the devices do not have to start TX or RX with potential garbage (assuming FIFOs are empty prior to enable). Additionally, the transmission ensures that after a clk_en, there is ample time to configure TX or RX and to input some data for transmission before the start of the next frame.

The signal sclk_en is provided that can be AND'd with sclk to disable the clock when the master device is disabled.

Note The sclk_en output cannot be used to gate the sclk inputs to the DW_apb_i2s master instance that is driving sclk_en. It can only be used to gate the sclk inputs to a slave I^2S device.

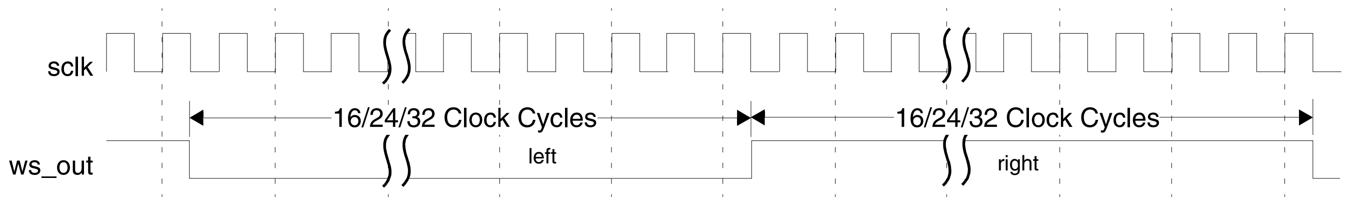
2.5.2 Word Select Generation

When DW_apb_i2s is configured as a master, the number of sclk cycles during which ws_out is held high or low is determined by the "Has a Word Select Length of?" parameter (I2S_WS_LENGTH). However, this setting can be reprogrammed during operation of the component by setting the Word Select Size (WSS) bits [4:3] of the Clock Configuration Register (CCR). You must disable the Clock Generation block (CER[0] = 0) before you can change the word select size.

Note The number of sclk cycles should be equal to or greater than the I2S_RX_WORDSIZE_*x*/I2S_TX_WORDSIZE_*x* parameter setting of the configured channel to prevent data loss. If this is not observed, the least significant bits of the transmission data and/or the received data are truncated.

The DW_apb_i2s supports 16, 24, or 32 sclk cycles per left/right ws cycle as illustrated in Figure 2-4.

Figure 2-4 Number of SCLK Cycles Per Left/Right WS Cycle

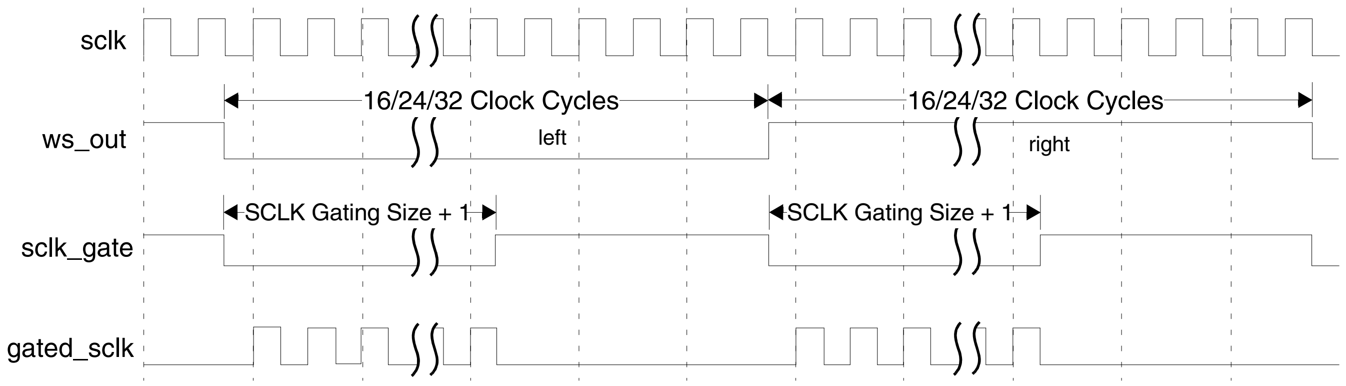


2.5.3 SCLK Gating

When DW_apb_i2s is configured as a master and the audio data resolution of the receive and transmit channels is less than the current word select size, the sclk can be gated off for the remainder of the left/right cycle, as illustrated in Figure 2-5. This gating is determined by the "Has a Serial Clock Gating of?"

parameter (I2S_SCLK_GATE). However, this can be reprogrammed during operation of the component by setting the SCLK Gating (SCLKG) bits [2:0] of the Clock Configuration Register (CCR).

Figure 2-5 Gating of SCLK



The Clock Generation Block must be disabled prior to any changes to sclk gating value. Since sclk_gate is 1 during the cycles that are to be gated off, the actual gating of sclk needs to be done externally by AND'ing the inverse of the generated sclk_gate signal with sclk.

Note The sclk_gate output cannot be used to gate the sclk inputs to the DW_apb_i2s master instance that is driving sclk_gate. It can only be used to gate the sclk inputs to a slave I^{2}S device.

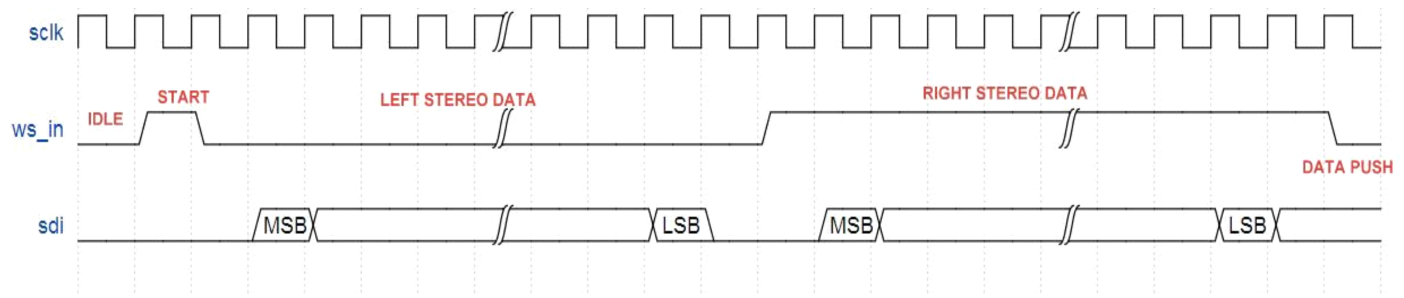
2.6 Transaction Example

In DW_apb_i2s, serial data is transmitted in two's complement format with the most significant bit (MSB) first. This means that the transmitter and receiver can have different word lengths, and neither the transmitter nor receiver needs to know what size words the other can handle. If the word being transferred is too large for the receiver, the least significant bits (LSB) are truncated. Similarly, if the word size is less than what the receiver can handle, the data is zero padded.

The word select line is used to time the multiplexed data streams. For instance, when ws is low, the word being transferred is left stereo data; when ws is high, the word being transferred is right stereo data. For standard I^{2}S formats, the MSB of a word is sent one sclk cycle after a ws change. Serial data sent by the transmitter can be synchronized with either the negative edge or positive edge of the sclk signal. However, the receiver must latch the serial data on the rising edge of sclk.

illustrates an example I^{2}S transfer in which DW_apb_i2s is a slave. The IDLE state of Word Select line is 0. Whenever the WS line makes a transition to 1, it means that after the next transition (0->1), the data starts being received. Therefore, the DW_apb_i2s slave treats the transfer as a START condition. When the stereo data is completely latched (signaled by Word Select Line going 0 again, also treated as start of new data frame), the data is pushed into the internal FIFO.

Figure 2-6 I^{2}S Transaction Example (DW_apb_i2s Slave)



2.7 APB Interface

The host processor accesses data, control, and status information on DW_apb_i2s through the APB interface. DW_apb_i2s supports APB data bus widths of 8, 16, and 32 bits.

For more information about the APB Interface and data widths, see "Integration Considerations" on page 159.

2.8 DW_apb_i2s Registers

2.8.1 Register Memory Map

An address definition (memory map) C header file is shipped with the DW_apb_i2s component. This header file is when the DW_apb_i2s is programmed in a C environment. "Register Descriptions" on page 81 provides details of the DW_apb_i2s memory map. The DW_apb_i2s component is little endian. Regardless of the APB bus width, aligning to 32-bit boundaries keeps the same memory map for all bus widths. The APB bus reset and presetn signals resets all registers. The base address of DW_apb_i2s is not fixed and is determined by the DW_apb component in the generation of the psel for DW_apb_i2s. The offset addresses from the base address are used for each register.

Note

- A read operation to an address location that contains unused bits results in 0 value being returned on each of the unused bits.
- The reset value for each register considers that all channels for both transmitter and receiver blocks and master mode have been configured.

For information about programming DW_apb_i2s using the software register described in this chapter, see "Programming the DW_apb_i2s" on page 147.

2.8.2 Coherency

When a register to be written or read is narrower than the data bus width a coherency logic is not required, and this logic is not implemented. It is possible for the Left Receive Buffer (LRBRx) and Right Receive Buffer (RRBRx) registers to be larger than the data bus width, therefore coherency logic maybe required when reading. It is also possible for the Left Transmit Holding (LTHRx) and Register Transmit Holding (RTHRx) registers to be larger than the data bus width, therefore coherency logic maybe required when writing to these registers. For a general discussion of coherency and the APB Interface, refer "Integration Considerations" on page 159.

2.9 DMA Handshaking Interface

2.9.1 DMA Controller Interface

The DW_apb_i2s has an optional built-in DMA capability that can be selected at configuration time; it has a handshaking interface to a DMA Controller to request and control transfers. The APB bus is used to perform the data transfer to or from the DMA. While the DW_apb_i2s DMA operation is designed in a generic way to fit any DMA controller as easily as possible, it is designed to work seamlessly, and best used, with the DesignWare DMA Controller, the DW_ahb_dmac. The settings of the DW_ahb_dmac that are relevant to the operation of the DW_apb_i2s are discussed here, mainly bit fields in the DW_ahb_dmac channel control register, CTL*x*, where *x* is the channel number.

Note When the DW_apb_i2s interfaces to the DW_ahb_dmac, the DW_ahb_dmac is always a flow controller; that is, it controls the block size. This must be programmed by software in the DW_ahb_dmac. The DW_ahb_dmac always transfers data using DMA burst transactions if possible, for efficiency. For more information, see the DW_ahb_dmac Databook. Other DMA controllers act in a similar manner.

DW_apb_i2s supports DMA handshake interface when it is configured with I2S_HAS_DMA_INTERFACE parameter set to 1. The I2S_DMA_HS_TYPE parameter selects DMA handshake interface type as described below. The I2S_DMA_HS_TYPE parameter is enabled only when the I2S_HAS_DMA_INTERFACE parameter is set to 1.

- I2S_DMA_HS_TYPE=0 – Dedicated DMA handshake interface for each transmit and receive channel.
- I2S_DMA_HS_TYPE=1 – Single DMA handshake interface for each transmitter block and receiver block; that is one DMA handshake interface for all transmit channels and one DMA handshake interface for all receive channels.

The relevant DMA settings are discussed in the following sections.

Note The DMA output dma_finish is a status signal to indicate that the DMA block transfer is complete. DW_apb_i2s does not use this status signal, and therefore does not appear in the I/O port list.

2.9.1.1 Dedicated DMA handshake Interface, I2S_DMA_HS_TYPE=0

When the parameter I2S_DMA_HS_TYPE is set to 0, DW_apb_i2s supports dedicated DMA handshake interface for each transmit and receive channel. For example, if DW_apb_i2s is configured with 4 transmit and 4 receive channels, then the DW_apb_i2s has 8 DMA handshaking interfaces.

2.9.1.1.1 Enabling the DMA Controller Interface on Transmit/Receive Channel

To enable the DMA Controller interface on transmit/receive channel of the DW_apb_i2s, you must write the DMA Control Register (DMACR).

- Writing 1 into the DMAEN_TXCH_x (x <= I2S_TX_CHANNELS-1) bit field of DMACR register, enables the DW_apb_i2s transmit handshaking interface on the transmitter channel x.

- Writing 1 into the DMAEN_RXCH_x (x <= I2S_RX_CHANNELS-1) bit field of the DMACR register, enables the DW_apb_i2s receive handshaking interface on the receiver channel x.

When dedicated DMA handshake Interface is enabled that is I2S_DMA_HS_TYPE=0, RXDMA_CHx and TXDMA_CHx registers are used for reading and writing of stereo data pairs. Cyclic channel access is not supported with dedicated DMA handshake Interface enabled. Therefore, RXDMA and TXDMA registers are not used if I2S_DMA_HS_TYPE=0.

It is not recommended to disable receiver or transmitter block in between of reading or writing of stereo data pairs. Otherwise, left or right stereo pairs might be transmitted out of sync. If the out of sync happens as receiver/transmitter block is disabled between reading or writing of stereo data pairs, receive or transmit channel FIFO must be flushed before enabling receiver or transmitter block again to ensure that left/right stereo pairs are transmitted in sync.

Table 2-3 provides description for different DMA transmit channel FIFO data level values.

Table 2-3 Transmit Channel x Threshold Decode Value (x <= I2S_TX_CHANNELS-1)

TFCRx.TXCHET Value	Description
0x0	dma_tx_req_x is asserted when 0 data entries are present in the transmit channel x FIFO
0x1	dma_tx_req_x is asserted when 1 or less data entries are present in the transmit channel x FIFO
0x2	dma_tx_req_x is asserted when 2 or less data entries are present in the transmit channel x FIFO
0x3	dma_tx_req_x is asserted when 3 or less data entries are present in the transmit channel x FIFO
0x4	dma_tx_req_x is asserted when 4 or less data entries are present in the transmit channel x FIFO
0x5	dma_tx_req_x is asserted when 5 or less data entries are present in the transmit channel x FIFO
0x6	dma_tx_req_x is asserted when 6 or less data entries are present in the transmit channel x FIFO
0x7	dma_tx_req_x is asserted when 7 or less data entries are present in the transmit channel x FIFO
0x8	dma_tx_req_x is asserted when 8 or less data entries are present in the transmit channel x FIFO
0x9	dma_tx_req_x is asserted when 9 or less data entries are present in the transmit channel x FIFO
0xa	dma_tx_req_x is asserted when 10 or less data entries are present in the transmit channel x FIFO
0xb	dma_tx_req_x is asserted when 11 or less data entries are present in the transmit channel x FIFO
0xc	dma_tx_req_x is asserted when 12 or less data entries are present in the transmit channel x FIFO
0xd	dma_tx_req_x is asserted when 13 or less data entries are present in the transmit channel x FIFO
0xe	dma_tx_req_x is asserted when 14 or less data entries are present in the transmit channel x FIFO
0xf	dma_tx_req_x is asserted when 15 or less data entries are present in the transmit channel x FIFO

Table 2-4 provides description for different DMA receive channel data level values.

Table 2-4 Receive Channel x Threshold Decode Value (x <= I2S_RX_CHANNELS-1)

RFCRx.RXCHET Value	Description
0x0	dma_rx_req_x is asserted when 1 or more data entries are present in the receive channel x FIFO
0x1	dma_rx_req_x is asserted when 2 or more data entries are present in the receive channel x FIFO
0x2	dma_rx_req_x is asserted when 3 or more data entries are present in the receive channel x FIFO
0x3	dma_rx_req_x is asserted when 4 or more data entries are present in the receive channel x FIFO
0x4	dma_rx_req_x is asserted when 5 or more data entries are present in the receive channel x FIFO
0x5	dma_rx_req_x is asserted when 6 or more data entries are present in the receive channel x FIFO
0x6	dma_rx_req_x is asserted when 7 or more data entries are present in the receive channel x FIFO
0x7	dma_rx_req_x is asserted when 8 or more data entries are present in the receive channel x FIFO
0x8	dma_rx_req_x is asserted when 9 or more data entries are present in the receive channel x FIFO
0x9	dma_rx_req_x is asserted when 10 or more data entries are present in the receive channel x FIFO
0xa	dma_rx_req_x is asserted when 11 or more data entries are present in the receive channel x FIFO
0xb	dma_rx_req_x is asserted when 12 or more data entries are present in the receive channel x FIFO
0xc	dma_rx_req_x is asserted when 13 or more data entries are present in the receive channel x FIFO
0xd	dma_rx_req_x is asserted when 14 or more data entries are present in the receive channel x FIFO
0xe	dma_rx_req_x is asserted when 15 or more data entries are present in the receive channel x FIFO
0xf	dma_rx_req_x is asserted when 16 data entries are present in the receive channel x FIFO

2.9.1.2 Single DMA Handshake Interface, I2S_DMA_HS_TYPE=1

When the I2S_DMA_HS_TYPE parameter is set to 1, DW_apb_i2s supports only one DMA handshake interface for each transmitter and receiver block. For example, if DW_apb_i2s is configured with 4 transmit and 4 receive channels, then there is 1 DMA handshake interface for transmit channels and 1 DMA handshake interface for receive channels.

In this mode of DMA operation, it is assumed that all transmit/receive channels are synchronous to each other.

2.9.1.2.1 Enabling the DMA Controller Interface on Transmitter/Receiver Block

To enable the DMA Controller interface on transmitter/receiver block of the DW_apb_i2s, you must write the DMA Control Register (DMACR).

- Writing 1 into the DMAEN_TXBLOCK bit field of DMACR register, enables the DMA handshaking interface on the transmitter block for all available transmit channels.
- Writing 1 into the DMAEN_RXBLOCK bit field of the DMACR register, enables the DMA handshaking interface on the receiver block for all available receive channels.

When I2S_DMA_HS_TYPE=1, RXDMA and TXDMA registers are used for reading and writing of stereo data pairs. The RXDMA/TXDMA register allows access to all enabled transmit or receive channels through a single point. The receive/transmit channels are targeted in a cyclical fashion (starting at the lowest numbered enabled channel) and takes two reads (left and right stereo data) before the component points to the next channel. For more information refer to the RXMDA and TXDMA registers in Chapter 5, "Register Descriptions" of DW_apb_i2s databook.

Table 2-5 provides description for different DMA transmit channel FIFO data level values.

Table 2-5 Transmit Channel x Threshold Decode Value (x <= I2S_TX_CHANNELS-1)

TFCRx.TXCHET Value	Description
0x0	dma_tx_req is asserted when 0 data entries are present in any of the transmit channel FIFO
0x1	dma_tx_req is asserted when 1 or less data entries are present in any of the transmit channel FIFO
0x2	dma_tx_req is asserted when 2 or less data entries are present in any of the transmit channel FIFO
0x3	dma_tx_req is asserted when 3 or less data entries are present in any of the transmit channel FIFO
0x4	dma_tx_req is asserted when 4 or less data entries are present in any of the transmit channel FIFO
0x5	dma_tx_req is asserted when 5 or less data entries are present in any of the transmit channel FIFO
0x6	dma_tx_req is asserted when 6 or less data entries are present in any of the transmit channel FIFO
0x7	dma_tx_req is asserted when 7 or less data entries are present in any of the transmit channel FIFO
0x8	dma_tx_req is asserted when 8 or less data entries are present in any of the transmit channel FIFO
0x9	dma_tx_req is asserted when 9 or less data entries are present in any of the transmit channel FIFO
0xa	dma_tx_req is asserted when 10 or less data entries are present in any of the transmit channel FIFO
0xb	dma_tx_req is asserted when 11 or less data entries are present in any of the transmit channel FIFO
0xc	dma_tx_req is asserted when 12 or less data entries are present in any of the transmit channel FIFO
0xd	dma_tx_req is asserted when 13 or less data entries are present in any of the transmit channel FIFO
0xe	dma_tx_req is asserted when 14 or less data entries are present in any of the transmit channel FIFO
0xf	dma_tx_req is asserted when 15 or less data entries are present in any of the transmit channel FIFO

Table 2-6 provides description for different DMA receive channel data level values.

Table 2-6 Receive Channel x Threshold Decode Value (x <= I2S_RX_CHANNELS-1)

2.9.1.3 Overview of Operation

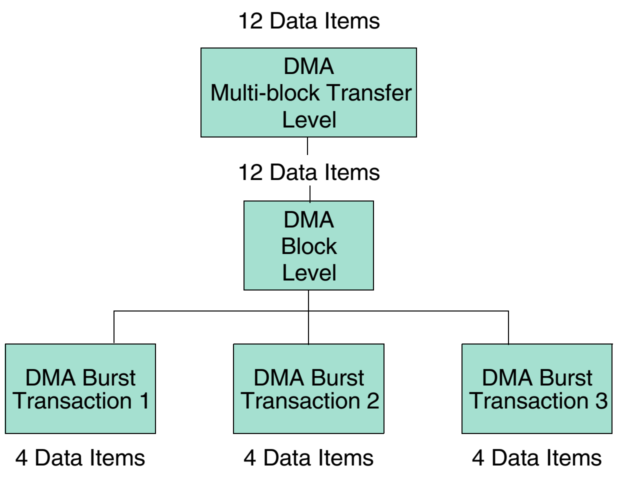
RFCRx.RXCHET Value	Description
0x0	dma_rx_req is asserted when 1 or more data entries are present in any of the receive channel FIFO
0x1	dma_rx_req is asserted when 2 or more data entries are present in any of the receive channel FIFO
0x2	dma_rx_req is asserted when 3 or more data entries are present in any of the receive channel FIFO
0x3	dma_rx_req is asserted when 4 or more data entries are present in any of the receive channel FIFO
0x4	dma_rx_req is asserted when 5 or more data entries are present in any of the receive channel FIFO
0x5	dma_rx_req is asserted when 6 or more data entries are present in any of the receive channel FIFO
0x6	dma_rx_req is asserted when 7 or more data entries are present in any of the receive channel FIFO
0x7	dma_rx_req is asserted when 8 or more data entries are present in any of the receive channel FIFO
0x8	dma_rx_req is asserted when 9 or more data entries are present in any of the receive channel FIFO
0x9	dma_rx_req is asserted when 10 or more data entries are present in any of the receive channel FIFO
0xa	dma_rx_req is asserted when 11 or more data entries are present in any of the receive channel FIFO
0xb	dma_rx_req is asserted when 12 or more data entries are present in any of the receive channel FIFO
0xc	dma_rx_req is asserted when 13 or more data entries are present in any of the receive channel FIFO
0xd	dma_rx_req is asserted when 14 or more data entries are present in any of the receive channel FIFO
0xe	dma_rx_req is asserted when 15 or more data entries are present in any of the receive channel FIFO
0xf	dma_rx_req is asserted when 16 data entries are present in any of the receive channel FIFO

As a block flow control device, the DMA Controller is programmed by the processor with the number of data items (block size) that are to be transmitted or received by DW_apb_i2s; this is programmed into the BLOCK_TS field of the DW_ahb_dmac CTL*x* register.

The block is broken into several transactions, each initiated by a request from the DW_apb_i2s. The DMA Controller must also be programmed with the number of data items (in this case, DW_apb_i2s FIFO entries) to be transferred for each DMA request. This is also known as the burst transaction length and is programmed into the SRC_MSIZE/DEST_MSIZE fields of the DW_ahb_dmac CTL*x* register for source and destination, respectively.

Figure 2-7 shows a single block transfer, where the block size programmed into the DMA Controller is 12 and the burst transaction length is set to 4. In this case, the block size is a multiple of the burst transaction length. Therefore, the DMA block transfer consists of a series of burst transactions. If the DW_apb_i2s makes a transmit request to this channel, four data items are written to the DW_apb_i2s TX FIFO. Similarly, if the DW_apb_i2s makes a receive request to this channel, four data items are read from the DW_apb_i2s RX FIFO. Three separate requests must be made to this DMA channel before all 12 data items are written or read.

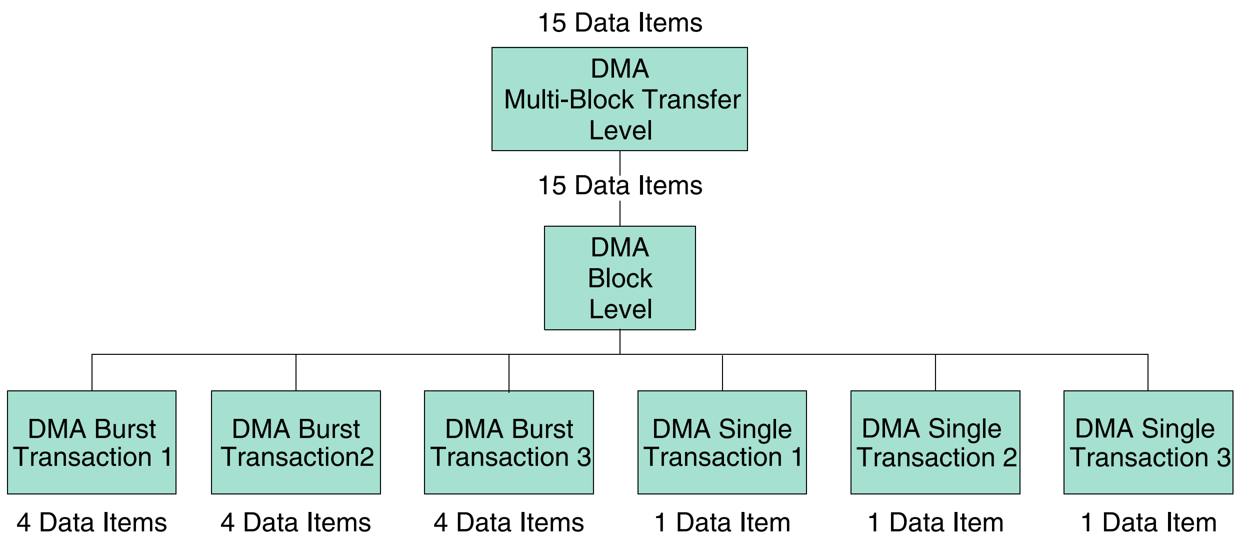
Figure 2-7 Breakdown of DMA Transfer into Burst Transactions



Block Size: DMA.CTLx.BLOCK_TS=12
Number of data items per source burst transaction: DMA.CTLx.SRC_MSIZE = 4
I^{2}S receive FIFO watermark level: I2S.RFCRx.RXCHET + 1 = DMA.CTLx.SRC_MSIZE = 4

When the block size programmed into the DMA Controller is not a multiple of the burst transaction length, as shown in Figure 2-8, a series of burst transactions followed by single transactions are needed to complete the block transfer.

Figure 2-8 Breakdown of DMA Transfer into Single and Burst Transactions



Block Size: DMA.CTLx.BLOCK_TS=15
Number of data items per burst transaction: DMA.CTLx.DEST_MSIZE = 4
I^{2}S transmit FIFO watermark level: I2S.TFCRx.TXCHET = DMA.CTLx.DEST_MSIZE = 4

2.9.1.4 Transmit Watermark Level and Transmit FIFO Underflow

During DW_apb_i2s serial transfers, transmit FIFO requests are made to the DW_ahb_dmac whenever the number of entries in the transmit FIFO is less than or equal to the DMA Transmit Data Level Register (TFCRx.TXCHET) value; this is known as the watermark level. The DW_ahb_dmac responds by writing a burst of data to the transmit FIFO buffer, of length CTL*x*.DEST_MSIZE.

Data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously; that is, when the FIFO begins to empty another DMA request should be triggered. Otherwise, the FIFO runs out of data causing a STOP to be inserted on the I^{2}S bus. To prevent this condition, you must set the watermark level correctly.

2.9.1.5 Choosing the Transmit Watermark Level

Consider the example where the assumption is made:

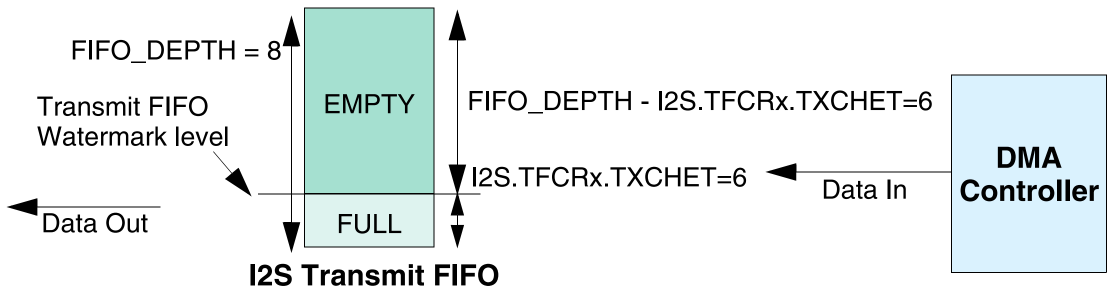
```
DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - I2S.TFCRx.TXCHET
```

Here the number of data items to be transferred in a DMA burst is equal to the empty space in the Transmit FIFO. Consider two different watermark level settings.

Case 1: I2S.TFCRx.TXCHET = 2

- Transmit FIFO watermark level = I2S.TFCRx.TXCHET = 2
- DMA.CTL*x*.DEST_MSIZE = FIFO_DEPTH - I2S.TFCRx.TXCHET = 6
- I^{2}S transmit FIFO_DEPTH = 8
- DMA.CTL*x*.BLOCK_TS = 30

Figure 2-9 Case 1 Watermark Levels



Therefore, the number of burst transactions needed equals the block size divided by the number of data items per burst:

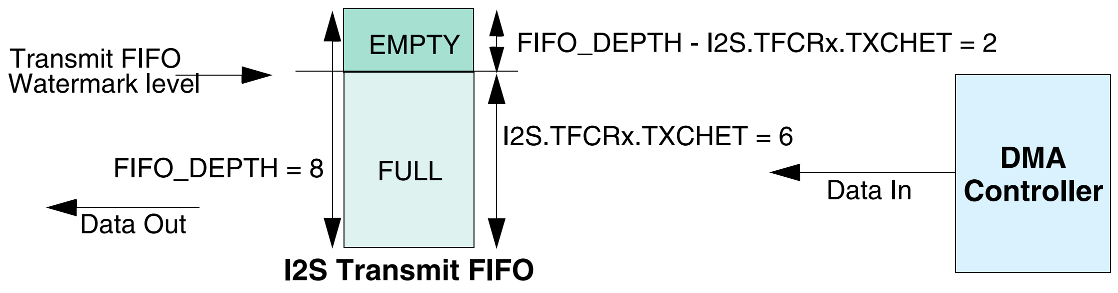
```
DMA.CTLx.BLOCK_TS/DMA.CTLx.DEST_MSIZE = 30/6 = 5
```

The number of burst transactions in the DMA block transfer is 5. But the watermark level, I2S.TFCRx.TXCHET, is quite low. Therefore, the probability of an I^{2}S underflow is high where the I^{2}S serial transmit line needs to transmit data, but where there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the transmit FIFO becomes empty.

Case 2: IC_DMA_TDLR = 6

- Transmit FIFO watermark level = I2S.TFCRx.TXCHET = 6
- DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - I2S.TFCRx.TXCHET = 2
- I^{2}S transmit FIFO_DEPTH = 8
- DMA.CTLx.BLOCK_TS = 30

Figure 2-10 Case 2 Watermark Levels



Number of burst transactions in Block:

```
DMA.CTLx.BLOCK_TS/DMA.CTLx.DEST_MSIZE = 30/2 = 15
```

In this block transfer, there are 15 destination burst transactions in a DMA block transfer. But the watermark level, I2S.TFCRx.TXCHET, is high. Therefore, the probability of an I^2S underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the I^2S transmit FIFO becomes empty.

Thus, the second case has a lower probability of underflow at the expense of more burst transactions per block. This provides a potentially greater amount of AMBA bursts per block and worse bus utilization than the former case.

Therefore, the goal in choosing a watermark level is to minimize the number of transactions per block, while at the same time keeping the probability of an underflow condition to an acceptable level. In practice, this is a function of the ratio of the rate at which the I^2S transmits data to the rate at which the DMA can respond to destination burst requests.

For example, promoting the channel to the highest priority channel in the DMA, and promoting the DMA master interface to the highest priority master in the AMBA layer, increases the rate at which the DMA controller can respond to burst transaction requests. This in turn allows you to decrease the watermark level, which improves bus utilization without compromising the probability of an underflow occurring.

2.9.1.6 Selecting DEST_MSIZE and Transmit FIFO Overflow

As described in Figure 2-10, programming DMA.CTL*x*.DEST_MSIZE to a value greater than the watermark level that triggers the DMA request may cause overflow when there is not enough space in the I^2S transmit FIFO to service the destination burst request. Therefore, the following equation must be adhered to in order to avoid overflow:

```
DMA.CTLx.DEST_MSIZE <= I2S.FIFO_DEPTH - I2S.TFCRx.TXCHET    (1)
```

In "Case 2: IC_DMA_TDLR = 6", the amount of space in the transmit FIFO at the time the burst request is made is equal to the destination burst length, DMA.CTL*x*.DEST_MSIZE. Thus, the transmit FIFO may be full, but not overflowed, at the completion of the burst transaction.

Therefore, for optimal operation, DMA.CTL*x*.DEST_MSIZE should be set at the FIFO level that triggers a transmit DMA request; that is:

```
DMA.CTLx.DEST_MSIZE = I2S.FIFO_DEPTH - I2S.TFCRx.TXCHET    (2)
```

This is the setting used in Figure 2-8.

Adhering to equation (2) reduces the number of DMA bursts needed for a block transfer, and this in turn improves AMBA bus utilization.

Note The transmit FIFO is not be full at the end of a DMA burst transfer if the I^2S has successfully transmitted one data item or more on the I^2S serial transmit line during the transfer.

2.9.1.7 Receive Watermark Level and Receive FIFO Overflow

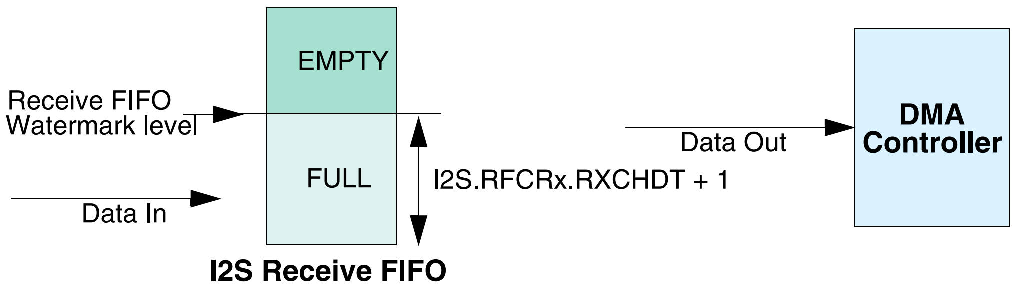
During DW_apb_i2s serial transfers, receive FIFO requests are made to the DW_ahb_dmac whenever the number of entries in the receive FIFO is at or above the DMA Receive Data Level Register; that is, RFCRx.RXCHDT+1. This is known as the watermark level. The DW_ahb_dmac responds by fetching a burst of data from the receive FIFO buffer of length CTLx.SRC_MSIZE.

Data should be fetched by the DMA often enough for the receive FIFO to accept serial transfers continuously; that is, when the FIFO begins to fill, another DMA transfer is requested. Otherwise, the FIFO fills with data (overflow). To prevent this condition, you must correctly set the watermark level.

2.9.1.8 Choosing the Receive Watermark level

Similar to choosing the transmit watermark level described earlier, the receive watermark level, RFCRx.RXCHDT+1, should be set to minimize the probability of overflow, as shown in Figure 2-11. It is a trade-off between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

Figure 2-11 I^{2}S Receive FIFO



2.9.1.9 Selecting SRC_MSIZE and Receive FIFO Underflow

As described in Figure 2-11, programming a source burst transaction length greater than the watermark level may cause underflow when there is not enough data to service the source burst request. Therefore, equation 3 following must be adhered to avoid underflow.

If the number of data items in the receive FIFO is equal to the source burst length at the time the burst request is made – DMA.CTL*x*.SRC_MSIZE – the receive FIFO may be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA.CTL*x*.SRC_MSIZE should be set at the watermark level; that is:

```
DMA.CTLx.SRC_MSIZE = I2S.RFCRx.RXCHDT + 1    (3)
```

Adhering to equation (3) reduces the number of DMA bursts in a block transfer, which in turn can avoid underflow and improve AMBA bus utilization.

Note The receive FIFO is not empty at the end of the source burst transaction if the I^{2}S has successfully received one data item or more on the I^{2}S serial receive line during the burst.

2.9.1.10 Handshaking Interface Operation

The following sections discuss the handshaking interface.

- If DW_apb_i2s is configured with dedicated type of DMA operation (i.e. I2S_DMA_HS_TYPE=0), then dedicated DMA handshaking interfaces are present for each receive and transmit channels. In this mode, DMA handshake signals are represented as dma_rx_req_x(_n)/dma_tx_req_x(_n), dma_rx_single_x(_n)/dma_tx_single_x(_n) and dma_rx_ack_x(_n)/dma_tx_ack_x(_n) - where, x <= I2S_RX_CHANNELS-1 or x <= I2S_TX_CHANNELS-1.
- If DW_apb_i2s is configured with combined type of DMA operation (i.e. I2S_DMA_HS_TYPE=1), then there is only one DMA handshake interface for each transmitter and receiver block. In this mode, DMA handshake signals are represented as dma_rx_req(_n)/dma_tx_req(_n), dma_rx_single(_n)/dma_tx_single(_n) and dma_rx_ack(_n)/dma_tx_ack(_n).

Polarity of DMA handshake signals is determined by the parameter I2S_DMA_POL. For example, if DW_apb_i2s is configured with dedicated DMA handshake interface type (I2S_DMA_HS_TYPE=0) and active High polarity (I2S_DMA_POL=1), then following signals are present on interface: dma_rx_req_x/dma_tx_req_x, dma_rx_single_x/dma_tx_single_x and dma_rx_ack_x/dma_tx_ack_x. If polarity is configured as active Low (I2S_DMA_POL=0) then following signals are present on interface: dma_rx_req_x(_n)/dma_tx_req_x(_n), dma_rx_single_x(_n)/dma_tx_single_x(_n) and dma_rx_ack_x(_n)/dma_tx_ack_x(_n).

The handshaking interface operation is described for active high signals. Similar argument follows for active Low signals, if it is configured with I2S_DMA_POL set to 0.

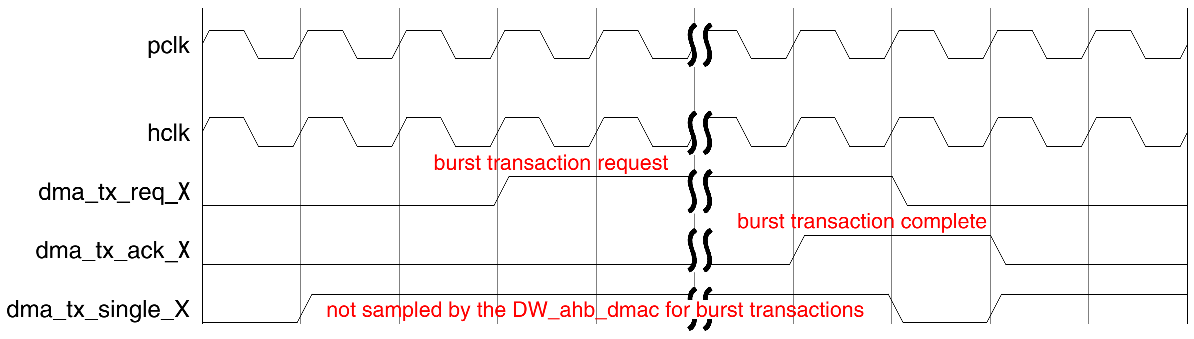
2.9.1.10.1 dma_tx_req(_x), dma_rx_req(_x)

The request signals for source and destination, dma_tx_req(_x) and dma_rx_req(_x), are activated when their corresponding FIFOs reach the watermark levels as discussed earlier.

The DW_ahb_dmac uses rising-edge detection of the dma_tx_req(_x) signal/dma_rx_req(_x) to identify a request on the channel. Upon reception of the dma_tx_ack(_x)/dma_rx_ack(_x) signal from the DW_ahb_dmac to indicate the burst transaction is complete, the DW_apb_i2s de-asserts the burst request signals, dma_tx_req/dma_rx_req, until dma_tx_ack(_x)/dma_rx_ack(_x) is de-asserted by the DW_ahb_dmac.

When the DW_apb_i2s samples that dma_tx_ack(_x)/dma_rx_ack(_x) is de-asserted, it can re-assert the dma_tx_req(_x)/dma_rx_req(_x) of the request line if their corresponding FIFOs exceed their watermark levels (back-to-back burst transaction). If this is not the case, the DMA request lines remain de-asserted. Figure 2-12 shows a timing diagram of a burst transaction where pclk = hclk.

Figure 2-12 Burst Transaction – pclk = hclk


pclk
hclk
dma_tx_req_X
dma_tx_ack_X
dma_tx_single_X
burst transaction request
burst transaction complete
not sampled by the DW_ahb_dmac for burst transactions

The handshaking loop is as follows:

- dma_tx_req(_x)/dma_rx_req(_x) asserted by DW_apb_i2s
- dma_tx_ack(_x)/dma_rx_ack(_x) asserted by DW_ahb_dmac
- dma_tx_req(_x)/dma_rx_req(_x) de-asserted by DW_apb_i2s
- dma_tx_ack(_x)/dma_rx_ack(_x) de-asserted by DW_ahb_dmac
- dma_tx_req(_x)/dma_rx_req(_x) reasserted by DW_apb_i2s, if back-to-back transaction is required

Note The burst transaction request signals, dma_tx_req(_x) and dma_rx_req(_x), are generated in the DW_apb_i2s off pclk and sampled in the DW_ahb_dmac by hclk. The acknowledge signals, dma_tx_ack(_x) and dma_rx_ack(_x), are generated in the DW_ahb_dmac off hclk and sampled in the DW_apb_i2s of pclk. The handshaking mechanism between the DW_ahb_dmac and the DW_apb_i2s supports quasi-synchronous clocks; that is, hclk and pclk must be phase-aligned, and the hclk frequency must be a multiple of the pclk frequency.

Note the following:

- The burst request lines, dma_tx_req signal(_x)/dma_rx_req(_x), once asserted remain asserted until their corresponding dma_tx_ack(_x)/dma_rx_ack(_x) signal is received even if the respective FIFO's drop below their watermark levels during the burst transaction.
- The dma_tx_req(_x)/dma_rx_req(_x) signals are de-asserted when their corresponding dma_tx_ack(_x)/dma_rx_ack(_x) signals are asserted, even if the respective FIFOs exceed their watermark levels.

2.9.1.10.2 dma_tx_req_single(_x), dma_rx_req_single(_x)

The dma_tx_single(_x) signal is asserted when there is at least one free entry in the transmit FIFO, and is cleared when the dma_tx_ack(_x) signal is active. The dma_tx_single(_x) signal is re-asserted when the dma_tx_ack(_x) signal is de-asserted, if the condition for setting still holds true.

The dma_rx_single(_x) signal is asserted when there is at least one valid data entry in the receive FIFO, and is cleared when the dma_rx_ack(_x) signal is active. The dma_rx_single(_x) signal is re-asserted when the dma_rx_ack(_x) signal is de-asserted, if the condition for setting still holds true.

These signals are needed by only the DW_ahb_dmac for the case where the block size, CTL*x*.BLOCK_TS, that is programmed into the DW_ahb_dmac is not a multiple of the burst transaction length, CTL*x*.SRC_MSIZE, CTL*x*.DEST_MSIZE, as shown in Figure 2-8. In this case, the DMA single outputs inform the DW_ahb_dmac that it is still possible to perform single data item transfers, so it can access all data items in the transmit/receive FIFO and complete the DMA block transfer. The DMA single outputs from the DW_apb_i2s are not sampled by the DW_ahb_dmac otherwise. This is illustrated in the following example.

Consider first an example where the receive FIFO channel of the DW_apb_i2s is as follows:

```
DMA.CTLx.SRC_MSIZE = I2S.RFCRx.RXCHDT + 1 = 4
DMA.CTLx.BLOCK_TS = 12
```

For example in Figure 2-7, with the block size set to 12, the dma_rx_req signal is asserted when four data items are present in the receive FIFO. The dma_rx_req signal is asserted three times during the DW_apb_i2s serial transfer, ensuring that all 12 data items are read by the DW_ahb_dmac. All DMA requests read a block of data items and no single DMA transactions are required. This block transfer is made up of three burst transactions.

Now, for the following block transfer:

```
DMA.CTLx.SRC_MSIZE = I2S.RFCRx.RXCHDT + 1 = 4
DMA.CTLx.BLOCK_TS = 15
```

The first 12 data items are transferred as already described using three burst transactions. But when the last three data frames enter the receive FIFO, the dma_rx_req(_x) signal is not activated because the FIFO level is below the watermark level. The DW_ahb_dmac samples dma_rx_single(_x) and completes the DMA block transfer using three single transactions. The block transfer is made up of three burst transactions followed by three single transactions.

Figure 2-13 shows a single transaction. The handshaking loop is as follows:

- dma_tx_single(_x)/dma_rx_single(_x) asserted by DW_apb_i2s
- dma_tx_ack(_x)/dma_rx_ack(_x) asserted by DW_ahb_dmac
- dma_tx_single(_x)/dma_rx_single(_x) de-asserted by DW_apb_i2s
- dma_tx_ack(_x)/dma_rx_ack(_x) de-asserted by DW_ahb_dmac

Figure 2-13 Single Transaction

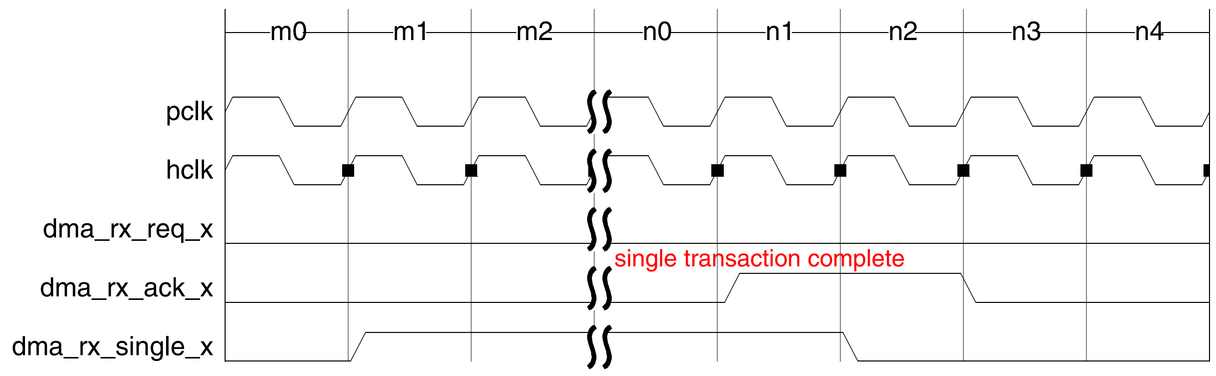
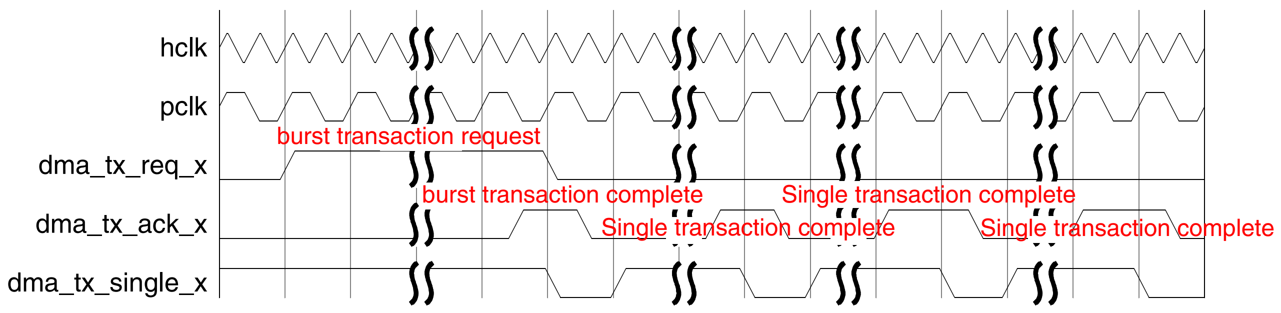


Figure 2-14 shows a burst transaction, followed by three back-to-back single transactions, where the hclk frequency is twice the pclk frequency.

Figure 2-14 Burst Transaction + 3 Back-to-Back Singles – hclk = 2*pclk


hclk
pclk
dma_tx_req_x
dma_tx_ack_x
dma_tx_single_x
burst transaction request
burst transaction complete
Single transaction complete
Single transaction complete
Single transaction complete

Note The single transaction request signals, dma_tx_single(_x) and dma_rx_single(_x), are generated in the DW_apb_i2s on the pclk edge and sampled in DW_ahb_dmac on hclk. The acknowledge signals, dma_tx_ack(_x) and dma_rx_ack(_x), are generated in the DW_ahb_dmac on the hclk edge hclk and sampled in the DW_apb_i2s on pclk. The handshaking mechanism between the DW_ahb_dmac and the DW_apb_i2s supports quasi-synchronous clocks; that is, hclk and pclk must be phase aligned and the hclk frequency must be a multiple of pclk frequency.

3 Parameter Descriptions

This chapter details all the configuration parameters. **You can use the coreConsultant GUI configuration reports to determine the actual configured state of the controller.** Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

The parameter descriptions in this chapter include the **Enabled:** attribute which indicates the values required to be set on other parameters before you can change the value of this parameter.

These tables define all of the user configuration options for this component.

- Basic Configuration on page 56
- Receiver Channel(s) on page 61
- Transmitter Channel(s) on page 63

3.1 Basic Configuration Parameters

Table 3-1 Basic Configuration Parameters

Label	Description
	General Configuration Settings
APB Data Bus Width?	Width of APB data bus to which this component is attached. **Values:** 8, 16, 32 **Default Value:** 32 **Enabled:** Always **Parameter Name:** APB_DATA_WIDTH
Receiver Block Enabled?	Controls whether the DW_apb_i2s component has I2S receiver block(s) or not. This must be enabled to be able to set the number of RX channels (I2S_RX_CHANNELS). For more information about the operation of the component when it is a receiver, refer to "DW_apb_i2s as Receiver". You can also program the enabling/disabling of this block during operation by setting bit 0 of the I2S Receiver Block Enable Register (IRER). **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** I2S_RECEIVER_BLOCK
Number of Receive Channels?	Controls the number of receive channels for this DW_apb_i2s component. For more information about enabling/disabling individual receive channels during operation, refer to "Receive Channel Enable". **Values:** 1, 2, 3, 4 **Default Value:** 1 **Enabled:** I2S_RECEIVER_BLOCK==1 **Parameter Name:** I2S_RX_CHANNELS
Transmitter Block Enabled?	Controls whether the DW_apb_i2s component has I2S transmitter block(s). This must be enabled in order to be able to set the number of TX channels (I2S_TX_CHANNELS). For more information about the operation of the component when it is a transmitter, refer to "DW_apb_i2s as Transmitter". You can also program the enabling/disabling of this block during operation by setting bit 0 of the I2S Transmitter Block Enable Register (ITER). **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** I2S_TRANSMITTER_BLOCK

Table 3-1 Basic Configuration Parameters (Continued)

Label	Description
Number of Transmit Channels?	Controls the number of transmit channels for this DW_apb_i2s component. For more information about enabling/disabling individual transmit channels during operation, refer to "Writing to a Transmit Channel". **Values:** 1, 2, 3, 4 **Default Value:** 1 **Enabled:** I2S_TRANSMITTER_BLOCK==1 **Parameter Name:** I2S_TX_CHANNELS
Is an I2S Master?	Determines whether the component acts as the I2S master or slave. For more information about clock generation, which can only occur when the component is a master, refer to "Clock Generation (Master Mode)". **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** I2S_MODE_EN
FIFO Depth for RX and TX Channels?	Determines the FIFO depth for all channels. Both the RX and TX channels have the same depth. This is used to set both the I2S_RX_FIFO_x and I2S_TX_FIFO_x values of all the channels chosen. For more information about the RX and TX channel FIFOs, refer to "Transmit Channel FIFOs" and "Receive Channel FIFOs" respectively. **Values:** 2, 4, 8, 16 **Default Value:** 8 **Enabled:** I2S_RECEIVER_BLOCK>=1 || I2S_TRANSMITTER_BLOCK>=1 **Parameter Name:** I2S_FIFO_DEPTH_GLOBAL
Master Clk/Slave Clk Settings	
Has a Word Select Length of?	Sets the default number of sclk cycles for which the Word Select Line is held high or low. For more information about the ws line, refer to "Word Select Generation". **Values:** ■ 16 (0x0) ■ 24 (0x1) ■ 32 (0x2) **Default Value:** 16 **Enabled:** I2S_MODE_EN!=0 **Parameter Name:** I2S_WS_LENGTH

Table 3-1 Basic Configuration Parameters (Continued)

Label	Description
Has a Serial Clock Gating of?	Selects the default type of clock gating used on the sclk output. For more information about this feature, refer to "SCLK Gating". **Values:** ■ No Gating (0x0) ■ 12 Clock Cycles (0x1) ■ 16 Clock Cycles (0x2) ■ 20 Clock Cycles (0x3) ■ 24 Clock Cycles (0x4) **Default Value:** No Gating **Enabled:** I2S_MODE_EN!=0 **Parameter Name:** I2S_SCLK_GATE
Interrupt Settings	
Multiple Interrupt Output Ports Present?	Determines whether the component has multiple individual interrupt outputs (True) or a single global interrupt output (False). For more information about these signals, refer to "Signal Descriptions". For more information about when these interrupts are generated, refer to "Transmit Channel Interrupts" and "Receive Channel Interrupts". **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** I2S_INTERRUPT_SIGNALS
Polarity of Interrupt Signals is Active High?	Sets the polarity of the interrupt signals. For more information about the interrupt signals, refer to "Signal Descriptions", "Transmit Channel Interrupts" and "Receive Channel Interrupts". **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** I2S_INTR_POL

Table 3-1 Basic Configuration Parameters (Continued)

Label	Description
Clock Domain Crossing Settings	
Clock Domain Crossing Synchronisation Depth?	Sets the number of synchronisation register stages used when crossing between clock domains. These are required to avoid metastability issues between clock domains. ■ 1 => Two-stage synchronisation, first stage negative edge. second stage positive edge. ■ 2 => Two-stage synchronisation, both stages positive edge. ■ 3 => Three-stage synchronisation, all stages positive edge. **Values:** 1, 2, 3 **Default Value:** 2 **Enabled:** Always **Parameter Name:** I2S_SYNC_DEPTH
DMA Handhshake Interface Settings	
Has DMA Controller Interface?	Selects if DW_apb_i2s requires DMA handshaking interface. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** (I2S_RECEIVER_BLOCK==1) ? ((I2S_TRANSMITTER_BLOCK==1) ? (APB_DATA_WIDTH>=I2S_RX_WORDSIZE_0 && APB_DATA_WIDTH>=I2S_RX_WORDSIZE_1 && APB_DATA_WIDTH >= I2S_RX_WORDSIZE_2 && APB_DATA_WIDTH >= I2S_RX_WORDSIZE_3) && (APB_DATA_WIDTH>=I2S_TX_WORDSIZE_0 && APB_DATA_WIDTH>=I2S_TX_WORDSIZE_1 && APB_DATA_WIDTH >= I2S_TX_WORDSIZE_2 && APB_DATA_WIDTH >= I2S_TX_WORDSIZE_3) : (APB_DATA_WIDTH>=I2S_RX_WORDSIZE_0 && APB_DATA_WIDTH>=I2S_RX_WORDSIZE_1 && APB_DATA_WIDTH >= I2S_RX_WORDSIZE_2 && APB_DATA_WIDTH >= I2S_RX_WORDSIZE_3)) : ((I2S_TRANSMITTER_BLOCK==1) ? (APB_DATA_WIDTH>=I2S_TX_WORDSIZE_0 && APB_DATA_WIDTH>=I2S_TX_WORDSIZE_1 && APB_DATA_WIDTH >= I2S_TX_WORDSIZE_2 && APB_DATA_WIDTH >= I2S_TX_WORDSIZE_3) : 0) **Parameter Name:** I2S_HAS_DMA_INTERFACE

Table 3-1 Basic Configuration Parameters (Continued)

Label	Description
DMA handshake interface type?	Selects the DMA handshake interface type: 0 (DEDICATED) - Dedicated DMA handshake interface for each transmit and receive channels. 1 (COMBINED) - Single DMA handshake interface for each transmitter and receiver block. **Values:** ■ DEDICATED (0) ■ COMBINED (1) **Default Value:** COMBINED **Enabled:** I2S_HAS_DMA_INTERFACE==1 **Parameter Name:** I2S_DMA_HS_TYPE
Polarity of DMA Interface Signals is Active High?	Sets the polarity of the DMA Interface signals. For more information about the DMA interface signals, refer to "DMA Interface Signals" in "Signals" chapter of DW_apb_i2s databook. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** I2S_HAS_DMA_INTERFACE==1 **Parameter Name:** I2S_DMA_POL

3.2 Receiver Channel(s) Parameters

Table 3-2 Receiver Channel(s) Parameters

Label	Description
	Receiver DMA
Receiver Block DMA Enabled?	Controls whether the DW_apb_i2s component has a DMA register for I2S RX Channels. **Values:** ■ false (0) ■ true (1) **Default Value:** I2S_RECEIVER_BLOCK==1 && I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==1 **Enabled:** I2S_RECEIVER_BLOCK==1 && APB_DATA_WIDTH>=I2S_RX_WORDSIZE_0 && APB_DATA_WIDTH>=I2S_RX_WORDSIZE_1 && APB_DATA_WIDTH >= I2S_RX_WORDSIZE_2 && APB_DATA_WIDTH >= I2S_RX_WORDSIZE_3 && I2S_HAS_DMA_INTERFACE==0 **Parameter Name:** I2S_RX_DMA
	Receiver Channel x
Max Audio Resolution - Receive Channel x (for x = 0; x <= 3)	Sets the maximum audio data resolution (word size) of the left and right data for Receive Channel x. For more information about this feature, refer to "Receive Channel Audio Data Resolution". **Values:** 12, 16, 20, 24, 32 **Default Value:** 16 **Enabled:** I2S_RECEIVER_BLOCK==1 && I2S_RX_CHANNELS>=x **Parameter Name:** I2S_RX_WORDSIZE_x
FIFO Depth - Receive Channel x (for x = 0; x <= 3)	Determines the FIFO depth for both the left and right RX_FIFOs for Receive Channel x. **Values:** 2, 4, 8, 16 **Default Value:** I2S_FIFO_DEPTH_GLOBAL **Enabled:** This is not selectable as it is set by the global FIFO depth value, I2S_FIFO_DEPTH_GLOBAL. **Parameter Name:** I2S_RX_FIFO_x

Table 3-2 Receiver Channel(s) Parameters (Continued)

Label	Description
RX FIFO Data Available Trigger Level - Receive Channel x (for x = 0; x <= 3)	Sets the level at which the data available signal for the Receive Channel x is generated. Data Available Trigger Level = Selected Value + 1 This parameter is only available when device is configured as a receiver (I2S_RECEIVER_BLOCK = 1) and must be set to a value less than the channel FIFO depth (I2S_RX_FIFO_x - 1). For more information about this interrupt, refer to "Receive Channel Interrupts". **Values:** 0, ..., 15 **Default Value:** ((I2S_RECEIVER_BLOCK == 1 && I2S_RX_CHANNELS>=x) ? 3 : 0) **Enabled:** I2S_RECEIVER_BLOCK==1 && I2S_RX_CHANNELS>=x **Parameter Name:** I2S_RX_FIFO_THRE_x

3.3 Transmitter Channel(s) Parameters

Table 3-3 Transmitter Channel(s) Parameters

Label	Description
	Transmitter DMA
Transmitter Block DMA Enabled?	Controls whether the DW_apb_i2s component has a DMA register for the I2S TX Channels or not. **Values:** ■ false (0) ■ true (1) **Default Value:** I2S_TRANSMITTER_BLOCK==1 && I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==1 **Enabled:** I2S_TRANSMITTER_BLOCK==1 && APB_DATA_WIDTH>=I2S_TX_WORDSIZE_0 && APB_DATA_WIDTH>=I2S_TX_WORDSIZE_1 && APB_DATA_WIDTH >= I2S_TX_WORDSIZE_2 && APB_DATA_WIDTH >= I2S_TX_WORDSIZE_3 && I2S_HAS_DMA_INTERFACE==0 **Parameter Name:** I2S_TX_DMA
	Transmitter Channel x
Max Audio Resolution - Transmit Channel x (for x = 0; x <= 3)	Sets the maximum audio data resolution (word size) of the left and right data for Transmit Channel x. For more information about this feature, refer to "Transmit Channel Audio Data Resolution". **Values:** 12, 16, 20, 24, 32 **Default Value:** 16 **Enabled:** I2S_TRANSMITTER_BLOCK==1 && I2S_TX_CHANNELS>=x **Parameter Name:** I2S_TX_WORDSIZE_x
FIFO Depth - Transmit Channel x (for x = 0; x <= 3)	Determines the FIFO depth for both the left and right TX_FIFOs for Transmitter Channel x. **Values:** 2, 4, 8, 16 **Default Value:** I2S_FIFO_DEPTH_GLOBAL **Enabled:** This is not selectable as it is set by the global FIFO depth value, I2S_FIFO_DEPTH_GLOBAL. **Parameter Name:** I2S_TX_FIFO_x

Table 3-3 Transmitter Channel(s) Parameters (Continued)

Label	Description
TX FIFO Empty Threshold Trigger Level - Transmit Channel x (for x = 0; x <= 3)	Set the level at which the empty threshold reached signal for Transmit Channel x is generated. This is only selectable when device is configured as a transmitter (I2S_TRANSMITTER_BLOCK==1) and It must be set to a value less than the channel's FIFO depth (I2S_TX_FIFO_0-1). For more information about this interrupt, refer to "Transmit Channel Interrupts". **Values:** 0, ..., 15 **Default Value:** ((I2S_TRANSMITTER_BLOCK==1 && I2S_TX_CHANNELS>=x) ? 3 : 0) **Enabled:** I2S_TRANSMITTER_BLOCK==1 && I2S_TX_CHANNELS>=x **Parameter Name:** I2S_TX_FIFO_THRE_x

4 Signal Descriptions

This chapter details all possible I/O signals in the controller. For configurable IP titles, your actual configuration might not contain all of these signals.

Inputs are on the left of the signal diagrams; outputs are on the right.

Attention: For configurable IP titles, do not use this document to determine the exact I/O footprint of the controller. It is for reference purposes only.

When you configure the controller in coreConsultant, you must access the I/O signals for your actual configuration at workspace/report/IO.html or workspace/report/IO.xml after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the I/O signals that are not in your actual configuration. This does not apply to non-configurable IP titles. In addition, all parameter expressions are evaluated to actual values. Therefore, the widths might change depending on your actual configuration.

Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

In addition to describing the function of each signal, the signal descriptions in this chapter include the following information:

Active State: Indicates whether the signal is active high or active low. When a signal is not intended to be used in a particular application, then this signal needs to be tied or driven to the inactive state (opposite of the active state).

Registered: Indicates whether or not the signal is registered directly inside the IP boundary without intervening logic (excluding simple buffers). A value of No does not imply that the signal is not synchronous, only that there is some combinatorial logic between the signal's origin or destination register and the boundary of the controller. A value of N/A indicates that this information is not provided for this IP title.

Synchronous to: Indicates which clock(s) in the IP sample this input (drive for an output) when considering all possible configurations. A particular configuration might not have all of the clocks listed. This clock might not be the same as the clock that your application logic should use to clock (sample/drive) this pin. For more details, consult the clock section in the databook.

Exists: Name of configuration parameter(s) that populates this signal in your configuration.

Validated by: Assertion or de-assertion of signal(s) that validates the signal being described.

Attributes used with Synchronous To

- Clock name - The name of the clock that samples an input or drive and output.
- None - This attribute may be used for clock inputs, hard-coded outputs, feed-through (direct or combinatorial), dangling inputs, unused inputs and asynchronous outputs.
- Asynchronous - This attribute is used for asynchronous inputs and asynchronous resets.

The I/O signals are grouped as follows:

- APB Slave Interface on page 67
- I2S Clock Interface on page 69
- I2S Clock Interface - Master Mode on page 70
- I2S Clock Interface - Slave Mode on page 71
- I2S Receiver Interface on page 72
- I2S Transmitter Interface on page 73
- DMA Interface on page 74
- I2S Interrupts on page 79

4.1 APB Slave Interface Signals

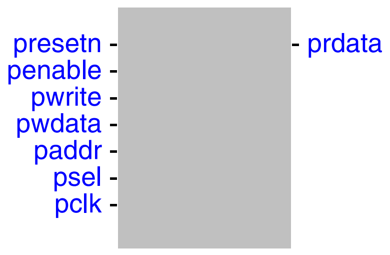


Table 4-1 APB Slave Interface Signals

Port Name	I/O	Description
presetn	I	An APB interface domain reset. This signal resets only the bus interface. The signal is asserted asynchronously, but is de-asserted synchronously after the rising edge of pclk. The synchronization must be provided external to the component. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
penable	I	APB enable control. Asserted for a single pclk cycle and used for timing read/write operations. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pwrite	I	APB write control. When high, indicates a write access to the peripheral; when low, indicates a read access. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-1 APB Slave Interface Signals (Continued)

Port Name	I/O	Description
pwdata[(APB_DATA_WIDTH-1):0]	I	APB write data bus. Driven by the bus master (bridge unit) during write cycles. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
prdata[(APB_DATA_WIDTH-1):0]	O	APB readback data. Driven by the selected peripheral during read cycles. **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
paddr[(I2S_ADDR_SLICE_LHS-1):0]	I	APB address bus. Uses lower 7 bits of the address bus for register decoding. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
psel	I	APB peripheral select that lasts for two pclk cycles. When asserted, indicates that the peripheral has been selected for read/write operation. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pclk	I	APB clock for the bus interface unit. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.2 I2S Clock Interface Signals

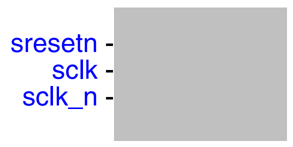


Table 4-2 I2S Clock Interface Signals

Port Name	I/O	Description
sresetn	I	An SCLK domain reset. The signal is asserted asynchronously, but is deasserted synchronously after the rising edge of sclk. The synchronization must be provided external to this component. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
sclk	I	Serial interface clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
sclk_n	I	Inverted serial interface clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.3 I2S Clock Interface - Master Mode Signals

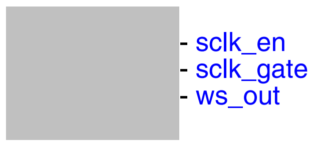


Table 4-3 I2S Clock Interface - Master Mode Signals

Port Name	I/O	Description
sclk_en	O	External sclk enable signal. This signal can be AND'd with sclk to disable the clock when the master device is disabled. **Exists:** I2S_MODE_EN != 0 **Synchronous To:** sclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
sclk_gate	O	Clock gating signal. Since sclk_gate is 1 during the cycles that are to be gated off, the actual gating of sclk needs to be done externally by AND'ing the inverse of the generated sclk_gate signal with sclk. **Exists:** I2S_MODE_EN != 0 **Synchronous To:** sclk_n **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
ws_out	O	Word select line when DW_apb_i2s is a master. **Exists:** I2S_MODE_EN != 0 **Synchronous To:** sclk_n **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4 I2S Clock Interface - Slave Mode Signals

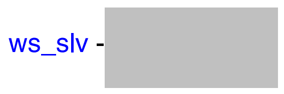


Table 4-4 I2S Clock Interface - Slave Mode Signals

Port Name	I/O	Description
ws_slv	I	Word select line when DW_apb_i2s is a slave. **Exists:** I2S_MODE_EN == 0 **Synchronous To:** sclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 I2S Receiver Interface (for x = 0; x <= I2S_RX_CHANNELS-1) Signals

Table 4-5 I2S Receiver Interface (for x = 0; x <= I2S_RX_CHANNELS-1) Signals

Port Name	I/O	Description
sdix	I	Serial data input for Receive Channel x, where x is the number of the receive channel. **Exists:** I2S_RECEIVER_BLOCK ==1 **Synchronous To:** sclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.6 I2S Transmitter Interface (for x = 0; x <= I2S_TX_CHANNELS-1) Signals

Table 4-6 I2S Transmitter Interface (for x = 0; x <= I2S_TX_CHANNELS-1) Signals

Port Name	I/O	Description
sdox	O	Serial data output for Transmit Channel x, where x is the number of the transmit channel. **Exists:** I2S_TRANSMITTER_BLOCK ==1 **Synchronous To:** sclk_n **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.7 DMA Interface Signals

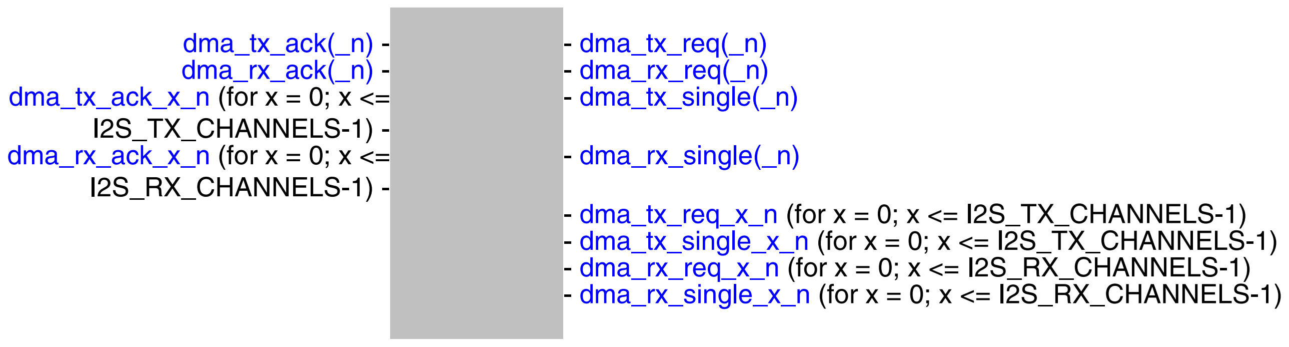

dma_tx_ack(_n) -
dma_rx_ack(_n) -
dma_tx_ack_x_n (for x = 0; x <= I2S_TX_CHANNELS-1) -
dma_rx_ack_x_n (for x = 0; x <= I2S_RX_CHANNELS-1) -
- dma_tx_req(_n)
- dma_rx_req(_n)
- dma_tx_single(_n)
- dma_rx_single(_n)
- dma_tx_req_x_n (for x = 0; x <= I2S_TX_CHANNELS-1)
- dma_tx_single_x_n (for x = 0; x <= I2S_TX_CHANNELS-1)
- dma_rx_req_x_n (for x = 0; x <= I2S_RX_CHANNELS-1)
- dma_rx_single_x_n (for x = 0; x <= I2S_RX_CHANNELS-1)

Table 4-7 DMA Interface Signals

Port Name	I/O	Description
dma_tx_ack(_n)	I	DMA Transmit Acknowledgement. Sent by the DMA Controller to acknowledge the end of each DMA burst or single transaction to the transmit FIFO. **Exists:** I2S_TRANSMITTER_BLOCK ==1 && I2S_DMA_HS_TYPE == 1 && I2S_HAS_DMA_INTERFACE ==1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low
dma_rx_ack(_n)	I	DMA Receive Acknowledgement. Sent by the DMA Controller to acknowledge the end of each DMA burst or single transaction to the receive FIFO. **Exists:** I2S_RECEIVER_BLOCK ==1 && I2S_DMA_HS_TYPE == 1 && I2S_HAS_DMA_INTERFACE ==1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low

Table 4-7 DMA Interface Signals (Continued)

Port Name	I/O	Description
dma_tx_req(_n)	O	Transmit FIFO DMA Request. Asserted when the transmit FIFO requires service from the DMA Controller; that is, the transmit FIFO is at or below the watermark level. 0 - not requesting 1 - requesting Software must set up the DMA controller with the number of words to be transferred when a request is made. When using the DW_ahb_dmac, this value is programmed in the DEST_MSIZE field of the CTLx register. **Exists:** I2S_TRANSMITTER_BLOCK ==1 && I2S_DMA_HS_TYPE == 1 && I2S_HAS_DMA_INTERFACE ==1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low
dma_rx_req(_n)	O	Receive FIFO DMA Request. Asserted when the receive FIFO requires service from the DMA Controller; that is, the receive FIFO is at or above the watermark level. 0 - not requesting 1 - requesting Software must set up the DMA controller with the number of words to be transferred when a request is made. When using the DW_ahb_dmac, this value is programmed in the SRC_MSIZE field of the CTLx register. **Exists:** I2S_RECEIVER_BLOCK ==1 && I2S_DMA_HS_TYPE == 1 && I2S_HAS_DMA_INTERFACE ==1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low
dma_tx_single(_n)	O	DMA Transmit FIFO Single Signal. This DMA status output informs the DMA Controller that there is at least one free entry in the transmit FIFO. This output does not request a DMA transfer. 0 - Transmit FIFO is full 1 - Transmit FIFO is not full **Exists:** I2S_TRANSMITTER_BLOCK ==1 && I2S_DMA_HS_TYPE == 1 && I2S_HAS_DMA_INTERFACE ==1 **Synchronous To:** pclk **Registered:** I2S_TRANSMITTER_BLOCK==1 && I2S_TX_CHANNELS>=2 ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low

Table 4-7 DMA Interface Signals (Continued)

Port Name	I/O	Description
dma_rx_single(_n)	O	DMA Receive FIFO Single Signal. This DMA status output informs the DMA Controller that there is at least one valid data entry in the receive FIFO. This output does not request a DMA transfer. 0 - Receive FIFO is empty 1 - Receive FIFO is not empty **Exists:** I2S_RECEIVER_BLOCK ==1 && I2S_DMA_HS_TYPE == 1 && I2S_HAS_DMA_INTERFACE ==1 **Synchronous To:** pclk **Registered:** I2S_RECEIVER_BLOCK==1 && I2S_RX_CHANNELS>=2 ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low
dma_tx_ack_x(_n) (for x = 0; x <= I2S_TX_CHANNELS-1)	I	DMA Transmit Acknowledgement. Sent by the DMA Controller to acknowledge the end of each DMA burst or single transaction to the transmit FIFO. **Exists:** I2S_TRANSMITTER_BLOCK ==1 && I2S_DMA_HS_TYPE == 0 && I2S_HAS_DMA_INTERFACE ==1 && I2S_TX_CHANNELS-1 >= x **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low
dma_rx_ack_x(_n) (for x = 0; x <= I2S_RX_CHANNELS-1)	I	DMA Receive Acknowledgement. Sent by the DMA controller to acknowledge the end of each DMA burst or single transaction from the receive FIFO. **Exists:** I2S_RECEIVER_BLOCK ==1 && I2S_DMA_HS_TYPE == 0 && I2S_HAS_DMA_INTERFACE ==1 && I2S_RX_CHANNELS-1 >= x **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low

Table 4-7 DMA Interface Signals (Continued)

Port Name	I/O	Description
dma_tx_req_x(_n) (for x = 0; x <= I2S_TX_CHANNELS-1)	O	Transmit FIFO DMA Request. Asserted when the transmit FIFO requires service from the DMA Controller; that is, the transmit FIFO is at or below the watermark level. 0 not requesting 1 requesting Software must set up the DMA controller with the number of words to be transferred when a request is made. When using the DW_ahb_dmac, this value is programmed in the DEST_MSIZE field of the CTLx register. **Exists:** I2S_TRANSMITTER_BLOCK ==1 && I2S_DMA_HS_TYPE == 0 && I2S_HAS_DMA_INTERFACE ==1 && I2S_TX_CHANNELS-1 >= x **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low
dma_tx_single_x(_n) (for x = 0; x <= I2S_TX_CHANNELS-1)	O	DMA Transmit FIFO Single Signal. This DMA status output informs the DMA Controller that there is at least one free entry in the transmit FIFO. This output does not request a DMA transfer. 0: Transmit FIFO is full 1: Transmit FIFO is not full **Exists:** I2S_TRANSMITTER_BLOCK ==1 && I2S_DMA_HS_TYPE == 0 && I2S_HAS_DMA_INTERFACE ==1 && I2S_TX_CHANNELS-1 >= x **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low
dma_rx_req_x(_n) (for x = 0; x <= I2S_RX_CHANNELS-1)	O	Receive FIFO DMA Request. Asserted when the receive FIFO requires service from the DMA Controller; that is, the receive FIFO is at or above the watermark level. 0 not requesting 1 requesting Software must set up the DMA controller with the number of words to be transferred when a request is made. When using the DW_ahb_dmac, this value is programmed in the SRC_MSIZE field of the CTLx register. **Exists:** I2S_RECEIVER_BLOCK ==1 && I2S_DMA_HS_TYPE == 0 && I2S_HAS_DMA_INTERFACE ==1 && I2S_RX_CHANNELS-1 >= x **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low

Table 4-7 DMA Interface Signals (Continued)

Port Name	I/O	Description
dma_rx_single_x(_n) (for x = 0; x <= I2S_RX_CHANNELS-1)	O	DMA Receive FIFO Single Signal. This DMA status output informs the DMA Controller that there is at least one valid data entry in the receive FIFO. This output does not request a DMA transfer. 0: Receive FIFO is empty 1: Receive FIFO is not empty **Exists:** I2S_RECEIVER_BLOCK ==1 && I2S_DMA_HS_TYPE == 0 && I2S_HAS_DMA_INTERFACE ==1 && I2S_RX_CHANNELS-1 >= x **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_DMA_POL=1 otherwise Low

4.8 I2S Interrupts Signals

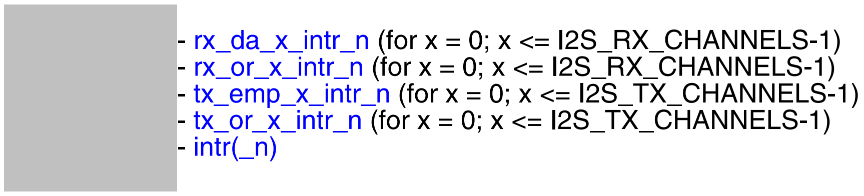

- rx_da_x_intr_n (for x = 0; x <= I2S_RX_CHANNELS-1)
- rx_or_x_intr_n (for x = 0; x <= I2S_RX_CHANNELS-1)
- tx_emp_x_intr_n (for x = 0; x <= I2S_TX_CHANNELS-1)
- tx_or_x_intr_n (for x = 0; x <= I2S_TX_CHANNELS-1)
- intr(_n)

Table 4-8 I2S Interrupts Signals

Port Name	I/O	Description
rx_da_x_intr(_n) (for x = 0; x <= I2S_RX_CHANNELS-1)	O	Data Available Interrupt for Receive Channel x, where x is the number of the receive channel. This interrupt is asserted when the trigger level for the RX FIFO is reached. **Exists:** I2S_INTERRUPT_SIGNALS == 1 & I2S_RECEIVER_BLOCK==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_INTR_POL=1 otherwise Low
rx_or_x_intr(_n) (for x = 0; x <= I2S_RX_CHANNELS-1)	O	Data Overrun Interrupt for Receive Channel x, where x is the number of the receive channel. This interrupt is asserted when an attempt is made to write received data to full RX FIFO (any data being written is lost while data in the FIFO is preserved). **Exists:** I2S_INTERRUPT_SIGNALS == 1 & I2S_RECEIVER_BLOCK==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_INTR_POL=1 otherwise Low
tx_emp_x_intr(_n) (for x = 0; x <= I2S_TX_CHANNELS-1)	O	FIFO Empty Interrupt for Transmit Channel x, where x is the number of the transmit channel. This interrupt is asserted when the empty trigger threshold level for the TX FIFO is reached. **Exists:** I2S_INTERRUPT_SIGNALS == 1 & I2S_TRANSMITTER_BLOCK==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_INTR_POL=1 otherwise Low

Table 4-8 I2S Interrupts Signals (Continued)

Port Name	I/O	Description
tx_or_x_intr(_n) (for x = 0; x <= I2S_TX_CHANNELS-1)	O	Data Overrun Interrupt for Transmit Channel x, where x is the number of the transmit channel. This interrupt is asserted when an attempt is made to write to a full TX FIFO (any data being written is lost while data in the FIFO is preserved). **Exists:** I2S_INTERRUPT_SIGNALS == 1 & I2S_TRANSMITTER_BLOCK==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_INTR_POL=1 otherwise Low
intr(_n)	O	DW_apb_i2s global interrupt signal. **Exists:** I2S_INTERRUPT_SIGNALS == 0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when I2S_INTR_POL=1 otherwise Low

5 Register Descriptions

This chapter details all possible registers in the controller. They are arranged hierarchically into maps and blocks (banks). For configurable IP titles, your actual configuration might not contain all of these registers.

Attention: For configurable IP titles, do not use this document to determine the exact attributes of your register map. It is for reference purposes only.

When you configure the controller in coreConsultant, you must access the register attributes for your actual configuration at workspace/report/ComponentRegisters.html or workspace/report/ComponentRegisters.xml after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the registers that are not in your actual configuration. This does not apply to non-configurable IP titles. In addition, all parameter expressions are evaluated to actual values. Therefore, the Offset and Memory Access values might change depending on your actual configuration.

Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Exists Expressions

These expressions indicate the combination of configuration parameters required for a register, field, or block to exist in the memory map. The expression is only valid in the local context and does not indicate the conditions for existence of the parent. For example, the expression for a bit field in a register assumes that the register exists and does not include the conditions for existence of the register.

Offset

The term *Offset* is synonymous with *Address.*

Memory Access Attributes

The Memory Access attribute is defined as <ReadBehavior>/<WriteBehavior> which are defined in the following table.

Table 5-1 Possible Read and Write Behaviors

Read (or Write) Behavior	Description
RC	A read clears this register field.
RS	A read sets this register field.
RM	A read modifies the contents of this register field.
Wo	You can only write to this register once field.
W1C	A write of 1 clears this register field.
W1S	A write of 1 sets this register field.
W1T	A write of 1 toggles this register field.
W0C	A write of 0 clears this register field.
W0S	A write of 0 sets this register field.
W0T	A write of 0 toggles this register field.
WC	Any write clears this register field.
WS	Any write sets this register field.
WM	Any write toggles this register field.
no Read Behavior attribute	You cannot read this register. It is Write-Only.
no Write Behavior attribute	You cannot write to this register. It is Read-Only.

Table 5-2 Memory Access Examples

Memory Access	Description
R	Read-only register field.
W	Write-only register field.
R/W	Read/write register field.
R/W1C	You can read this register field. Writing 1 clears it.
RC/W1C	Reading this register field clears it. Writing 1 clears it.
R/Wo	You can read this register field. You can only write to it once.

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

Attribute	Description
Volatile	As defined by the IP-XACT specification. If true, indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this field can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. For example, when the core updates the register field contents.
Testable	As defined by the IP-XACT specification. Possible values are unconstrained, untestable, readOnly, writeAsRead, restore. Untestable means that this field is untestable by a simple automated register test. For example, the read-write access of the register is controlled by a pin or another register. readOnly means that you should not write to this register; only read from it. This might apply for a register that modifies the contents of another register.
Reset Mask	As defined by the IP-XACT specification. Indicates that this register field has an unknown reset value. For example, the reset value is set by another register or an input pin; or the register is implemented using RAM.
* Varies	Indicates that the memory access (or reset) attribute (read, write behavior) is not fixed. For example, the read-write access of the register is controlled by a pin or another register. Or when the access depends on some configuration parameter; in this case the post-configuration report in coreConsultant gives the actual access value.

Component Banks/Blocks

The following table shows the address blocks for each memory map. Follow the link for an address block to see a table of its registers.

Table 5-4 Address Banks/Blocks for Memory Map: DW_apb_i2s_mem_map

Address Block	Description
DW_apb_i2s_addr_block1 on page 84	DW_apb_i2s address block **Exists:** Always

5.1 DW_apb_i2s_mem_map/DW_apb_i2s_addr_block1 Registers

DW_apb_i2s address block. Follow the link for the register to see a detailed description of the register.

Table 5-5 Registers for Address Block: DW_apb_i2s_mem_map/DW_apb_i2s_addr_block1

Register	Offset	Description
IER on page 87	0x0	This register acts as a global enable/disable for DW_apb_i2s.
IRER on page 88	0x4	This register acts as an enable/disable for the DW_apb_i2s Receiver block.
ITER on page 89	0x8	This register acts as an enable/disable for the DW_apb_i2s Transmitter block.
CER on page 90	0xc	This register acts as an enable/disable for the DW_apb_i2s Clock Generation block, which is only...
CCR on page 91	0x10	This register configures the ws_out and sclk_gate signals when DW_apb_i2s is a master.
RXFFR on page 93	0x14	This register specifies the Receiver Block FIFO Reset Register.
TXFFR on page 94	0x18	This register specifies the Transmitter Block FIFO Reset Register.
LRBRx (for x = 0; x <= I2S_RX_CHANNELS-1) on page 95	0x020 + 0x40*x	This specifies the Left Receive Buffer Register.
LTHRx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 97	0x020 + 0x40*x	This specifies the Left Transmit Holding Register.
RRBRx (for x = 0; x <= I2S_RX_CHANNELS-1) on page 99	0x024 + 0x40*x	This specifies the Right Receive Buffer Register.
RTHRx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 101	0x024 + 0x40*x	This specifies the Right Transmit Holding Register.
RERx (for x = 0; x <= I2S_RX_CHANNELS-1) on page 103	0x028 + 0x40*x	This specifies the Receive Enable Register.
TERx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 104	0x02C + 0x40*x	This specifies the Transmit Enable Register.

Table 5-5 Registers for Address Block: DW_apb_i2s_mem_map/DW_apb_i2s_addr_block1 (Continued)

Register	Offset	Description
RCRx (for x = 0; x <= I2S_RX_CHANNELS-1) on page 105	0x030 + 0x40*x	This specifies the Receive Configuration Register.
TCRx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 107	0x034 + 0x40*x	This specifies the Transmit Configuration Register.
ISRx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 109	0x038 + 0x40*x	This specifies the Interrupt Status Register.
IMRx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 111	0x03C + 0x40*x	This specifies the Interrupt Mask Register.
RORx (for x = 0; x <= I2S_RX_CHANNELS-1) on page 113	0x040 + 0x40*x	This specifies the Receive Overrun Register.
TORx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 114	0x044 + 0x40*x	This specifies the Transmit Overrun Register.
RFCRx (for x = 0; x <= I2S_RX_CHANNELS-1) on page 115	0x048 + 0x40*x	This specifies the Receive FIFO Configuration Register.
TFCRx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 118	0x04C + 0x40*x	This specifies the Transmit FIFO Configuration Register.
RFFx (for x = 0; x <= I2S_RX_CHANNELS-1) on page 121	0x050 + 0x40*x	This specifies the Receive FIFO Flush Register.
TFFx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 122	0x054 + 0x40*x	This specifies the Transmit FIFO Flush Register.
RXDMA on page 123	0x1c0	The RXDMA register allows access to all enabled Receive channels via a single point rather than...
RRXDMA on page 125	0x1c4	The RXDMA can be reset to the lowest enabled Channel via the RRXDMA register. The RRXDMA register can...
TXDMA on page 127	0x1c8	The TXDMA register functions similar to the RXDMA register and allows write accesses to all of...
RTXDMA on page 128	0x1cc	This register provides the same functionality as the RRXDMA register but targets TXDMA...

Table 5-5 Registers for Address Block: DW_apb_i2s_mem_map/DW_apb_i2s_addr_block1 (Continued)

Register	Offset	Description
I2S_COMP_PARAM_2 on page 129	0x1f0	This specifies bits for Component Parameter Register 2. Note: This is a constant read-only register...
I2S_COMP_PARAM_1 on page 132	0x1f4	This specifies bits for Component Parameter Register 1. Note: This is a constant read-only register...
I2S_COMP_VERSION on page 136	0x1f8	This register specifies the I2S Component Version.
I2S_COMP_TYPE on page 137	0x1fc	This register specifies the I2S Component Type.
DMACR on page 138	0x200	This register is only valid when DW_apb_i2s is configured with a set of DMA Controller interface...
RXDMA_CHx (for x = 0; x <= I2S_RX_CHANNELS-1) on page 142	0x204	The RXDMA_CHx register allows access to enabled Receive channel x via a single point rather than...
TXDMA_CHx (for x = 0; x <= I2S_TX_CHANNELS-1) on page 144	0x214	The TXDMA_CHx register allows access to enabled Transmit channel x via a single point rather than...

5.1.1 IER

- **Name:** DW_apb_i2s Enable Register
- **Description:** This register acts as a global enable/disable for DW_apb_i2s.
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always

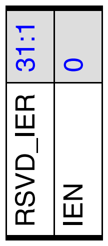


Table 5-6 Fields for Register: IER

Bits	Name	Memory Access	Description
31:1	RSVD_IER	R	RSVD_IER Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	IEN	R/W	DW_apb_i2s enable. This bit enables or disables DW_apb_i2s. A disable on this bit overrides any other block or channel enables and flushes all FIFOs. For more information about how this register affects the other DW_apb_i2s blocks, refer to "DW_apb_i2s Enable". **Values:** ■ 0x0 (DISABLED): DW_apb_i2s disabled. ■ 0x1 (ENABLED): DW_apb_i2s enabled **Value After Reset:** 0x0 **Exists:** Always

5.1.2 IRER

- **Name:** I2S Receiver Block Enable Register
- **Description:** This register acts as an enable/disable for the DW_apb_i2s Receiver block.
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** I2S_RECEIVER_BLOCK==1

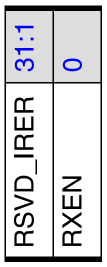


Table 5-7 Fields for Register: IRER

Bits	Name	Memory Access	Description
31:1	RSVD_IRER	R	RSVD_IRER Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	RXEN	R/W	Receiver block enable. This bit enables or disables the receiver. A disable on this bit overrides any individual receive channel enables. For more information about the receiver block, refer to "DW_apb_i2s as Receiver". **Values:** ■ 0x0 (DISABLED): Receiver disabled ■ 0x1 (ENABLED): Receiver enabled **Value After Reset:** 0x0 **Exists:** I2S_RECEIVER_BLOCK==1

5.1.3 ITER

- **Name:** I2S Transmitter Block Enable Register
- **Description:** This register acts as an enable/disable for the DW_apb_i2s Transmitter block.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** I2S_TRANSMITTER_BLOCK==1

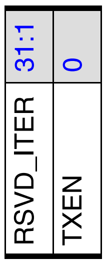


Table 5-8 Fields for Register: ITER

Bits	Name	Memory Access	Description
31:1	RSVD_ITER	R	RSVD_ITER Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	TXEN	R/W	Transmitter block enable. This bit enables or disables the transmitter. A disable on this bit overrides any individual transmit channel enables. For more information about the transmitter block, refer to "DW_apb_i2s as Transmitter". **Values:** ■ 0x0 (DISABLED): Transmitter disabled ■ 0x1 (ENABLED): Transmitter enabled **Value After Reset:** 0x0 **Exists:** I2S_TRANSMITTER_BLOCK==1

5.1.4 CER

- **Name:** Clock Enable Register
- **Description:** This register acts as an enable/disable for the DW_apb_i2s Clock Generation block, which is only relevant in master mode (I2S_MODE_EN = 1). When this block is enabled, the clock signals sclk_en, ws_out, and sclk_gate appear on the interface.
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** I2S_MODE_EN==1

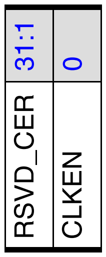


Table 5-9 Fields for Register: CER

Bits	Name	Memory Access	Description
31:1	RSVD_CER	R	RSVD_CER Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	CLKEN	R/W	Clock generation enable/disable. This bit enables/disables the clock generation signals when DW_apb_i2s is a master: sclk_en, ws_out, and sclk_gate. For more information about clock generation, refer to "Clock Generation (Master Mode)". **Values:** ■ 0x0 (DISABLED): Clock generation disabled ■ 0x1 (ENABLED): Clock generation enabled **Value After Reset:** 0x0 **Exists:** I2S_MODE_EN==1

5.1.5 CCR

- **Name:** Clock Configuration Register
- **Description:** This register configures the ws_out and sclk_gate signals when DW_apb_i2s is a master.
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** I2S_MODE_EN==1

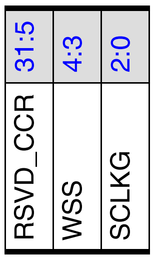


Table 5-10 Fields for Register: CCR

Bits	Name	Memory Access	Description
31:5	RSVD_CCR	R	RSVD_CCR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
4:3	WSS	R/W	These bits are used to program the number of sclk cycles for which the word select line (ws_out) stays in the left or right sample mode. The I2S Clock Generation block must be disabled (CER[0] = 0) prior to any changes in this value. **Values:** ■ 0x0 (CLOCK_CYCLES_16): 16 sclk cycles ■ 0x1 (CLOCK_CYCLES_24): 24 sclk cycles ■ 0x2 (CLOCK_CYCLES_32): 32 sclk cycles **Value After Reset:** I2S_WS_LENGTH **Exists:** I2S_MODE_EN==1

Table 5-10 Fields for Register: CCR (Continued)

Bits	Name	Memory Access	Description
2:0	SCLKG	R/W	These bits are used to program the gating of sclk. The programmed gating value must be less than or equal to the largest configured/programmed audio resolution to prevent the truncating of RX/TX data. The I2S Clock Generation block must be disabled (CER[0] = 0) before making any changes in this value. **Values:** ■ 0x0 (NO_CLOCK_GATING): Clock gating is disabled ■ 0x1 (CLOCK_CYCLES_12): Gating after 12 sclk cycles ■ 0x2 (CLOCK_CYCLES_16): Gating after 16 sclk cycles ■ 0x3 (CLOCK_CYCLES_20): Gating after 20 sclk cycles ■ 0x4 (CLOCK_CYCLES_24): Gating after 24 sclk cycles **Value After Reset:** I2S_SCLK_GATE **Exists:** I2S_MODE_EN==1

5.1.6 RXFFR

- **Name:** Receiver Block FIFO Reset Register
- **Description:** This register specifies the Receiver Block FIFO Reset Register.
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** I2S_RECEIVER_BLOCK==1

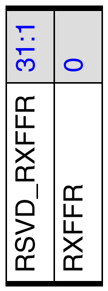


Table 5-11 Fields for Register: RXFFR

Bits	Name	Memory Access	Description
31:1	RSVD_RXFFR	W	RSVD_RXFFR Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RXFFR	W	Receiver FIFO Reset. Writing a 1 to this register flushes all the RX FIFOs (this is a self clearing bit). The Receiver Block must be disabled before writing to this bit. **Values:** ■ 0x0 (NO_FLUSH): Does not flush the RX FIFO ■ 0x1 (FLUSH): Flushes the RX FIFO **Value After Reset:** 0x0 **Exists:** I2S_RECEIVER_BLOCK==1 **Volatile:** true

5.1.7 TXFFR

- **Name:** Transmitter Block FIFO Reset Register
- **Description:** This register specifies the Transmitter Block FIFO Reset Register.
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** I2S_TRANSMITTER_BLOCK==1

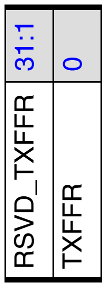


Table 5-12 Fields for Register: TXFFR

Bits	Name	Memory Access	Description
31:1	RSVD_TXFFR	W	RSVD_TXFFR Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	TXFFR	W	Transmitter FIFO Reset. Writing a 1 to this register flushes all the TX FIFOs (this is a self clearing bit). The Transmitter Block must be disabled prior to writing this bit. **Values:** ■ 0x0 (NO_FLUSH): Does not flush the TX FIFO ■ 0x1 (FLUSH): Flushes the TX FIFO **Value After Reset:** 0x0 **Exists:** I2S_TRANSMITTER_BLOCK==1 **Volatile:** true

5.1.8 LRBRx (for x = 0; x <= I2S_RX_CHANNELS-1)

- **Name:** Left Receive Buffer Register x
- **Description:** This specifies the Left Receive Buffer Register.
- **Size:** 32 bits
- **Offset:** 0x020 + 0x40*x
- **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1)

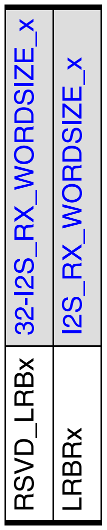


Table 5-13 Fields for Register: LRBRx (for x = 0; x <= I2S_RX_CHANNELS-1)

Bits	Name	Memory Access	Description
32-I2S_RX_WORDSIZE_x	RSVD_LRBx	R	RSVD_LRBRx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-13 Fields for Register: LRBRx (for x = 0; x <= I2S_RX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
I2S_RX_WORDSIZE_x	LRBRx	R	The left stereo data received serially from the receive channel input (sdix). If the RX FIFO is full and the two-stage read operation (for instance, a read from LRBRx followed by a read from RRBRx) is not performed before the start of the next stereo pair, then the new data is lost and an overrun interrupt occurs. (data already in the RX FIFO is preserved.) **Note**: Before reading this register again, the right stereo data must be read from RRBRx or the status/interrupts will not be valid. **Value After Reset:** 0x0 **Exists:** I2S_RX_CHANNELS > x **Volatile:** true

5.1.9 LTHRx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Left Transmit Holding Register x
- **Description:** This specifies the Left Transmit Holding Register.
- **Size:** 32 bits
- **Offset:** 0x020 + 0x40*x
- **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1)

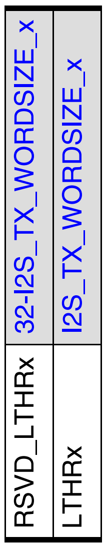


Table 5-14 Fields for Register: LTHRx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
32-I2S_TX_WORDSIZE_x	RSVD_LTHRx	W	RSVD_LTHRx Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-14 Fields for Register: LTHRx (for x = 0; x <= I2S_TX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
I2S_TX_WORDSIZE_x	LTHRx	W	The left stereo data to be transmitted serially through the transmit channel output (sdox) is written through this register. Writing is a two-stage process: 1. A write to this register passes the left stereo sample to the transmitter. 2. This MUST be followed by writing the right stereo sample to the RTHRx register. Data must only be written to the FIFO when it is not full. Any attempt to write to a full FIFO results in that data being lost and an overrun interrupt being generated. **Value After Reset:** 0x0 **Exists:** I2S_TX_CHANNELS > x **Volatile:** true

5.1.10 RRBRx (for x = 0; x <= I2S_RX_CHANNELS-1)

- **Name:** Right Transmit Holding Register x
- **Description:** This specifies the Right Receive Buffer Register.
- **Size:** 32 bits
- **Offset:** 0x024 + 0x40*x
- **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1)

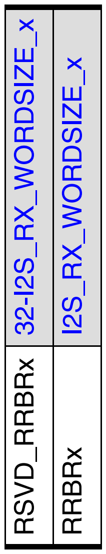


Table 5-15 Fields for Register: RRBRx (for x = 0; x <= I2S_RX_CHANNELS-1)

Bits	Name	Memory Access	Description
32-I2S_RX_WORDSIZE_x	RSVD_RRBRx	R	RSVD_RRBRx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-15 Fields for Register: RRBRx (for x = 0; x <= I2S_RX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
I2S_RX_WORDSIZE_x	RRBRx	R	The right stereo data received serially from the receive channel input (sdix) is read through this register. If the RX FIFO is full and the two-stage read operation (for instance, read from LRBRx followed by a read from RRBRx) is not performed before the start of the next stereo pair, then the new data is lost and an overrun interrupt occurs. (Data already in the RX FIFO is preserved.) **Note**: Prior to reading this register, the left stereo data MUST be read from LRBRx, or the status/interrupts will not be valid. **Value After Reset:** 0x0 **Exists:** I2S_RX_CHANNELS > x **Volatile:** true

5.1.11 RTHRx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Description:** This specifies the Right Transmit Holding Register.
- **Size:** 32 bits
- **Offset:** 0x024 + 0x40*x
- **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1)

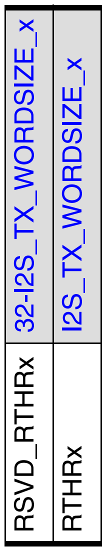


Table 5-16 Fields for Register: RTHRx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
32-I2S_TX_WORDSIZE_x	RSVD_RTHRx	W	RSVD_RTHRx Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-16 Fields for Register: RTHRx (for x = 0; x <= I2S_TX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
I2S_TX_WORDSIZE_x	RTHRx	W	The right stereo data to be transmitted serially through the transmit channel output (sdox) is written through this register. Writing is a two-stage process: 1. A left stereo sample MUST be written to the LTHRx register. 2. A write to this register passes the right stereo sample to the transmitter. Data should only be written to the FIFO when it is not full. Any attempt to write to a full FIFO results in that data being lost and an overrun interrupt being generated. **Value After Reset:** 0x0 **Exists:** I2S_TX_CHANNELS > x **Volatile:** true

5.1.12 RERx (for x = 0; x <= I2S_RX_CHANNELS-1)

- **Name:** Receive Enable Register x
- **Description:** This specifies the Receive Enable Register.
- **Size:** 32 bits
- **Offset:** 0x028 + 0x40*x
- **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1)

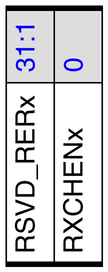


Table 5-17 Fields for Register: RERx (for x = 0; x <= I2S_RX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:1	RSVD_RERx	R	RSVD_RERx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	RXCHENx	R/W	Receive channel enable. This bit enables/disables a receive channel, independently of all other channels. On enable, the channel begins receiving on the next left stereo cycle. A global disable of DW_apb_i2s (IER[0] = 0) or the Receiver block (IRER[0] = 0) overrides this value. **Values:** ■ 0x0 (DISABLED): Receive Channel Disable ■ 0x1 (ENABLED): Receive Channel Enable **Value After Reset:** 0x1 **Exists:** I2S_RX_CHANNELS > x

5.1.13 TERx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Transmit Enable Register x
- **Description:** This specifies the Transmit Enable Register.
- **Size:** 32 bits
- **Offset:** 0x02C + 0x40*x
- **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1)

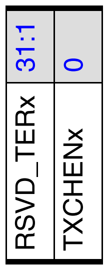


Table 5-18 Fields for Register: TERx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:1	RSVD_TERx	R	RSVD_TERx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	TXCHENx	R/W	Transmit channel enable. This bit enables/disables a transmit channel, independently of all other channels. On enable, the channel begins transmitting on the next left stereo cycle. A global disable of DW_apb_i2s (IER[0] = 0) or Transmitter block (ITER[0] = 0) overrides this value. **Values:** ■ 0x0 (DISABLED): Transmit Channel Disable ■ 0x1 (ENABLED): Transmit Channel Enable **Value After Reset:** 0x1 **Exists:** I2S_TX_CHANNELS > x

5.1.14 RCRx (for x = 0; x <= I2S_RX_CHANNELS-1)

- **Name:** Receive Configuration Register x
- **Description:** This specifies the Receive Configuration Register.
- **Size:** 32 bits
- **Offset:** 0x030 + 0x40*x
- **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1)

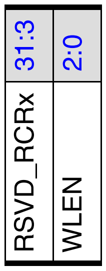


Table 5-19 Fields for Register: RCRx (for x = 0; x <= I2S_RX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:3	RSVD_RCRx	R	RSVD_RCRx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-19 Fields for Register: RCRx (for x = 0; x <= I2S_RX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
2:0	WLEN	R/W	These bits are used to program the desired data resolution of the receiver and enables the LSB of the incoming left (or right) word to be placed in the LSB of the LRBRx (or RRBRx) register. Programmed data resolution must be less than or equal to I2S_RX_WORDSIZE_x. If the selected resolution is greater than the I2S_RX_WORDSIZE_x, the receive channel defaults back to I2S_RX_WORDSIZE_RESET_VALUE_x. The channel must be disabled prior to any changes in this value (RER0[0] = 0). **Values:** ■ 0x0 (IGNORE_WORD_LENGTH): Ignore the word length ■ 0x1 (RESOLUTION_12_BIT): 12-bit data resolution of the receiver. ■ 0x2 (RESOLUTION_16_BIT): 16-bit data resolution of the receiver. ■ 0x3 (RESOLUTION_20_BIT): 20-bit data resolution of the receiver. ■ 0x4 (RESOLUTION_24_BIT): 24-bit data resolution of the receiver. ■ 0x5 (RESOLUTION_32_BIT): 32-bit data resolution of the receiver. **Value After Reset:** I2S_RX_WORDSIZE_RESET_VALUE_[x] **Exists:** I2S_RX_CHANNELS > x

5.1.15 TCRx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Transmit Configuration Register x
- **Description:** This specifies the Transmit Configuration Register.
- **Size:** 32 bits
- **Offset:** 0x034 + 0x40*x
- **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1)

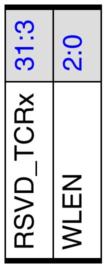


Table 5-20 Fields for Register: TCRx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:3	RSVD_TCRx	R	RSVD_TCRx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-20 Fields for Register: TCRx (for x = 0; x <= I2S_TX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
2:0	WLEN	R/W	These bits are used to program the data resolution of the transmitter and ensures the MSB of the data is transmitted first. Programmed resolution must be less than or equal to I2S_TX_WORDSIZE_x. If the selected resolution is greater than I2S_TX_WORDSIZE_x, the transmit channel defaults back to I2S_TX_WORDSIZE_RESET_VALUE_x value. The channel must be disabled prior to any changes in this value (TER0[0] = 0). **Values:** ■ 0x0 (IGNORE_WORD_LENGTH): Ignore the word length ■ 0x1 (RESOLUTION_12_BIT): 12-bit data resolution of the transmitter. ■ 0x2 (RESOLUTION_16_BIT): 16-bit data resolution of the transmitter. ■ 0x3 (RESOLUTION_20_BIT): 20-bit data resolution of the transmitter. ■ 0x4 (RESOLUTION_24_BIT): 24-bit data resolution of the transmitter. ■ 0x5 (RESOLUTION_32_BIT): 32-bit data resolution of the transmitter. **Value After Reset:** I2S_TX_WORDSIZE_RESET_VALUE_[x] **Exists:** I2S_TX_CHANNELS > x

5.1.16 ISRx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Interrupt status Register x
- **Description:** This specifies the Interrupt Status Register.
- **Size:** 32 bits
- **Offset:** 0x038 + 0x40*x
- **Exists:** ((I2S_TX_CHANNELS>x) ? 1 : ((I2S_RX_CHANNELS>x) ? 1 : 0))

31:6	5	4	3:2	1	0
RSVD31_6	TXFO	TXFE	RSVD3_2	RXFO	RXDA

Table 5-21 Fields for Register: ISRx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:6	RSVD31_6	R	RSVD31_6 Reserved bits - Read Only **Exists:** Always **Volatile:** true
5	TXFO	R	Status of Data Overrun interrupt for the TX channel. This bit specifies whether the TX FIFO write is valid or an overrun. Attempt to write to full TX FIFO. **Values:** ■ 0x0 (WRITE_VALID): TX FIFO write valid ■ 0x1 (WRITE_OVERRUN): TX FIFO write overrun **Value After Reset:** 0x0 **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1) **Volatile:** true

Table 5-21 Fields for Register: ISRx (for x = 0; x <= I2S_TX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
4	TXFE	R	Status of Transmit Empty Trigger interrupt. This bit specifies whether the TX FIFO trigger level has reached or not. TX FIFO is empty. **Values:** ■ 0x0 (REACHED_TRIGGER_LEVEL): TX FIFO trigger level is reached ■ 0x1 (NOT_REACHED): TX FIFO trigger level is not reached **Value After Reset:** 0x1 **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1) **Volatile:** true
3:2	RSVD3_2	R	RSVD3_2 Reserved bits - Read Only **Exists:** Always **Volatile:** true
1	RXFO	R	Status of Data Overrun interrupt for the RX channel. Incoming data lost due to a full RX FIFO. **Values:** ■ 0x0 (WRITE_VALID): RX FIFO write valid ■ 0x1 (WRITE_OVERRUN): RX FIFO write overrun **Value After Reset:** 0x0 **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1) **Volatile:** true
0	RXDA	R	Status of Receive Data Available interrupt. This bit denotes the status of the RX FIFO trigger level. **Values:** ■ 0x1 (REACHED_TRIGGER_LEVEL): RX FIFO trigger level is reached ■ 0x0 (NOT_REACHED): RX FIFO trigger level is not reached **Value After Reset:** 0x0 **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1) **Volatile:** true

5.1.17 IMRx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Interrupt Mask Register x
- **Description:** This specifies the Interrupt Mask Register.
- **Size:** 32 bits
- **Offset:** 0x03C + 0x40*x
- **Exists:** ((I2S_TX_CHANNELS>x) ? 1 : ((I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK) ? 1 : 0))

31:6	5	4	3:2	1	0
RSVD_IMR0_6_31	TXFOM	TXFEM	RSVD_IMR0_2_3	RXFOM	RXDAM

Table 5-22 Fields for Register: IMRx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:6	RSVD_IMR0_6_31	R	RSVD_IMR0_6_31 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
5	TXFOM	(I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1) ? read-write : read-only	Mask TX FIFO Overrun interrupt. This bit masks or unmasks a TX FIFO overrun interrupt. **Values:** ■ 0x1 (MASK_INTERRUPT): Masks TX FIFO Overrun interrupt ■ 0x0 (UNMASK_INTERRUPT): Unmasks TX FIFO Overrun interrupt **Value After Reset:** I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1 **Exists:** I2S_TX_CHANNELS > x

Table 5-22 Fields for Register: IMRx (for x = 0; x <= I2S_TX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
4	TXFEM	(I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1) ? read-write : read-only	Mask TX FIFO Empty interrupt. This bit masks or unmasks a TX FIFO Empty interrupt. **Values:** ■ 0x1 (MASK_INTERRUPT): Masks TX FIFO Empty interrupt ■ 0x0 (UNMASK_INTERRUPT): Unmasks TX FIFO Empty interrupt **Value After Reset:** I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1 **Exists:** I2S_TX_CHANNELS > x
3:2	RSVD_IMR0_2_3	R	RSVD_IMRO_2_3 Reserved bits - Read Only **Exists:** Always
1	RXFOM	(I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1) ? read-write : read-only	Mask RX FIFO Overrun interrupt. This bit masks or unmasks an RX FIFO Overrun interrupt. **Values:** ■ 0x1 (MASK_INTERRUPT): Masks RX FIFO Overrun interrupt ■ 0x0 (UNMASK_INTERRUPT): Unmasks RX FIFO Overrun interrupt **Value After Reset:** I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1 **Exists:** I2S_RX_CHANNELS > x
0	RXDAM	(I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1) ? read-write : read-only	Mask RX FIFO Data Available interrupt. This bit masks or unmasks an RX FIFO Data Available interrupt. **Values:** ■ 0x1 (MASK_INTERRUPT): Masks RX FIFO data available interrupt ■ 0x0 (UNMASK_INTERRUPT): Unmasks RX FIFO data available interrupt **Value After Reset:** I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1 **Exists:** I2S_RX_CHANNELS > x

5.1.18 RORx (for x = 0; x <= I2S_RX_CHANNELS-1)

- **Name:** Receive Overrun Register x
- **Description:** This specifies the Receive Overrun Register.
- **Size:** 32 bits
- **Offset:** 0x040 + 0x40*x
- **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK)

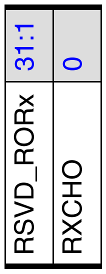


Table 5-23 Fields for Register: RORx (for x = 0; x <= I2S_RX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:1	RSVD_RORx	R	RSVD_RORx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RXCHO	R	Read this bit to clear the RX FIFO Data Overrun interrupt. **Values:** ■ 0x0 (WRITE_VALID): RX FIFO write valid ■ 0x1 (WRITE_OVERRUN): RX FIFO write overrun **Value After Reset:** 0x0 **Exists:** I2S_RX_CHANNELS > x **Volatile:** true

5.1.19 TORx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Transmit Overrun Register x
- **Description:** This specifies the Transmit Overrun Register.
- **Size:** 32 bits
- **Offset:** 0x044 + 0x40*x
- **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1)

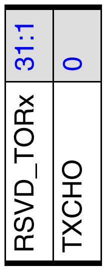


Table 5-24 Fields for Register: TORx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:1	RSVD_TORx	R	RSVD_TORx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	TXCHO	R	Read this bit to clear the TX FIFO Data Overrun interrupt. **Values:** ■ 0x0 (WRITE_VALID): TX FIFO write valid ■ 0x1 (WRITE_OVERRUN): TX FIFO write overrun **Value After Reset:** 0x0 **Exists:** I2S_TX_CHANNELS > x **Volatile:** true

5.1.20 RFCRx (for x = 0; x <= I2S_RX_CHANNELS-1)

- **Name:** Receive FIFO Configuration Register x
- **Description:** This specifies the Receive FIFO Configuration Register.
- **Size:** 32 bits
- **Offset:** 0x048 + 0x40*x
- **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1)

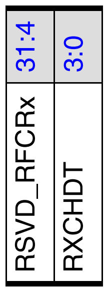


Table 5-25 Fields for Register: RFCRx (for x = 0; x <= I2S_RX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:4	RSVD_RFCRx	R	RSVD_RFCRx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: RFCRx (for x = 0; x <= I2S_RX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
3:0	RXCHDT	R/W	These bits program the trigger level in the RX FIFO at which the Received Data Available interrupt and DMA request is generated. Trigger Level = Programmed Value + 1 (for example, 1 to I2S_RX_FIFO_DEPTH_0) Valid RXCHDT values: 0 to (I2S_RX_FIFO_0 - 1) If an illegal value is programmed, these bits saturate to (I2S_RX_FIFO_0 - 1). The channel must be disabled prior to any changes in this value (that is, RERx[0] = 0). **Values:** ■ 0x0 (TRIGGER_LEVEL_1): Interrupt trigger and DMA request asserted when FIFO level is 1. ■ 0x1 (TRIGGER_LEVEL_2): Interrupt trigger and DMA request asserted when FIFO level is 2. ■ 0x2 (TRIGGER_LEVEL_3): Interrupt trigger and DMA request asserted when FIFO level is 3. ■ 0x3 (TRIGGER_LEVEL_4): Interrupt trigger and DMA request asserted when FIFO level is 4. ■ 0x4 (TRIGGER_LEVEL_5): Interrupt trigger and DMA request asserted when FIFO level is 5. ■ 0x5 (TRIGGER_LEVEL_6): Interrupt trigger and DMA request asserted when FIFO level is 6. ■ 0x6 (TRIGGER_LEVEL_7): Interrupt trigger and DMA request asserted when FIFO level is 7. ■ 0x7 (TRIGGER_LEVEL_8): Interrupt trigger and DMA request asserted when FIFO level is 8. ■ 0x8 (TRIGGER_LEVEL_9): Interrupt trigger and DMA request asserted when FIFO level is 9. ■ 0x9 (TRIGGER_LEVEL_10): Interrupt trigger and DMA request asserted when FIFO level is 10. ■ 0xa (TRIGGER_LEVEL_11): Interrupt trigger and DMA request asserted when FIFO level is 11. ■ 0xb (TRIGGER_LEVEL_12): Interrupt trigger and DMA request asserted when FIFO level is 12.

Table 5-25 Fields for Register: RFCRx (for x = 0; x <= I2S_RX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
			■ 0xc (TRIGGER_LEVEL_13): Interrupt trigger and DMA request asserted when FIFO level is 13. ■ 0xd (TRIGGER_LEVEL_14): Interrupt trigger and DMA request asserted when FIFO level is 14. ■ 0xe (TRIGGER_LEVEL_15): Interrupt trigger and DMA request asserted when FIFO level is 15. ■ 0xf (TRIGGER_LEVEL_16): Interrupt trigger and DMA request asserted when FIFO level is 16. **Value After Reset:** I2S_RX_FIFO_THRE_[x] **Exists:** I2S_RX_CHANNELS > x

5.1.21 TFCRx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Transmit FIFO Configuration Register x
- **Description:** This specifies the Transmit FIFO Configuration Register.
- **Size:** 32 bits
- **Offset:** 0x04C + 0x40*x
- **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1

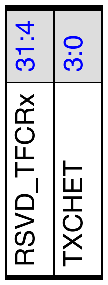


Table 5-26 Fields for Register: TFCRx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:4	RSVD_TFCRx	R	RSVD_TFCRx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-26 Fields for Register: TFCRx (for x = 0; x <= I2S_TX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
3:0	TXCHET	R/W	These bits program the trigger level in the TX FIFO at which the Empty Threshold Reached Interrupt and DMA request is generated. Trigger Level = TXCHET TXCHET values: 0 to (I2S_TX_FIFO_0 - 1) If an illegal value is programmed, these bits saturate to (I2S_TX_FIFO_0 - 1). The channel must be disabled prior to any changes in this value (that is, TER0[0] = 0). **Values:** ■ 0x0 (TRIGGER_LEVEL_1): Interrupt trigger and DMA request asserted when FIFO level is 1. ■ 0x1 (TRIGGER_LEVEL_2): Interrupt trigger and DMA request asserted when FIFO level is 2. ■ 0x2 (TRIGGER_LEVEL_3): Interrupt trigger and DMA request asserted when FIFO level is 3. ■ 0x3 (TRIGGER_LEVEL_4): Interrupt trigger and DMA request asserted when FIFO level is 4. ■ 0x4 (TRIGGER_LEVEL_5): Interrupt trigger and DMA request asserted when FIFO level is 5. ■ 0x5 (TRIGGER_LEVEL_6): Interrupt trigger and DMA request asserted when FIFO level is 6. ■ 0x6 (TRIGGER_LEVEL_7): Interrupt trigger and DMA request asserted when FIFO level is 7. ■ 0x7 (TRIGGER_LEVEL_8): Interrupt trigger and DMA request asserted when FIFO level is 8. ■ 0x8 (TRIGGER_LEVEL_9): Interrupt trigger and DMA request asserted when FIFO level is 9. ■ 0x9 (TRIGGER_LEVEL_10): Interrupt trigger and DMA request asserted when FIFO level is 10. ■ 0xa (TRIGGER_LEVEL_11): Interrupt trigger and DMA request asserted when FIFO level is 11. ■ 0xb (TRIGGER_LEVEL_12): Interrupt trigger and DMA request asserted when FIFO level is 12.

Table 5-26 Fields for Register: TFCRx (for x = 0; x <= I2S_TX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
			■ 0xc (TRIGGER_LEVEL_13): Interrupt trigger and DMA request asserted when FIFO level is 13. ■ 0xd (TRIGGER_LEVEL_14): Interrupt trigger and DMA request asserted when FIFO level is 14. ■ 0xe (TRIGGER_LEVEL_15): Interrupt trigger and DMA request asserted when FIFO level is 15. ■ 0xf (TRIGGER_LEVEL_16): Interrupt trigger and DMA request asserted when FIFO level is 16. **Value After Reset:** I2S_TX_FIFO_THRE_[x] **Exists:** I2S_TX_CHANNELS > x

5.1.22 RFFx (for x = 0; x <= I2S_RX_CHANNELS-1)

- **Name:** Receive FIFO Flush Register x
- **Description:** This specifies the Receive FIFO Flush Register.
- **Size:** 32 bits
- **Offset:** 0x050 + 0x40*x
- **Exists:** (I2S_RX_CHANNELS>x && I2S_RECEIVER_BLOCK==1)

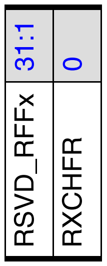


Table 5-27 Fields for Register: RFFx (for x = 0; x <= I2S_RX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:1	RSVD_RFFx	W	RSVD_RFFx Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RXCHFR	W	Receive Channel FIFO Reset. Writing a 1 to this register flushes an individual RX FIFO (This is a self clearing bit.). A Rx channel or block must be disabled prior to writing to this bit. **Values:** ■ 0x0 (NO_FLUSH): Does not flush an individual RX FIFO ■ 0x1 (FLUSH): Flushes an individual RX FIFO **Value After Reset:** 0x0 **Exists:** I2S_RX_CHANNELS > x **Volatile:** true

5.1.23 TFFx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Transmit FIFO Flush Register x
- **Description:** This specifies the Transmit FIFO Flush Register.
- **Size:** 32 bits
- **Offset:** 0x054 + 0x40*x
- **Exists:** (I2S_TX_CHANNELS>x && I2S_TRANSMITTER_BLOCK==1)

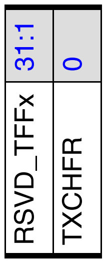


Table 5-28 Fields for Register: TFFx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:1	RSVD_TFFx	W	RSVD_TFFx Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	TXCHFR	W	Transmit Channel FIFO Reset. Writing a 1 to this register flushes channel's TX FIFO (This is a self clearing bit.). The TX channel or block must be disabled prior to writing to this bit. **Values:** ■ 0x0 (NO_FLUSH): Do not flushes channel's TX FIFO. ■ 0x1 (FLUSH): Flushes channel's TX FIFO. **Value After Reset:** 0x0 **Exists:** I2S_TX_CHANNELS > x **Volatile:** true

5.1.24 RXDMA

- **Name:** Receiver Block DMA Register
- **Description:** The RXDMA register allows access to all enabled Receive channels via a single point rather than through the LRBRx and RRBRx registers. The Receive channels are targeted in a cyclical fashion (starting at the lowest numbered enabled channel) and takes two reads (left and right stereo data) before the component points to the next channel.

  The following example describes the behavior of this register for a component that has been configured with four Receive channels, where Channels 0 and 3 are enabled:

  Order of returned read data:

  1. Ch0 - Left Data
  2. Ch0 - Right Data
  3. Ch3 - Left Data
  4. Ch3 - Right Data
  5. Ch0 - Left Data
  6. Ch0 - Right Data, and so on

  Note: There is no read coherency logic; hence, the APB_DATA_WIDTH must be greater than or equal to the largest Receive channel word size to ensure all half data pairs can be accessed using a single read.

  Channels can be enabled or disabled during the read cycles; however, DW_apb_i2s does not support disabling a channel in the middle of a stereo pair.
- **Size:** 32 bits
- **Offset:** 0x1c0
- **Exists:** I2S_RECEIVER_BLOCK==1

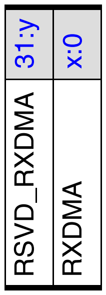


Table 5-29 Fields for Register: RXDMA

Bits	Name	Memory Access	Description
31:y	RSVD_RXDMA	R	RSVD_RXDMA Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** APB_DATA_WIDTH
x:0	RXDMA	R	Receiver Block DMA Register. These bits are used to cycle repeatedly through the enabled receive channels (from lowest numbered to highest), reading stereo data pairs. **Value After Reset:** 0x0 **Exists:** I2S_RECEIVER_BLOCK==1 **Volatile:** true **Range Variable[x]:** APB_DATA_WIDTH - 1

5.1.25 RRXDMA

- **Name:** Reset Receiver Block DMA Register
- **Description:** The RXDMA can be reset to the lowest enabled Channel via the RRXDMA register. The RRXDMA register can be written to at any stage of the RXDMA's read cycle, however, it has no effect when the component is in the middle of a stereo pair read. The following example describes the operation of this register for a system with four Receive channels, where channels 0, 1, 2, and 3 are enabled.

  Order of returned read data:

  1. Ch0 - Left Data
  2. Ch0 - Right Data
  3. RRXDMA Reset
  4. Ch0 - Left Data
  5. Ch0 - Right Data
  6. Ch1 - Left Data
  7. RRXDMA Reset - No effect (read not complete)
  8. Ch1 - Right Data, etc.
  9. Ch2 - Left Data
  10. Ch2 - Right Data
  11. RRXDMA Reset
  12. Ch0 - Left Data
  13. Ch0 - Right Data
- **Size:** 32 bits
- **Offset:** 0x1c4
- **Exists:** ((I2S_RX_CHANNELS>1) ? (I2S_RECEIVER_BLOCK) : 0)

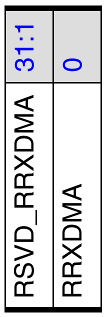


Table 5-30 Fields for Register: RRXDMA

Bits	Name	Memory Access	Description
31:1	RSVD_RRXDMA	W	RSVD_RRXDMA Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RRXDMA	W	Reset Receiver Block DMA Register. Writing a 1 to this self-clearing register resets the RXDMA register mid-cycle to point to the lowest enabled Receive channel. **Note**: Writing to this register has no effect if the component is performing a stereo pair read (such as, when left stereo data has been read but not right stereo data). **Values:** ■ 0x1 (RESET): Reset Receiver Block DMA Register **Value After Reset:** 0x0 **Exists:** I2S_RX_CHANNELS>1 && I2S_RECEIVER_BLOCK==1 **Volatile:** true

5.1.26 TXDMA

- **Name:** Transmitter Block DMA Register
- **Description:** The TXDMA register functions similar to the RXDMA register and allows write accesses to all of the enabled Transmit channels via a single point rather than through the LTHRx and RTHRx registers.

  Note: There is no write coherency logic, the APB_DATA_WIDTH must be greater than or equal to the largest Transmit channel word size to ensure all half data pairs can be written using a single write.

  Channels can be enabled or disabled during the write cycles; however, DW_apb_i2s does not support disabling a channel in the middle of a stereo pair.
- **Size:** 32 bits
- **Offset:** 0x1c8
- **Exists:** I2S_TRANSMITTER_BLOCK==1

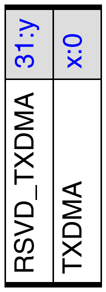


Table 5-31 Fields for Register: TXDMA

Bits	Name	Memory Access	Description
31:y	RSVD_TXDMA	W	RSVD_TXDMA Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** APB_DATA_WIDTH
x:0	TXDMA	W	Transmitter Block DMA Register. The register bits can be used to cycle repeatedly through the enabled Transmit channels (from lowest numbered to highest) to allow writing of stereo data pairs. **Value After Reset:** 0x0 **Exists:** I2S_TRANSMITTER_BLOCK==1 **Volatile:** true **Range Variable[x]:** APB_DATA_WIDTH - 1

5.1.27 RTXDMA

- **Name:** Reset Transmitter Block DMA Register
- **Description:** This register provides the same functionality as the RRXDMA register but targets TXDMA instead.
- **Size:** 32 bits
- **Offset:** 0x1cc
- **Exists:** ((I2S_TX_CHANNELS>1) ? (I2S_TRANSMITTER_BLOCK) : 0)

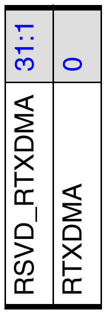


Table 5-32 Fields for Register: RTXDMA

Bits	Name	Memory Access	Description
31:1	RSVD_RTXDMA	W	RSVD_RTXDMA Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RTXDMA	W	Reset Transmitter Block DMA Register. Writing a 1 to this self-clearing register resets the TXDMA register mid-cycle to point to the lowest enabled Transmit channel. **Note**: This register has no effect in the middle of a stereo pair write (such as, when left stereo data has been written but not right stereo data). **Values:** ■ 0x1 (RESET): Reset Transmitter Block DMA Register **Value After Reset:** 0x0 **Exists:** I2S_TX_CHANNELS>1 && I2S_TRANSMITTER_BLOCK==1 **Volatile:** true

5.1.28 I2S_COMP_PARAM_2

- **Name:** Component Parameter Register 2
- **Description:** This specifies bits for Component Parameter Register 2.

  Note: This is a constant read-only register that contains encoded information about the component's parameter settings. The reset value depends on coreConsultant parameter(s).
- **Size:** 32 bits
- **Offset:** 0x1f0
- **Exists:** Always

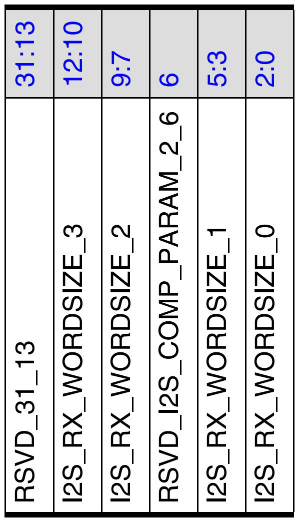


Table 5-33 Fields for Register: I2S_COMP_PARAM_2

Bits	Name	Memory Access	Description
31:13	RSVD_31_13	R	RSVD_31_13 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-33 Fields for Register: I2S_COMP_PARAM_2 (Continued)

Bits	Name	Memory Access	Description
12:10	I2S_RX_WORDSIZE_3	R	These bits specify the RX resolution for WORDSIZE_3. **Values:** ■ 0x0 (RESOLUTION_12_BIT): 12-bit Resolution ■ 0x1 (RESOLUTION_16_BIT): 16-bit Resolution ■ 0x2 (RESOLUTION_20_BIT): 20-bit Resolution ■ 0x3 (RESOLUTION_24_BIT): 24-bit Resolution ■ 0x4 (RESOLUTION_32_BIT): 32-bit Resolution **Value After Reset:** ENCODED_I2S_RX_WORDSIZE_3 **Exists:** Always
9:7	I2S_RX_WORDSIZE_2	R	These bits specify the RX resolution for WORDSIZE_2. **Values:** ■ 0x0 (RESOLUTION_12_BIT): 12-bit Resolution ■ 0x1 (RESOLUTION_16_BIT): 16-bit Resolution ■ 0x2 (RESOLUTION_20_BIT): 20-bit Resolution ■ 0x3 (RESOLUTION_24_BIT): 24-bit Resolution ■ 0x4 (RESOLUTION_32_BIT): 32-bit Resolution **Value After Reset:** ENCODED_I2S_RX_WORDSIZE_2 **Exists:** Always
6	RSVD_I2S_COMP_PARAM_2_6	R	RSVD_I2S_COMP_PARAM_2_6 Reserved bits - Read Only **Exists:** Always
5:3	I2S_RX_WORDSIZE_1	R	These bits specify the RX resolution for WORDSIZE_1. **Values:** ■ 0x0 (RESOLUTION_12_BIT): 12-bit Resolution ■ 0x1 (RESOLUTION_16_BIT): 16-bit Resolution ■ 0x2 (RESOLUTION_20_BIT): 20-bit Resolution ■ 0x3 (RESOLUTION_24_BIT): 24-bit Resolution ■ 0x4 (RESOLUTION_32_BIT): 32-bit Resolution **Value After Reset:** ENCODED_I2S_RX_WORDSIZE_1 **Exists:** Always

Table 5-33 Fields for Register: I2S_COMP_PARAM_2 (Continued)

Bits	Name	Memory Access	Description
2:0	I2S_RX_WORDSIZE_0	R	These bits specify the RX resolution for WORDSIZE_0. **Values:** ■ 0x0 (RESOLUTION_12_BIT): 12-bit Resolution ■ 0x1 (RESOLUTION_16_BIT): 16-bit Resolution ■ 0x2 (RESOLUTION_20_BIT): 20-bit Resolution ■ 0x3 (RESOLUTION_24_BIT): 24-bit Resolution ■ 0x4 (RESOLUTION_32_BIT): 32-bit Resolution **Value After Reset:** ENCODED_I2S_RX_WORDSIZE_0 **Exists:** Always

5.1.29 I2S_COMP_PARAM_1

- **Name:** Component Parameter Register 1
- **Description:** This specifies bits for Component Parameter Register 1.

  Note: This is a constant read-only register that contains encoded information about the component's parameter settings. The reset value depends on coreConsultant parameter(s).
- **Size:** 32 bits
- **Offset:** 0x1f4
- **Exists:** Always

31:28	27:25	24:22	21:19	18:16	15:11	10:9	8:7	6	5	4	3:2	1:0
RSVD_PARAM_1_28_31	I2S_TX_WORDSIZE_3	I2S_TX_WORDSIZE_2	I2S_TX_WORDSIZE_1	I2S_TX_WORDSIZE_0	RSVD_PARAM_1_11_15	I2S_TX_CHANNELS	I2S_RX_CHANNELS	I2S_RECEIVER_BLOCK	I2S_TRANSMITTER_BLOCK	I2S_MODE_EN	I2S_FIFO_DEPTH_GLOBAL	APB_DATA_WIDTH

Table 5-34 Fields for Register: I2S_COMP_PARAM_1

Bits	Name	Memory Access	Description
31:28	RSVD_PARAM_1_28_31	R	RSVD_I2S_COMP_PARAM_1_28_31 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-34 Fields for Register: I2S_COMP_PARAM_1 (Continued)

Bits	Name	Memory Access	Description
27:25	I2S_TX_WORDSIZE_3	R	These bits specify the TX resolution for WORDSIZE_3. **Values:** ■ 0x0 (RESOLUTION_12_BIT): 12-bit Resolution ■ 0x1 (RESOLUTION_16_BIT): 16-bit Resolution ■ 0x2 (RESOLUTION_20_BIT): 20-bit Resolution ■ 0x3 (RESOLUTION_24_BIT): 24-bit Resolution ■ 0x4 (RESOLUTION_32_BIT): 32-bit Resolution **Value After Reset:** ENCODED_I2S_TX_WORDSIZE_3 **Exists:** Always
24:22	I2S_TX_WORDSIZE_2	R	These bits specify the TX resolution for WORDSIZE_2. **Values:** ■ 0x0 (RESOLUTION_12_BIT): 12-bit Resolution ■ 0x1 (RESOLUTION_16_BIT): 16-bit Resolution ■ 0x2 (RESOLUTION_20_BIT): 20-bit Resolution ■ 0x3 (RESOLUTION_24_BIT): 24-bit Resolution ■ 0x4 (RESOLUTION_32_BIT): 32-bit Resolution **Value After Reset:** ENCODED_I2S_TX_WORDSIZE_2 **Exists:** Always
21:19	I2S_TX_WORDSIZE_1	R	These bits specify the TX resolution for WORDSIZE_1. **Values:** ■ 0x0 (RESOLUTION_12_BIT): 12-bit Resolution ■ 0x1 (RESOLUTION_16_BIT): 16-bit Resolution ■ 0x2 (RESOLUTION_20_BIT): 20-bit Resolution ■ 0x3 (RESOLUTION_24_BIT): 24-bit Resolution ■ 0x4 (RESOLUTION_32_BIT): 32-bit Resolution **Value After Reset:** ENCODED_I2S_TX_WORDSIZE_1 **Exists:** Always
18:16	I2S_TX_WORDSIZE_0	R	These bits specify the TX resolution for WORDSIZE_0. **Values:** ■ 0x0 (RESOLUTION_12_BIT): 12-bit Resolution ■ 0x1 (RESOLUTION_16_BIT): 16-bit Resolution ■ 0x2 (RESOLUTION_20_BIT): 20-bit Resolution ■ 0x3 (RESOLUTION_24_BIT): 24-bit Resolution ■ 0x4 (RESOLUTION_32_BIT): 32-bit Resolution **Value After Reset:** ENCODED_I2S_TX_WORDSIZE_0 **Exists:** Always

Table 5-34 Fields for Register: I2S_COMP_PARAM_1 (Continued)

Bits	Name	Memory Access	Description
15:11	RSVD_PARAM_1_11_15	R	RSVD_I2S_COMP_PARAM_1_11_15 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
10:9	I2S_TX_CHANNELS	R	These bits specify the number of TX channels. **Values:** ■ 0x0 (TX_CHANNEL_1): 1 Transmit Channel ■ 0x1 (TX_CHANNEL_2): 2 Transmit Channels ■ 0x2 (TX_CHANNEL_3): 3 Transmit Channels ■ 0x3 (TX_CHANNEL_4): 4 Transmit Channels **Value After Reset:** ENCODED_I2S_TX_CHANNELS **Exists:** Always
8:7	I2S_RX_CHANNELS	R	These bits specify the number of RX channels. **Values:** ■ 0x0 (RX_CHANNEL_1): 1 Receive Channel ■ 0x1 (RX_CHANNEL_2): 2 Receive Channels ■ 0x2 (RX_CHANNEL_3): 3 Receive Channels ■ 0x3 (RX_CHANNEL_4): 4 Receive Channels **Value After Reset:** ENCODED_I2S_RX_CHANNELS **Exists:** Always
6	I2S_RECEIVER_BLOCK	R	This bit specifies whether the receiver block is enabled or not. **Values:** ■ 0x0 (FALSE): Receiver block is disabled ■ 0x1 (TRUE): Receiver block is enabled **Value After Reset:** I2S_RECEIVER_BLOCK **Exists:** Always
5	I2S_TRANSMITTER_BLOCK	R	This bit specifies whether the transmitter block is enabled or not. **Values:** ■ 0x0 (FALSE): Transmitter block is disabled ■ 0x1 (TRUE): Transmitter block is enabled **Value After Reset:** I2S_TRANSMITTER_BLOCK **Exists:** Always

Table 5-34 Fields for Register: I2S_COMP_PARAM_1 (Continued)

Bits	Name	Memory Access	Description
4	I2S_MODE_EN	R	This bit specifies whether the master mode is enabled or not. **Values:** ■ 0x0 (FALSE): Master mode is disabled ■ 0x1 (TRUE): Master mode is enabled **Value After Reset:** I2S_MODE_EN **Exists:** Always
3:2	I2S_FIFO_DEPTH_GLOBAL	R	These bits specify the FIFO depth for TX and RX channels. **Values:** ■ 0x0 (FIFO_DEPTH_2): FIFO depth is equals to 2 for TX and RX channels ■ 0x1 (FIFO_DEPTH_4): FIFO depth is equals to 4 for TX and RX channels ■ 0x2 (FIFO_DEPTH_8): FIFO depth is equals to 8 for TX and RX channels ■ 0x3 (FIFO_DEPTH_16): FIFO depth is equals to 16 for TX and RX channels **Value After Reset:** ENCODED_I2S_FIFO_DEPTH_GLOBAL **Exists:** Always
1:0	APB_DATA_WIDTH	R	These bits specify the APB data width. **Values:** ■ 0x0 (BITS_8): 8 bits APB data width ■ 0x0 (BITS_16): 16 bits APB data width ■ 0x0 (BITS_32): 32 bits APB data width **Value After Reset:** ENCODED_APB_DATA_WIDTH **Exists:** Always

5.1.30 I2S_COMP_VERSION

- **Name:** I2S Component Version Register
- **Description:** This register specifies the I2S Component Version.
- **Size:** 32 bits
- **Offset:** 0x1f8
- **Exists:** Always

31:0
I2S_COMP_VERSION

Table 5-35 Fields for Register: I2S_COMP_VERSION

Bits	Name	Memory Access	Description
31:0	I2S_COMP_VERSION	R	These bits specify the I2S component version. The value for I2S_COMP_VERSION are described in the "DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI Release Notes". **Value After Reset:** I2S_COMP_VERSION **Exists:** Always

5.1.31 I2S_COMP_TYPE

- **Name:** I2S Component Type Register
- **Description:** This register specifies the I2S Component Type.
- **Size:** 32 bits
- **Offset:** 0x1fc
- **Exists:** Always

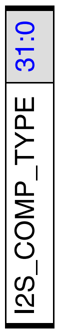


Table 5-36 Fields for Register: I2S_COMP_TYPE

Bits	Name	Memory Access	Description
31:0	I2S_COMP_TYPE	R	DesignWare Component Type number = 0x445701a0. This unique hexadecimal value is constant and is derived from the two ASCII letters 'DW' followed by a 16-bit unsigned number. **Value After Reset:** I2S_COMP_TYPE **Exists:** Always

5.1.32 DMACR

- **Name:** DMA Control Register
- **Description:** This register is only valid when DW_apb_i2s is configured with a set of DMA Controller interface signals (I2S_HAS_DMA_INTERFACE = 1). When DW_apb_i2s is not configured with DMA handshake interface, this register will not exist and writing to the register?s address will have no effect and reading from this register address will return zero. The register is used to enable the DMA Controller interface operation.
- **Size:** 32 bits
- **Offset:** 0x200
- **Exists:** I2S_HAS_DMA_INTERFACE==1

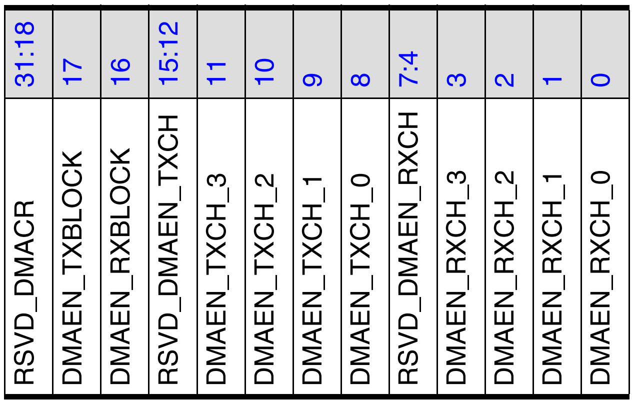


Table 5-37 Fields for Register: DMACR

Bits	Name	Memory Access	Description
31:18	RSVD_DMACR	R	DMACR Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always
17	DMAEN_TXBLOCK	R/W	DMA Enable for transmit block. The corresponding bits of this field enables/disables the DMA handshake logic for transmitter block. **Values:** ■ 0x0 (DISABLED): DMA disabled for transmit block ■ 0x1 (ENABLED): DMA enabled for transmit block **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==1 && I2S_TRANSMITTER_BLOCK==1)

Table 5-37 Fields for Register: DMACR (Continued)

Bits	Name	Memory Access	Description
16	DMAEN_RXBLOCK	R/W	DMA Enable for receive block. The corresponding bits of this field enables/disables the DMA handshake logic for receiver block **Values:** ■ 0x0 (DISABLED): DMA disabled for receiver block ■ 0x1 (ENABLED): DMA enabled for receiver block **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==1 && I2S_RECEIVER_BLOCK==1)
15:12	RSVD_DMAEN_TXCH	R	Reserved bits for transmit channel DMA Enable - Read Only. **Value After Reset:** 0x0 **Exists:** Always
11	DMAEN_TXCH_3	R/W	DMA Enable for transmit channel 3. The corresponding bits of this field enables/disables the transmit FIFO DMA for channel 3. **Values:** ■ 0x0 (DISABLED): DMA disabled for transmit channel 3 ■ 0x1 (ENABLED): DMA enabled for transmit channel 3 **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_TX_CHANNELS>3 && I2S_TRANSMITTER_BLOCK==1)
10	DMAEN_TXCH_2	R/W	DMA Enable for transmit channel 2. The corresponding bits of this field enables/disables the transmit FIFO DMA for channel 2. **Values:** ■ 0x0 (DISABLED): DMA disabled for transmit channel 2 ■ 0x1 (ENABLED): DMA enabled for transmit channel 2 **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_TX_CHANNELS>2 && I2S_TRANSMITTER_BLOCK==1)

Table 5-37 Fields for Register: DMACR (Continued)

Bits	Name	Memory Access	Description
9	DMAEN_TXCH_1	R/W	DMA Enable for transmit channel 1. The corresponding bits of this field enables/disables the transmit FIFO DMA for channel 1. **Values:** ■ 0x0 (DISABLED): DMA disabled for transmit channel 1 ■ 0x1 (ENABLED): DMA enabled for transmit channel 1 **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_TX_CHANNELS>1 && I2S_TRANSMITTER_BLOCK==1)
8	DMAEN_TXCH_0	R/W	DMA Enable for transmit channel 0. The corresponding bits of this field enables/disables the transmit FIFO DMA for channel 0. **Values:** ■ 0x0 (DISABLED): DMA disabled for transmit channel 0 ■ 0x1 (ENABLED): DMA enabled for transmit channel 0 **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_TRANSMITTER_BLOCK==1)
7:4	RSVD_DMAEN_RXCH	R	Reserved bits for receive channel DMA Enable - Read Only. **Value After Reset:** 0x0 **Exists:** Always
3	DMAEN_RXCH_3	R/W	DMA Enable for receive channel 3. The corresponding bits of this field enables/disables the receive FIFO DMA for channel 3. **Values:** ■ 0x0 (DISABLED): DMA disabled for receive channel 3 ■ 0x1 (ENABLED): DMA enabled for receive channel 3 **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_RX_CHANNELS>3 && I2S_RECEIVER_BLOCK==1)

Table 5-37 Fields for Register: DMACR (Continued)

Bits	Name	Memory Access	Description
2	DMAEN_RXCH_2	R/W	DMA Enable for receive channel 2. The corresponding bits of this field enables/disables the receive FIFO DMA for channel 2. **Values:** ■ 0x0 (DISABLED): DMA disabled for receive channel 2 ■ 0x1 (ENABLED): DMA enabled for receive channel 2 **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_RX_CHANNELS>2 && I2S_RECEIVER_BLOCK==1)
1	DMAEN_RXCH_1	R/W	DMA Enable for receive channel 1. The corresponding bits of this field enables/disables the receive FIFO DMA for channel 1. **Values:** ■ 0x0 (DISABLED): DMA disabled for receive channel 1 ■ 0x1 (ENABLED): DMA enabled for receive channel 1 **Value After Reset:** 0x0 **Exists:** (I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_RX_CHANNELS>1 && I2S_RECEIVER_BLOCK==1)
0	DMAEN_RXCH_0	R/W	DMA Enable for receive channel 0. The corresponding bits of this field enables/disables the receive FIFO DMA for channel 0. **Values:** ■ 0x0 (DISABLED): DMA disabled for receive channel 0 ■ 0x1 (ENABLED): DMA enabled for receive channel 0 **Value After Reset:** 0x0 **Exists:** I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_RECEIVER_BLOCK==1

5.1.33 RXDMA_CHx (for x = 0; x <= I2S_RX_CHANNELS-1)

- **Name:** Receiver Block DMA Register
- **Description:** The RXDMA_CHx register allows access to enabled Receive channel x via a single point rather than through the LRBRx and RRBRx registers. This register is available only if I2S has dedicated DMA handshaking interface enabled for channel x.

  Note: There is no read coherency logic; hence, the APB_DATA_WIDTH must be greater than or equal to the largest Receive channel word size to ensure all half data pairs can be accessed using a single read.

  Channels can be enabled or disabled during the read cycles; however, it is recommended not to disable a channel in the middle of a stereo pair.
- **Size:** 32 bits
- **Offset:** 0x204
- **Exists:** I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_RECEIVER_BLOCK==1 && I2S_RX_CHANNELS>x

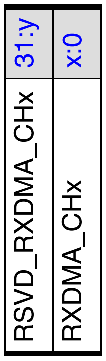


Table 5-38 Fields for Register: RXDMA_CHx (for x = 0; x <= I2S_RX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:y	RSVD_RXDMA_CHx	R	RXDMA_CHx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** APB_DATA_WIDTH

Table 5-38 Fields for Register: RXDMA_CHx (for x = 0; x <= I2S_RX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
x:0	RXDMA_CHx	R	Receiver Block DMA Register for channel x. These bits are used for reading stereo data pairs. **Value After Reset:** 0x0 **Exists:** I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_RECEIVER_BLOCK==1 && I2S_RX_CHANNELS>x **Volatile:** true **Range Variable[x]:** APB_DATA_WIDTH - 1

5.1.34 TXDMA_CHx (for x = 0; x <= I2S_TX_CHANNELS-1)

- **Name:** Receiver Block DMA Register
- **Description:** The TXDMA_CHx register allows access to enabled Transmit channel x via a single point rather than through the LTHRx and RTHRx registers. This register is available only if I2S has dedicated DMA handshaking interface enabled for channel x.

  Note: There is no read coherency logic; hence, the APB_DATA_WIDTH must be greater than or equal to the largest Transmit channel word size to ensure all half data pairs can be accessed using a single read.

  Channels can be enabled or disabled during the read cycles; however, it is recommended not to disable a channel in the middle of a stereo pair.
- **Size:** 32 bits
- **Offset:** 0x214
- **Exists:** I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_TRANSMITTER_BLOCK==1 && I2S_TX_CHANNELS>x

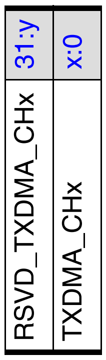


Table 5-39 Fields for Register: TXDMA_CHx (for x = 0; x <= I2S_TX_CHANNELS-1)

Bits	Name	Memory Access	Description
31:y	RSVD_TXDMA_CHx	W	TXDMA_CHx Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** APB_DATA_WIDTH

Table 5-39 Fields for Register: TXDMA_CHx (for x = 0; x <= I2S_TX_CHANNELS-1) (Continued)

Bits	Name	Memory Access	Description
x:0	TXDMA_CHx	W	Receiver Block DMA Register for channel x. These bits are used for reading stereo data pairs. **Value After Reset:** 0x0 **Exists:** I2S_HAS_DMA_INTERFACE==1 && I2S_DMA_HS_TYPE==0 && I2S_TRANSMITTER_BLOCK==1 && I2S_TX_CHANNELS>x **Volatile:** true **Range Variable[x]:** APB_DATA_WIDTH - 1

6 Programming the DW_apb_i2s

The DW_apb_i2s can be programmed through software registers, which are described in more detail in "Register Descriptions" on page 81. This chapter describes the following:

- "DW_apb_i2s as Transmitter" on page 147
- "DW_apb_i2s as Receiver" on page 148
- "DW_apb_i2s as Transmitter—With DMA Handshake Interface" on page 149
- "DW_apb_i2s as Receiver—With DMA Handshake Interface" on page 150
- "Example Configurations" on page 152

6.1 DW_apb_i2s as Transmitter

This section describes how to program the DW_apb_i2s when it is configured as a transmitter in either slave mode or master mode with one stereo channel.

6.1.1 Slave Mode

The following subsections describe normal and TX DMA sequences when DW_apb_i2s is a transmitter in slave mode.

6.1.1.1 Normal Mode

To program DW_apb_i2s when it is a transmitter in slave mode, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Fill the TX-FIFO by writing data to the Left Transmit Holding Register (LTHR) and the Right Transmit Holding Register (RTHR), respectively. Keep writing in this order—left and then right—until the FIFO is filled with data.

   The DW_apb_i2s starts transmitting stereo data on the first left cycle when the ws signal goes low.
3. Enable the I^{2}S Transmitter block by writing 1 in bit 0 (TXEN) of the ^{2}S Transmitter Block Enable Register (ITER).

6.1.1.2 TX DMA Mode

To program DW_apb_i2s when it is a transmitter in slave mode, and when TX DMA mode is enabled, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I^{2}S Transmitter block by writing a 1 in bit 0 (TXEN) of the I^{2}S Transmitter Block Enable Register (ITER).
3. Fill the TX-FIFO by writing to the Transmitter Block DMA Register (TXDMA).

6.1.2 Master Mode

To program DW_apb_i2s when it is a transmitter in master mode, complete the following steps:

1. Complete steps 1 to 3 of the previous procedure (transmitter, slave mode).
2. Enable the I^{2}S Clock Generation block by writing 1 in bit 0 (CLKEN) of the Clock Enable Register (CER).

6.2 DW_apb_i2s as Receiver

This section describe how to program the DW_apb_i2s when it is configured as a receiver in either slave mode or master mode with one stereo channel.

6.2.1 Slave Mode

To program DW_apb_i2s when it is a receiver in slave mode, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I^{2}S Receiver block by writing 1 in bit 0 (RXEN) of the I^{2}S Receiver Block Enable Register (IRER).
3. Read bit 0 (RXDA) of the Interrupt Status Register (ISR). When bit 0 of the goes high, the default trigger has been reached.
4. Read the contents of the Left Receive Buffer Register (LRBR) and Right Receive Buffer Register (RRBR).

   The bit is dependent on how you have configured (or programmed) the trigger level.

6.2.1.1 RX DMA Mode

When RX DMA is enabled, the data contents is read from RXDMA instead of the Left Receive Buffer Register (LRBR) and the Right Receive Buffer Register (RRBR), as defined in step 4 of the previous procedure (receiver, slave mode).

6.2.2 Master Mode

To program DW_apb_i2s when it is a receiver in master mode, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I^{2}S Receiver block by writing 1 in bit 0 (RXEN) of the I^{2}S Receiver Block Enable Register (IRER).

3. Enable the I^{2}S Clock Generation block by writing 1 in bit 0 (CLKEN) of the Clock Enable Register (CER).
4. Read bit 0 (RXDA) of the Interrupt Status Register (ISR). When bit 0 of the goes high, the default trigger has been reached.
5. Read the contents of the Left Receive Buffer Register (LRBR) and Right Receive Buffer Register (RRBR).

   The bit is dependent on how you have configured (or programmed) the trigger level.

6.2.2.1 RX DMA Mode

When RX DMA is enabled, the data contents is read from RXDMA instead of the Left Receive Buffer Register (LRBR) and the Right Receive Buffer Register (RRBR), as defined in step 5 of the previous procedure (receiver, master mode).

6.3 DW_apb_i2s as Transmitter—With DMA Handshake Interface

The following sections describe:

- How to program DW_apb_i2s when it is configured as the transmitter in either slave mode or master mode with one stereo channel
- How to program DW_apb_i2s for DMA controller interface operation (I2S_HAS_DMA_INTERFACE=1)
- How to fill TX FIFO using DMA handshaking interface

6.3.1 Slave Mode

The following subsections describe dedicated and combined DMA handshake sequences when DW_apb_i2s is a transmitter in slave mode.

6.3.1.1 Dedicated DMA Handshake Interface Mode (I2S_DMA_HS_TYPE=0)

To program DW_apb_i2s when it is a transmitter in slave mode, and when dedicated DMA handshake interface mode is enabled, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I^{2}S Transmitter block by writing 1 in bit 0 (TXEN) of the I^{2}S Transmitter Block Enable Register (ITER).
3. Enable DMA handshaking by writing 1 in bit 8 (DMAEN_TXCH_0) of the DMA Control Register (DMACR).
4. Fill the TX-FIFO by writing to the Transmitter Channel 0 DMA Register (TXDMA_CH0).

6.3.1.2 Combined DMA Handshake Interface Mode (I2S_DMA_HS_TYPE=1)

To program DW_apb_i2s when it is a transmitter in slave mode, and when combined DMA handshake interface mode is enabled, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.

2. Enable the I^{2}S Transmitter block by writing 1 in bit 0 (TXEN) of the I^{2}S Transmitter Block Enable Register (ITER).
3. Enable DMA handshaking by writing 1 in bit 8 (DMAEN_TXCH_0) of the DMA Control Register (DMACR).
4. Enable the Transmit channel by writing 1 in bit 0 (TXCHEN0) of the Transmit Enable Register 0 (TER0).
5. Fill the TX-FIFO by writing to the Transmitter Block DMA Register (TXDMA).

6.3.2 Master Mode

To program DW_apb_i2s when it is a transmitter in master mode, complete the following steps:

1. Complete all the steps described in "Slave Mode" on page 149.
2. Enable the I^{2}S Clock Generation block by writing 1 in bit 0 (CLKEN) of the Clock Enable Register (CER).

6.4 DW_apb_i2s as Receiver—With DMA Handshake Interface

The following flows describe:

- How to program the DW_apb_i2s when it is configured as the receiver in either slave mode or master mode with one stereo channel
- How to program DW_apb_i2s for DMA Controller interface operation (I2S_HAS_DMA_INTERFACE=1)
- How to empty RX FIFO using DMA handshaking interface

6.4.1 Slave Mode

The following subsections describe dedicated and combined DMA handshake sequences when DW_apb_i2s is a receiver in slave mode.

6.4.1.1 Dedicated DMA Handshake Interface Mode (I2S_DMA_HS_TYPE=0)

To program DW_apb_i2s when it is a receiver in slave mode, and when dedicated DMA handshake interface mode is enabled, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I^{2}S Receiver block by writing 1 in bit 0 (RXEN) of the I^{2}S Receiver Block Enable Register (IRER).
3. Enable DMA handshaking by writing 1 in bit 0 (DMAEN_RXCH_0) of the DMA Control Register (DMACR).
4. Empty the RX-FIFO by reading from the Receiver Channel 0 DMA Register (RXDMA_CH0).

6.4.1.2 Combined DMA Handshake Interface Mode (I2S_DMA_HS_TYPE=1)

To program DW_apb_i2s when it is a receiver in slave mode, and when combined DMA handshake interface mode is enabled, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I^2S Receiver block by writing 1 in bit 0 (RXEN) of the I^2S Receiver Block Enable Register (IRER).
3. Enable DMA handshaking by writing 1 in bit 8 (DMAEN_RXCH_0) of the DMA Control Register (DMACR).
4. Enable the Receiver channel by writing 1 in bit 0 (RXCHEN0) of the Receiver Enable Register 0 (RER0).
5. Empty the RX-FIFO by reading from the Receiver Block DMA Register (RXDMA).

6.4.2 Master Mode

To program DW_apb_i2s when it is a receiver in master mode, complete the following steps:

6.4.2.1 Dedicated DMA Handshake Interface Mode (I2S_DMA_HS_TYPE=0)

To program DW_apb_i2s when it is a receiver in master mode, and when dedicated DMA handshake interface mode is enabled, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I^2S Receiver block by writing 1 in bit 0 (RXEN) of the I^2S Receiver Block Enable Register (IRER).
3. Enable the I^2S Clock Generation block by writing 1 in bit 0 (CLKEN) of the Clock Enable Register (CER).
4. Enable DMA handshaking by writing 1 in bit 0 (DMAEN_RXCH_0) of the DMA Control Register (DMACR).
5. Empty the RX-FIFO by reading from the Receiver Channel 0 DMA Register (RXDMA_CH0).

6.4.2.2 Combined DMA Handshake Interface mode (I2S_DMA_HS_TYPE=1)

To program DW_apb_i2s when it is a receiver in master mode, and when combined DMA handshake interface mode is enabled, complete the following steps:

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I^2S Receiver block by writing 1 in bit 0 (RXEN) of the I^2S Receiver Block Enable Register (IRER).
3. Enable the I^2S Clock Generation block by writing 1 in bit 0 (CLKEN) of the Clock Enable Register (CER).
4. Enable DMA handshaking by writing 1 in bit 8 (DMAEN_RXCH_0) of the DMA Control Register (DMACR).
5. Enable the Receiver channel by writing 1 in bit 0 (RXCHEN0) of the Receiver Enable Register 0 (RER0).

6. Empty the RX-FIFO by reading from the Receiver Block DMA Register (RXDMA).

6.5 Example Configurations

The following examples describe how to program DW_apb_i2s for when it is configured as a receiver in slave mode with multiple channels. If you configured the component as a transmitter with the following configurations, the behavior would be similar.

Configure DW_apb_i2s using coreConsultant or coreAssembler and set the following configuration parameter:

coreConsultant Label	Configuration Parameter	Setting
Number of Receive Channels?	I2S_RX_CHANNELS	3

6.5.1 Example 1

This example illustrates the following operations:

- Clearing a RX FIFO
- Adjusting the RX FIFO data available trigger level and the word select size of one receive channel while two other channels are receiving.

The procedure for this example is as follows:

1. Enable DW_apb_i2s by writing 1 in IER[0].
2. Enable Receiver block by writing 1 in IRER[0].

   All three channels begin receiving once ws_slv goes low (ws_slv = 0)
3. Read from the channels through LRBRx and RRBRx registers, where x is the channel number.
4. Disable RX Channel 2 independently of Channel 1 and 3 by writing 0 in RER2[0].

   Incoming data for RX Channel 2 is lost, while data in the RX FIFO is preserved.
5. Clear RX Channel 2 FIFO independently of the other channels by writing 1 in RFF2[0].
6. Reprogram the RX FIFO data available trigger level by writing to RFCR2[3:0].
7. Reprogram the word select size by writing to RCR2[2:0].
8. Enable RX Channel 2 by writing 1 in RER2[0].

   DW_apb_i2s begins receiving new data when ws_slv goes low (ws_slv = 0).

Note The entire time RX Channel 2 is disabled, channels 1 and 3 are functioning normally and could be accessed.

6.5.2 Example 2

This example starts two of the three channels when the receiver block is enabled.

1. Enable DW_apb_i2s by writing 1 in IER[0].
2. Disable RX Channel 2 by writing 0 in RER2[0].
3. Enable the Receiver block by writing 1 in IRER[0].

   Only channels 1 and 3 begin receiving once ws_slv goes low.

6.5.3 Example 3

This example illustrates the following operations:

- Start receiving
- Stop receiving
- Flush all FIFOs
- Resume receiving

1. Enable DW_apb_i2s by writing 1 in IER[0].
2. Enable the Receiver block by writing 1 in IRER[0].

   All three channels begin receiving once ws_slv goes low (ws_slv = 0).
3. Read data from the channels by reading the LRBRx and RRBRx registers.
4. Disable the Receiver block by writing 0 in IRER[0].
5. Flush all RX FIFOs by writing 1 in RXFFR[0].
6. Enable the Receive block again by writing 1 in IRER[0].

   All three channels resume receiving once ws_slv goes low (ws_slv = 0).

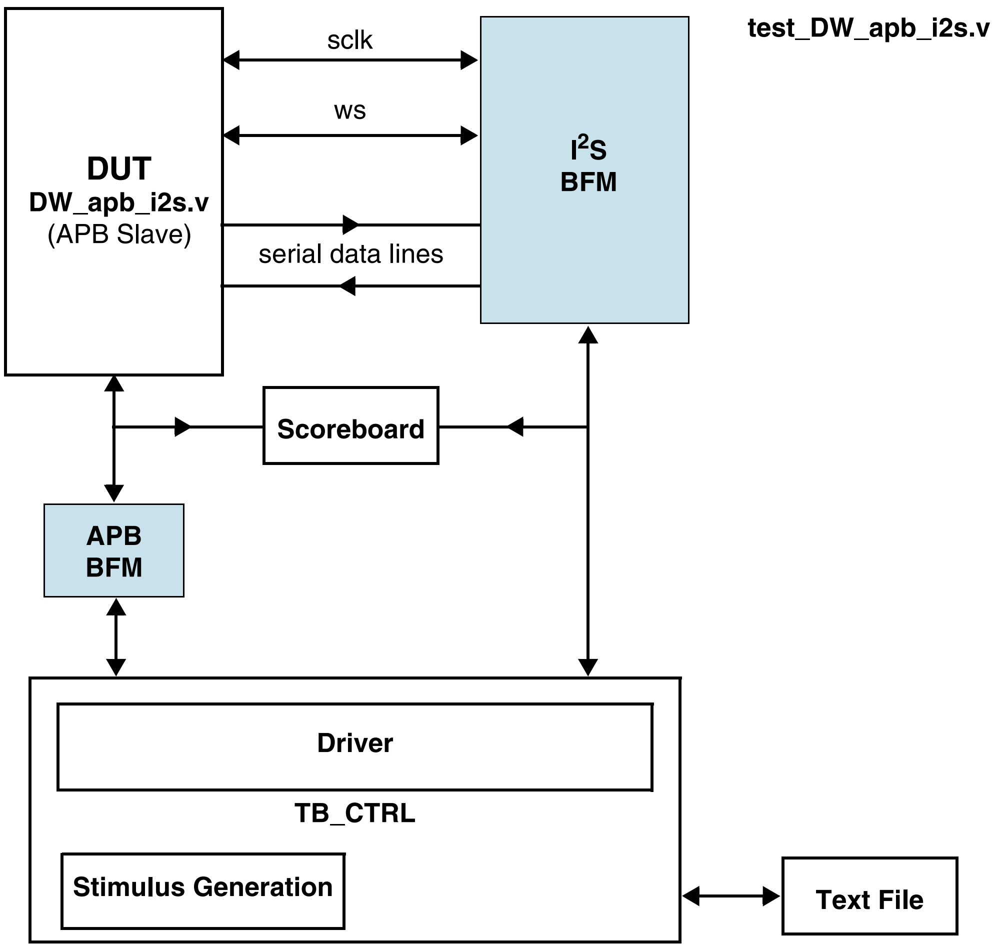
7 Verification

This chapter provides an overview of the testbench available for DW_apb_i2s verification. Once you have configured the DW_apb_i2s in coreConsultant or DesignWare coreAssembler and have set up the verification environment, you can run simulations automatically. The following sections describe the testbench.

Figure 7-1 illustrates the Verilog DW_apb_i2s testbench.

The testbench tests the user configuration specified in the Specify Configuration task of coreConsultant and Verify Component task in DesignWare coreAssembler. The testbench also tests that the component is AMBA-compliant and includes a self-checking mechanism. When a coreKit has been unpacked and configured, the verification environment is stored in *workspace*/sim. Files in *workspace*/sim/test_i2s form the actual testbench for DW_apb_i2s.

Figure 7-1 DW_apb_i2s Testbench



The DW_apb_i2s testbench consists of the following components:

- test_DW_apb_i2s.v – The test_DW_apb_i2s.v file shows the instantiation of the top-level component in a testbench and resides in the *workspace*/sim/testbench directory.
- DUT – A single instantiation of the DW_apb_i2s or DUT is required. The DUT can be configured as an I^{2}S Master or Slave. The testbench automatically wires up the DUT to form a fully functional single-link I^{2}S system.

  In order to verify the DUT operating as a I^{2}S Master or Slave, separate configurations need to be built with independent test simulations.
- I^{2}S BFM – The principle mode of testing used in the testbench is based on arbitrary transfers between the DUT and the I^{2}S BFM. All other types of checks on the DUT are invoked in parallel while the I^{2}S transfers are made in either direction.
- APB BFM – An APB BFM implements all the required interfacing to the two APB interfaces of DW_apb_i2s. This BFM ensures that all APB clocking, signal controls, address drives, and data drives and samplings are correctly implemented. The APB BFM serves to hide the drive and timing aspects of the APB bus from the Testbench Control block (TB_CTRL).

- Stimulus Generation – In order to separate the “how” from the “what” in the testbench, the stimulus generation (SG) process is coded within but separately from the main parts of the TB_CTRL block. The SG handles all of the generation of the required parameters, which are required to drive the DUT and the BFM. For example, the direction of transfer, the data patterns used for the serial transfer, and so on. To allow for simulation replays with a specific set of data which the SG uses, all the generated parameters are stored into an ASCII file for later retrieval. This also facilitates manual modifications to the simulation parameters if so desired. During generation, randomization can be applied to all or some of the parameters generated by the SG block.
- TB_CTRL – The Testbench Control block is responsible for the actual controls of the simulations executed on the testbench, Data fed from the SG is used to direct the TB_CTRL to perform the detailed steps/processes required to achieve the I_2S transfers. These include the appropriate programming of the DUT’s registers, writing of the transmit FIFOs, reading of the receive FIFOs, and so on.
- Scoreboard – To track the transfers made to and from the DUT, the Scoreboard keeps track of all the writes and reads to all the FIFOs in the DUT and the BFM by recording in separate entries. At the end of each simulation iteration, the Scoreboard is signalled by TB_CTRL to perform an internal check on the entries. The testbench simulation is either allowed to continue if the Scoreboard deems that all transfers have been correctly made (transmit and received) or otherwise terminated.

  In addition, any check failures detected during the I^2S transfers also force the simulation to terminate early. An internal debug mode is available to facilitate completion of the simulation while logging all transfer information and error messages.
- Checkers
  - Register integrity checks – ensure that the registers in the DUT are accessible through the APB interface.
  - Control checks – ensure that the DUT transmit and/or receive channel enables, as controlled through the APB interface, influence the corresponding transmit and/or receive abilities in the DUT.
  - Transfer checks – ensure that the DUT is capable of performing the transmit and/or receive operations correctly for the various modes and word sizes.
  - Status checks – ensure that the status bits in the DUT, accessed through the APB interface, correspond to the expected transmit and/or receive behaviors.
  - Clocking – ensure that the clock and transfer control signals are generated correctly as controlled through the APB interface.
  - Interrupt checks – ensure that the DUT’s interrupt outport is asserted and negated correctly.

8 Integration Considerations

After you have configured, tested, and synthesized your component with the coreTools flow, you can integrate the component into your own design environment. The following sections discuss general integration considerations for the slave interface of APB peripherals.

8.1 Reading and Writing from an APB Slave

When writing to and reading from DesignWare APB slaves, you should consider the following:

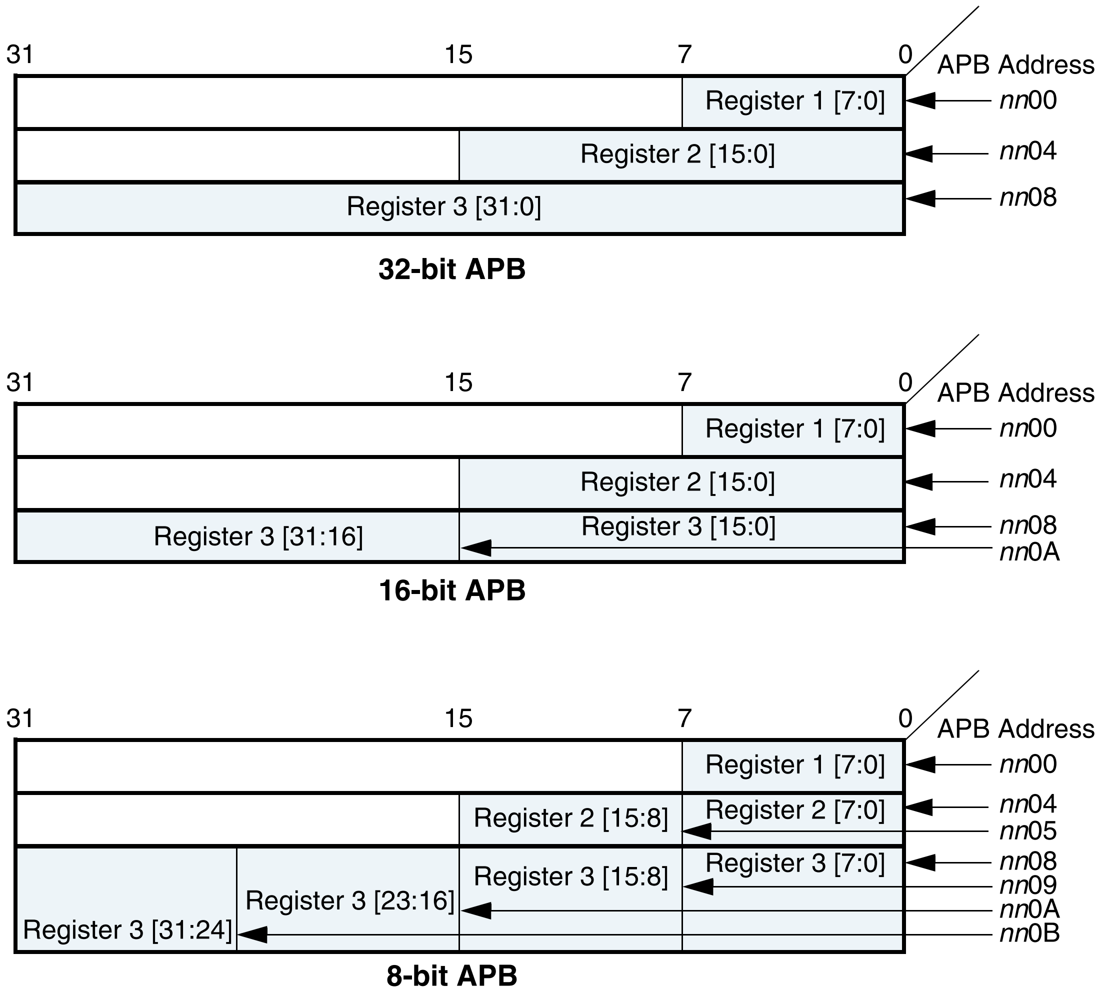
- The size of the APB peripheral should always be set equal to the size of the APB data bus, if possible.
- The APB bus has no concept of a transfer size or a byte lane, unlike the DW_ahb.
- The APB slave subsystem is little endian; the DW_apb performs the conversion from a big-endian AHB to the little-endian APB.
- All APB slave programming registers are aligned on 32-bit boundaries, irrespective of the APB bus size.
- The maximum APB_DATA_WIDTH is 32 bits. Registers larger than this occupies more than one location in the memory map.
- The DW_apb does not return any ERROR, SPLIT, or RETRY responses; it always returns an OKAY response to the AHB.
- For all bus widths:
    - In the case of a read transaction, registers less than the full bus width returns zeros in the unused upper bits.
    - Writing to bit locations larger than the register width does not have any effect. Only the pertinent bits are written to the register.
- The APB slaves do not need the full 32-bit address bus, paddr. The slaves include the lower bits even though they are not actually used in a 32- or 16-bit system.

8.1.1 Reading From Unused Locations

Reading from an unused location or unused bits in a particular register always returns zeros. Unlike an AHB slave interface, which would return an error, there is no error mechanism in an APB slave and, therefore, in the DW_apb.

The following sections show the relationship between the register map and the read/write operations for the three possible APB_DATA_WIDTH values: 8-, 16-, and 32-bit APB buses.

Figure 8-1 Read/Write Locations for Different APB Bus Data Widths



8.1.2 32-bit Bus System

For 32-bit bus systems, all programming registers can be read or written with one operation, as illustrated in the previous figure.

Because all registers are on 32-bit boundaries, paddr[1:0] is not actually needed in the 32-bit bus case. But these bits still exist in the configured code for usability purposes.

Note If you write to an address location not on a 32-bit boundary, the bottom bits are ignored/not used.

8.1.3 16-bit Bus System

For 16-bit bus systems, two scenarios exist, as illustrated in the previous picture:

1. The register to be written to or read from is less than or equal to 16 bits

   In this case, the register can be read or written with one transaction. In the case of a read transaction, registers less than 16 bits wide returns zeros in the un-used bits. Writing to bit locations larger than the register width causes nothing to happen, i.e. only the pertinent bits are written to the register.

2. The register to be written to or read from is >16 and <= 32 bits

   In this case, two AHB transactions are required, which in turn creates two APB transactions, to read or write the register. The first transaction should read/write the lower two bytes (half-word) and the second transaction the upper half-word.

Because the bus is reading a half-word at a time, paddr[0] is not actually needed in the 16-bit bus case. But these bits still exist in the configured code for connectivity purposes.

Note If you write to an address location not on a 16-bit boundary, the bottom bits are ignored/not used.

8.1.4 8-bit Bus System

For 8-bit bus systems, three scenarios exist, as illustrated in the previous picture:

1. The register to be written to or read from is less than or equal to 8 bits

   In this case, the register can be read or written with one transaction. In the case of a read transaction, registers less than 8 bits wide returns zeros in the unused bits. Writing to bit locations larger than the register width causes nothing to happen, that is, only the pertinent bits are written to the register.

2. The register to be written to or read from is >8 and <=16 bits

   In this case, two AHB transactions are required, which in turn creates two APB transactions, to read or write the register. The first transaction should read/write the lower byte and the second transaction the upper byte.

3. The register to be written to or read from is >16 and <=32 bits

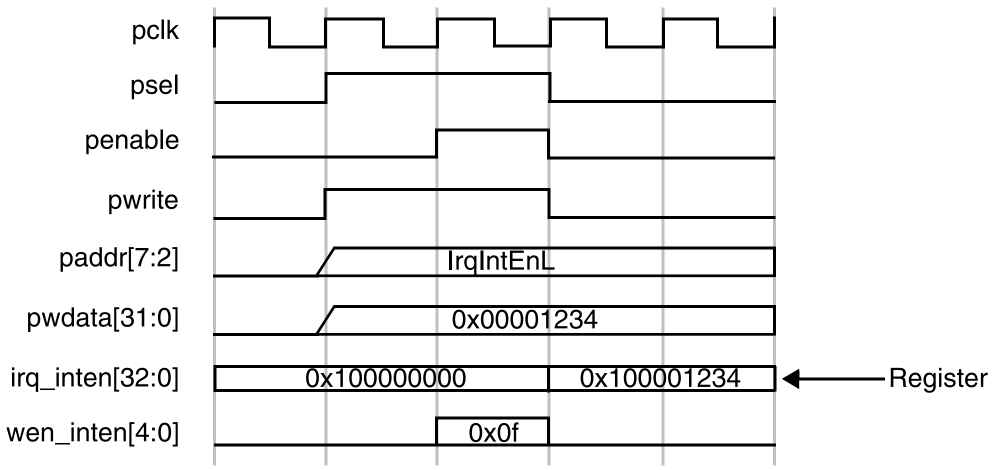
   In this case, four AHB transactions are required, which in turn creates four APB transactions, to read or write the register. The first transaction should read/write the lower byte and the second transaction the second byte, and so on.

Because the bus is reading a byte at a time, all lower bits of paddr are decoded in the 8-bit bus case.

8.2 Write Timing Operation

A timing diagram of an APB write transaction for an APB peripheral register (an earlier version of the DW_apb_ictl) is shown in the following figure. Data, address, and control signals are aligned. The APB frame lasts for two cycles when psel is high.

Figure 8-2 APB Write Transaction



A write can occur after the first phase with penable low, or after the second phase when penable is high. The second phase is preferred and is used in all APB slave components. The timing diagram is shown with the write occurring after the second phase. Whenever the address on paddr matches a corresponding address from the memory map and provided psel, pwrite, and penable are high, then the corresponding register write enable is generated.

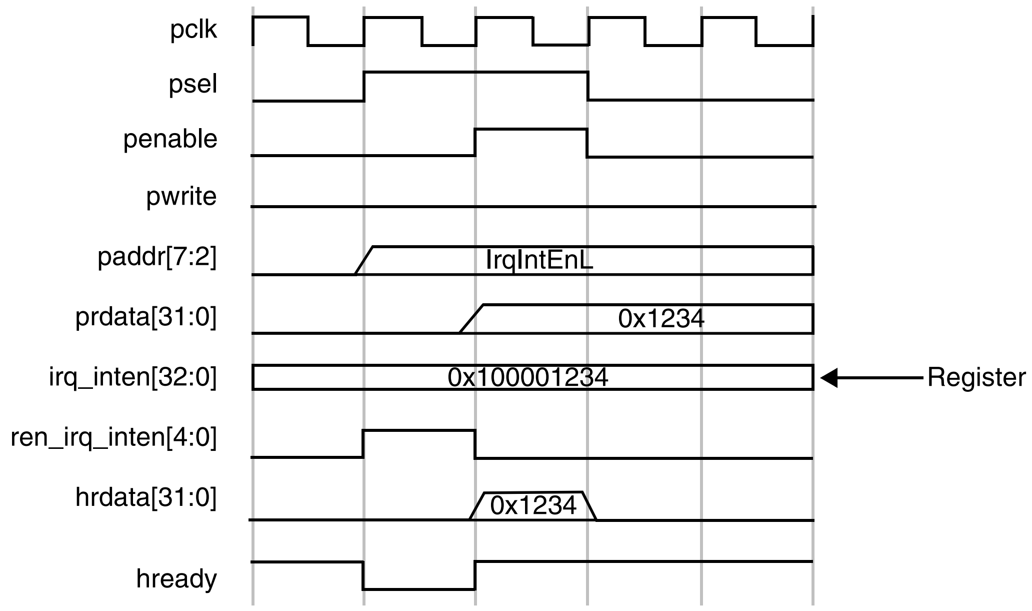
A write from the AHB to the APB does not require the AHB system bus to stall until the transfer on the APB has completed. A write to the APB can be followed by a read transaction from another AHB peripheral (not the DW_apb).

The timing example is a 33-bit register and a 32-bit APB data bus. To write this, 5 byte enables would be generated internally. The example shows writing to the first 32 bits with one write transaction.

8.3 Read Timing Operation

A timing diagram of an APB read transaction for an APB peripheral (an earlier version of the DW_apb_ictl) is shown in the following figure. The APB frame lasts for two cycles, when psel is high.

Figure 8-3 APB Read Transaction



Whenever the address on paddr matches the corresponding address from the memory map—psel is high, pwrite and penable are low—then the corresponding read enable is generated. The read data is registered within the peripheral before passing back to the master through the DW_apb and DW_ahb.

The qualification of the read-back data with hready from the bridge is shown in the timing diagram, but this does not form part of the APB interface. The read happens in the first APB cycle and is passed straight back to the AHB master in the same cycles as it passes through the bridge. By returning the data immediately to the AHB bus, the bridge can release control of the AHB data bus faster. This is important for systems where the APB clock is slower than the AHB clock.

Once a read transaction is started, it is completed and the AHB bus is held until the data is returned from the slave

Note If a read enable is not active, then the previously read data is maintained on the read-back data bus.

8.4 Accessing Top-level Constraints

To get SDC constraints out of coreConsultant, you need to first complete the synthesis activity and then use the “write_sdc” command to write out the results:

1. This cC command sets synthesis to write out scripts only, without running DC:

```
set_activity_parameter Synthesize ScriptsOnly 1
```

2. This cC command autocompletes the activity:

   ```
   autocomplete_activity Synthesize
   ```

3. Finally, this cC command writes out SDC constraints:

   ```
   write_sdc <filename>
   ```

8.5 Coherency

Coherency is where bits within a register are logically connected. For instance, part of a register is read at time 1 and another part is read at time 2. Being coherent means that the part read at time 2 is at the same value it was when the register was read at time 1. The unread part is stored into a shadow register and this is read at time 2. When there is no coherency, no shadow registers are involved.

A bus master may need to be able to read the contents of a register, regardless of the data bus width, and be guaranteed of the coherency of the value read. A bus master may need to be able to write a register coherently regardless of the data bus width and use that register only when it has been fully programmed. This may need to be the case regardless of the relationship between the clocks.

Coherency enables a value to be read that is an accurate reflection of the state of the counter, independent of the data bus width, the counter width, and even the relationship between the clocks. Additionally, a value written in one domain is transferred to another domain in a seamless and coherent fashion.

Throughout this appendix the following terms are used:

- **Writing**. A bus master programs a configuration register. An example is programming the load value of a counter into a register.
- **Transferring**. The programmed register is in a different clock domain to where it is used, therefore, it needs to be transferred to the other clock domain.
- **Loading**. Once the programmed register is transferred into the correct clock domain, it needs to be loaded or used to perform its function. For example, once the load value is transferred into the counter domain, it gets loaded into the counter.

8.5.1 Writing Coherently

Writing coherently means that all the bits of a register can be written at the same time. A peripheral may have programmable registers that are wider than the width of the connected APB data bus, which prevents all the bits being programmed at the same time unless additional coherency circuitry is provided.

The programmable register could be the load value for a counter that may exist in a different clock domain. Not only does the value to be programmed need to be coherent, it also needs to be transferred to a different clock domain and then loaded into the counter. Depending on the function of the programmable register, a qualifier may need to be generated with the data so that it knows when the new value is currently transferred and when it should be loaded into the counter.

Depending on the system and on the register being programmed, there may be no need for any special coherency circuitry. One example that requires coherency circuitry is a 32-bit timer within an 8-bit APB system. The value is entirely programmed only after four 8-bit wide write transfers. It is safe to transfer or use the register when the last byte is currently written. An example where no coherency is required is a 16-bit wide timer within a 16-bit APB system. The value is entirely programmed after a single 16-bit wide write transfer.

Coherency circuitry enables the value to be loaded into the counter only when fully programmed and crossed over clock domains if the peripheral clock is not synchronous to the processor clock. While the load register is being programmed, the counter has access to the previous load value in case it needs to reload the counter.

Coherency circuitry is only added in cores where it is needed. The coherency circuitry incorporates an upper byte method that requires users to program the load register in LSB to MSB order when the peripheral width is smaller than the register width. When the upper byte is programmed, the value can be transferred and loaded into the load register. When the lower bytes are being programmed, they need to be stored in shadow registers so that the previous load register is available to the counter if it needs to reload. When the upper byte is programmed, the contents of the shadow registers and the upper byte are loaded into the load register.

The upper byte is the top byte of a register. A register can be transferred and loaded into the counter only when it has been fully programmed. A new value is available to the counter once this upper byte is written into the register. The following table shows the relationship between the register width and the peripheral bus width for the generation of the correct upper byte. The numbers in the table represent bytes, Byte 0 is the LSB and Byte 3 is the MSB. NCR means that no coherency circuitry is required, as the entire register is written with one access.

Table 8-1 Upper Byte Generation

	Upper Byte Bus Width		
Load Register Width	8	16	32
1 - 8	NCR	NCR	NCR
9 - 16	1	NCR	NCR
17 - 24	2	2	NCR
25 - 32	3	2 (or 3)	NCR

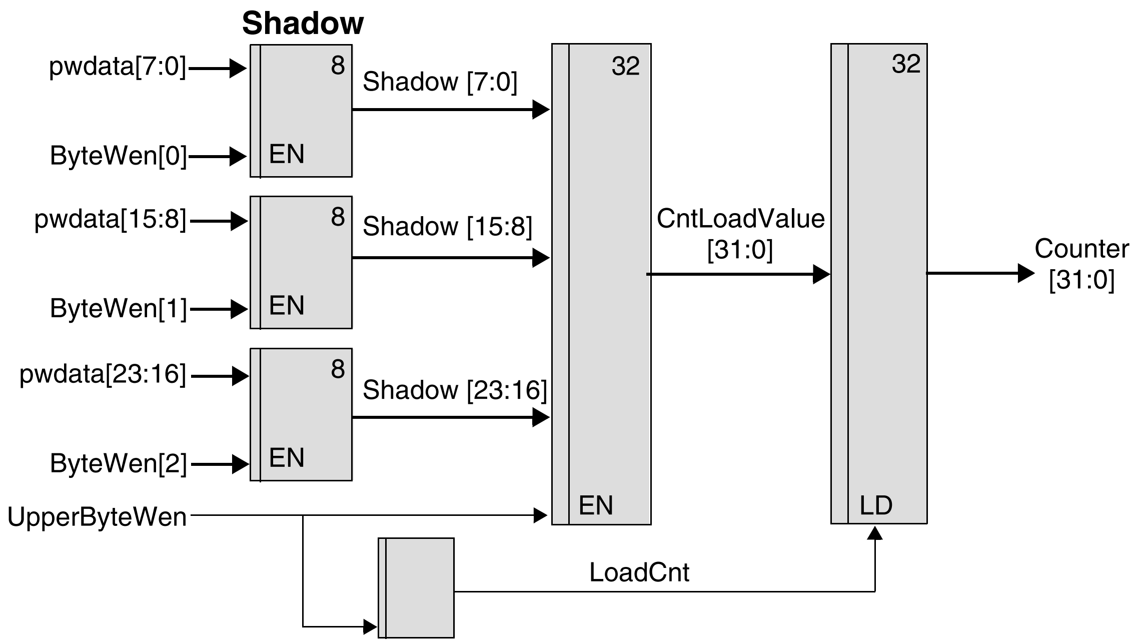
There are three relationship cases to be considered for the processor and peripheral clocks:

- Identical
- Synchronous (phase coherent but of an integer fraction)
- Asynchronous

8.5.1.1 Identical Clocks

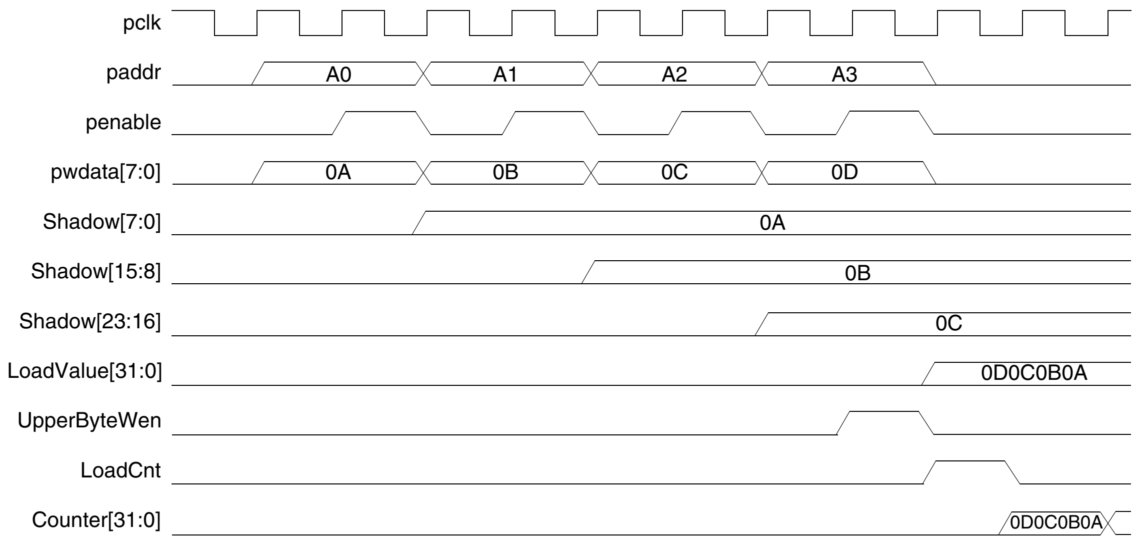
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

Figure 8-4 Coherent Loading – Identical Synchronous Clocks



The following figure shows a 32-bit register that is written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal lasts for one cycle and is used to load the counter with CntLoadValue.

Figure 8-5 Coherent Loading – Identical Synchronous Clocks



Each of the bytes that make up the load register are stored into shadow registers until the final byte is written. The shadow register is up to three bytes wide. The contents of the shadow registers and the final byte are transferred into the CntLoadValue register when the final byte is written. The counter uses this register to load/initialize itself. If the counter is operating in a periodic mode, it reloads from this register each time the count expires.

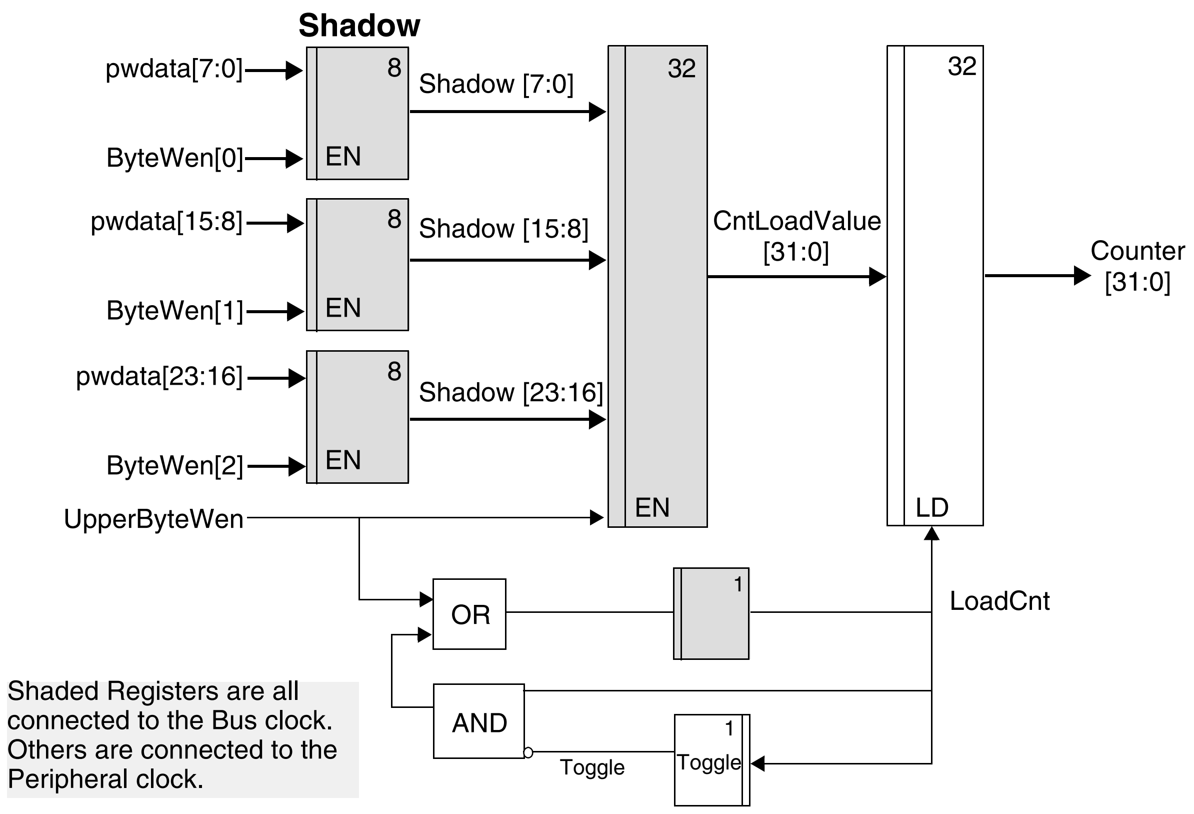
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this is signaled by LoadCnt = 1. After the upper byte is written, the LoadCnt goes to zero.

8.5.1.2 Synchronous Clocks

When the clocks are synchronous but do not have identical periods, the circuitry needs to be extended so that the LoadCnt signal is kept high until a rising edge of the counter clock occurs. This extension is necessary so that the value can be loaded, using LoadCnt, into the counter on the first counter clock edge. At the rising edge of the counter clock if LoadCnt is high, then a register clocked with the counter clock toggles, otherwise it keeps its current value. A circuit detecting the toggling is used to clear the original LoadCnt by looking for edge changes. The value is loaded into the counter when a toggle has been detected. Once it is loaded, the counter should be free to increment or decrement by normal rules.

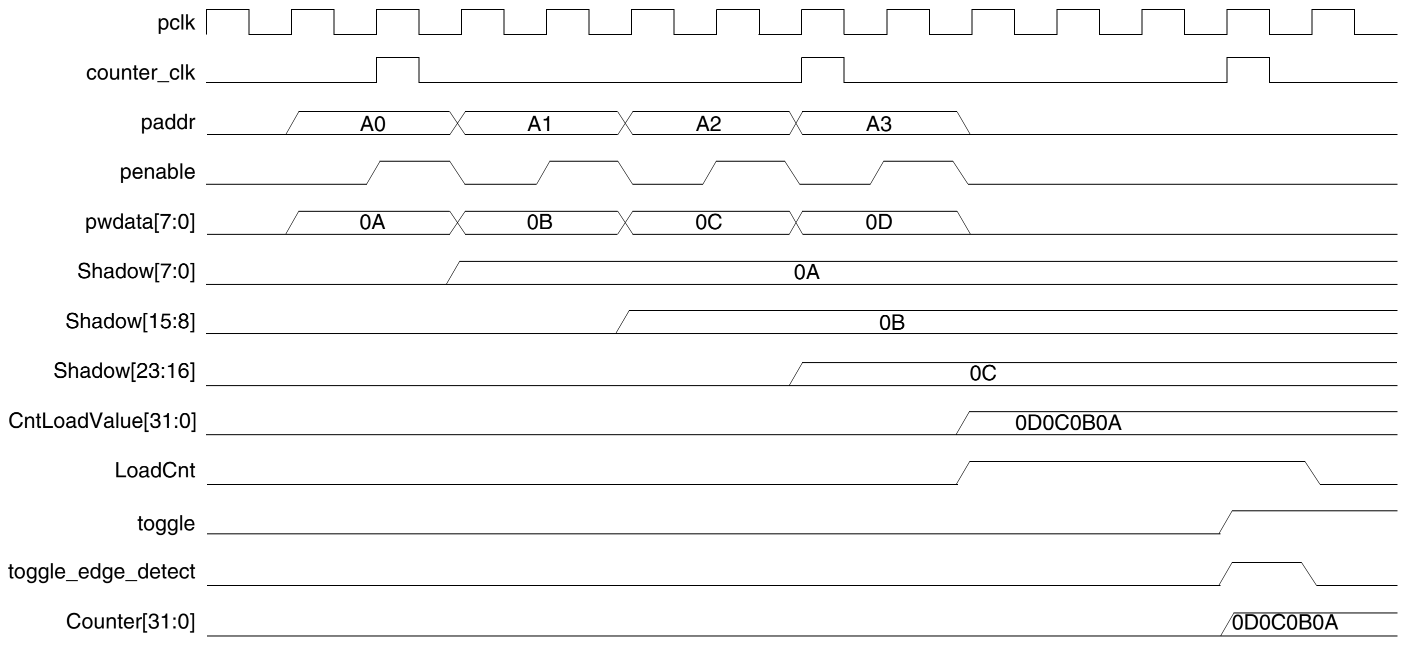
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

Figure 8-6 Coherent Loading – Synchronous Clocks



The following figure shows a 32-bit register being written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal is extended until a change in the toggle is detected and is used to load the counter.

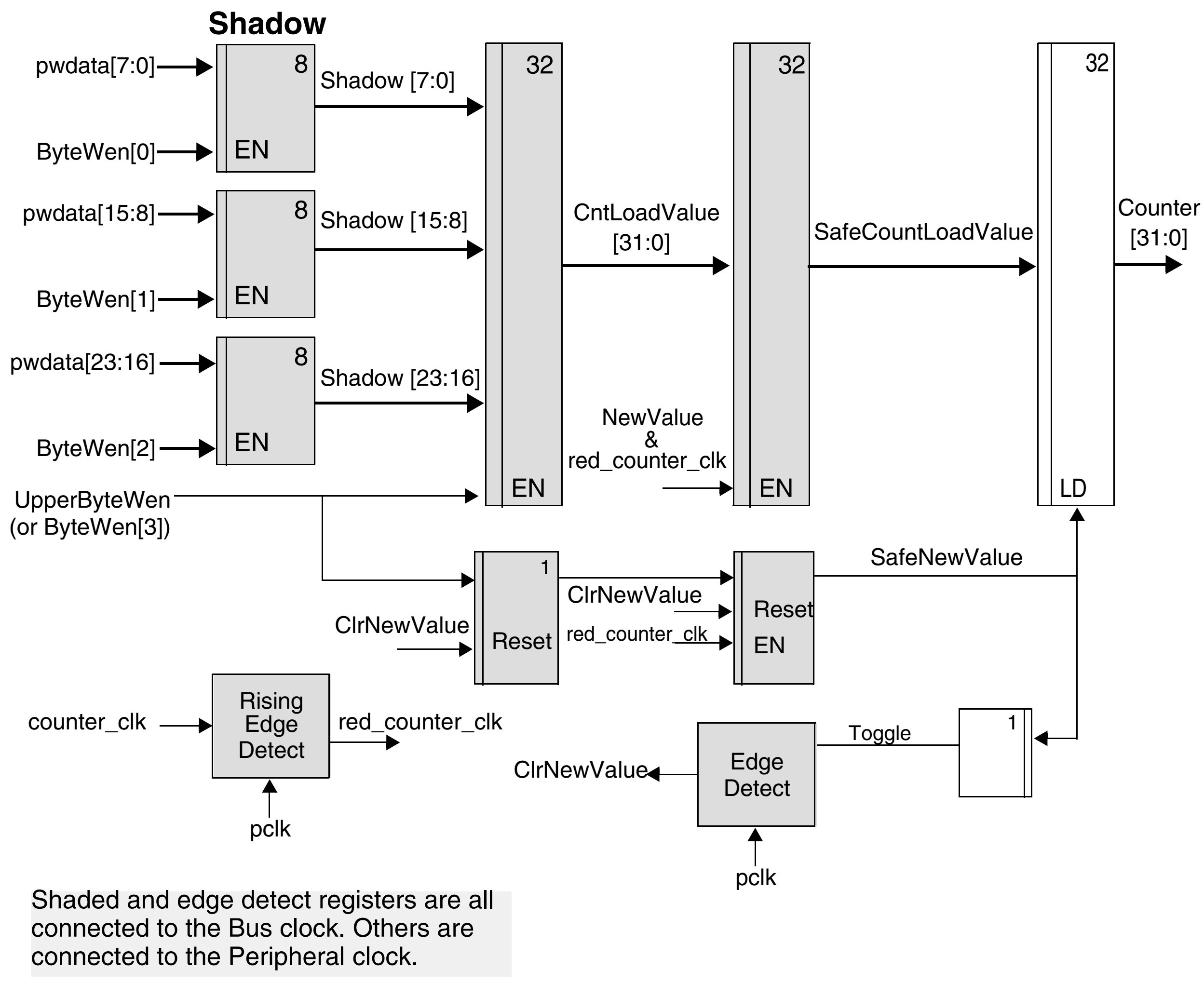
Figure 8-7 Coherent Loading – Synchronous Clocks



8.5.1.3 Asynchronous Clocks

When the clocks are asynchronous, the processor clock needs to be three-times the speed of the peripheral clock for the re-timing to operate correctly. The high pulse time of the peripheral clock needs to be greater than the period of the processor clock. The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are asynchronous.

Figure 8-8 Coherent Loading – Asynchronous Clocks

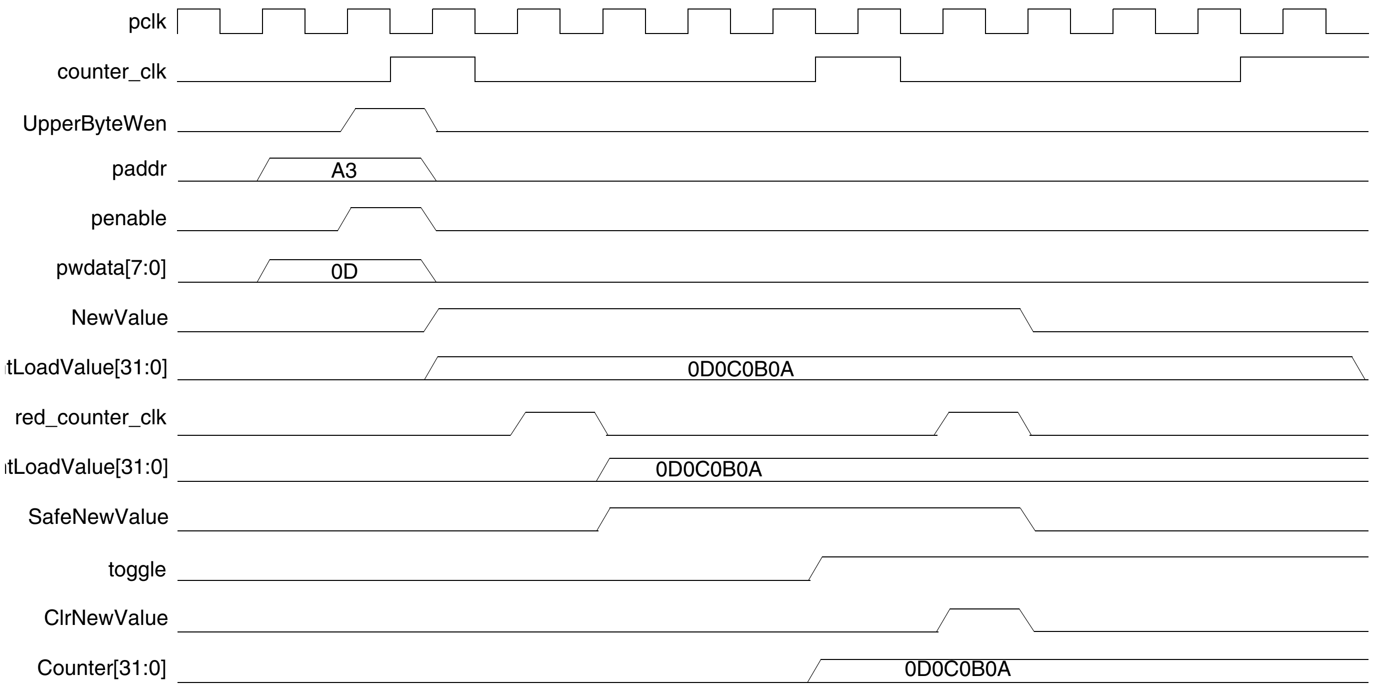


When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock to be used to re-time the peripheral clock. Upon a rising edge of the re-timed clock, the new value signal, NewValue, is transferred into a safe new value signal, SafeNewValue, which happens after the edge of the peripheral clock has occurred.

Every time there is a rising edge of the peripheral clock detected, the CntLoadValue is transferred into a SafeCntLoadValue. This value is used to transfer the load value across the clock domains. The SafeCntLoadValue only changes a number of bus clock cycles after the peripheral clock edge changes. A counter running on the peripheral clock is able to use this value safely. It could be up to two peripheral clock periods before the value is loaded into the counter. Along with this loaded value, there also is a single bit transferred that is used to qualify the loading of the value into the counter.

The timing diagram depicted in the following figure does not show the shadow registers being loaded. This is identical to the loading for the other clock modes.

Figure 8-9 Coherent Loading – Asynchronous Clocks



The NewValue signal is extended until a change in the toggle is detected and is used to update the safe value. The SafeNewValue is used to load the counter at the rising edge of the peripheral clock. Each time a new value is written the toggle bit is flipped and the edge detection of the toggle is used to remove both the NewValue and the SafeNewValue.

8.5.2 Reading Coherently

For writing to registers, an upper-byte concept is proposed for solving coherency issues. For read transactions, a lower-byte concept is required. The following table provides the relationship between the register width and the bus width for the generation of the correct lower byte.

Table 8-2 Lower Byte Generation

	Lower Byte Bus Width		
Counter Register Width	8	16	32
1 - 8	NCR	NCR	NCR
9 - 16	0	NCR	NCR

Table 8-2 **Lower Byte Generation**

	Lower Byte Bus Width		
17 - 24	0	0	NCR
25 - 32	0	0	NCR

Depending on the bus width and the register width, there may be no need to save the upper bits because the entire register is read in one access, in which case there is no problem with coherency. When the lower byte is read, the remaining upper bytes within the counter register are transferred into a holding register. The holding register is the source for the remaining upper bytes. Users must read LSB to MSB for this solution to operate correctly. NCR means that no coherency circuitry is required, as the entire register is read with one access.

There are two cases regarding the relationship between the processor and peripheral clocks to be considered as follows:

- Identical and/or synchronous
- Asynchronous

8.5.2.1 Synchronous Clocks

When the clocks are identical and/or synchronous, the remaining unread bits (if any) need to be saved into a holding register once a read is started. The first read byte must be the lower byte provided in the previous table, which causes the other bits to be moved into the holding register, SafeCntVal, provided that the register cannot be read in one access. The upper bytes of the register are read from the holding register rather than the actual register so that the value read is coherent. This is illustrated in the following figure and in the timing diagram after it.

Figure 8-10 Coherent Registering – Synchronous Clocks

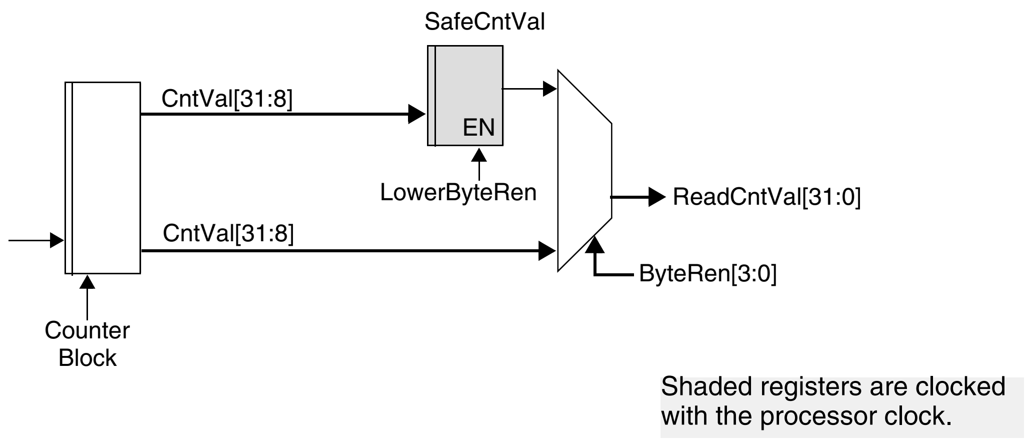
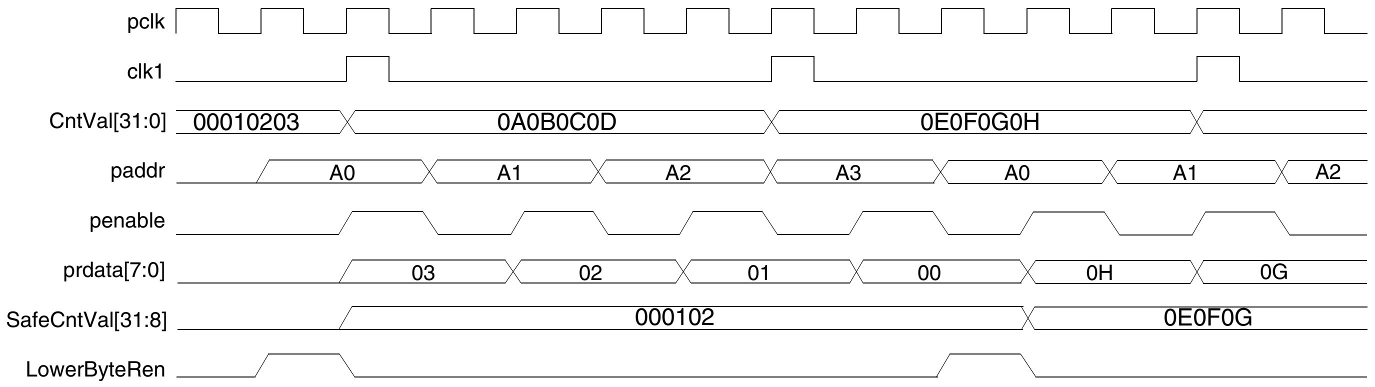


Figure 8-11 Coherent Registering – Synchronous Clocks



8.5.2.2 Asynchronous Clocks

When the clocks are asynchronous, the processor clock needs to be three times the speed of the peripheral clock for the re-timing to operate correctly. The high pulse time of the peripheral clock needs to be greater than the period of the processor clock.

To safely transfer a counter value from the counter clock domain to the bus clock domain, the counter clock signal should be transferred to the bus clock domain. When the rising edge detect of this re-timed counter clock signal is detected, it is safe to use the counter value to update a shadow register that holds the current value of the counter.

While reading the counter contents it may take multiple APB transfers to read the value.

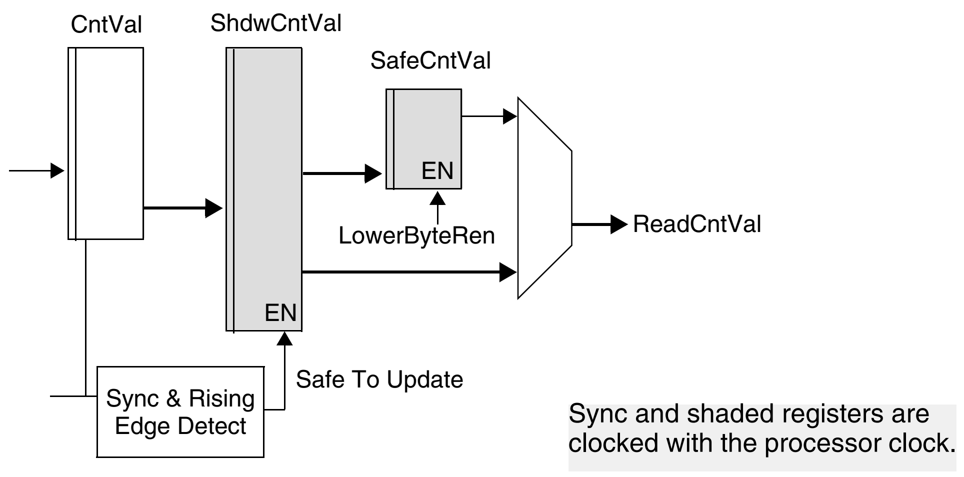
Note You must read LSB to MSB when the bus width is narrower than the counter width.

Once a read transaction has started, the value of the upper register bits need to be stored into a shadow register so that they can be read with subsequent read accesses. Storing these upper bits preserves the coherency of the value that is being read. When the processor reads the current value it actually reads the contents of the shadow register instead of the actual counter value. The holding register is read when the bus width is narrower than the counter width. When the LSB is read, the value comes from the shadow register; when the remaining bytes are read they come from the holding register. If the data bus width is wide enough to read the counter in one access, then the holding registers do not exist.

The counter clock is registered and successively pipelined to sense a rising edge on the counter clock. Having detected the rising edge, the value from the counter is known to be stable and can be transferred into the shadow register. The coherency of the counter value is maintained before it is transferred, because the value is stable.

The following figure illustrates the synchronization of the counter clock and the update of the shadow register.

Figure 8-12 Coherency and Shadow Registering – Asynchronous Clocks


CntVal
ShdwCntVal
SafeCntVal
EN
LowerByteRen
ReadCntVal
EN
Safe To Update
Sync & Rising Edge Detect
Sync and shaded registers are clocked with the processor clock.

8.6 Performance

This section discusses performance and the hardware configuration parameters that affect the performance of the DW_apb_i2s.

8.6.1 Power Consumption, Frequency, and Area Results

Table 8-3 provides information about the synthesis results (power consumption, frequency, and area) of the DW_apb_i2s using the industry standard 28nm technology library and how it affects performance.

Table 8-3 Power Consumption, Frequency, and Area Results for DW_apb_i2s Using 28nm Technology Library

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption
Default Configuration	pclk: 200 MHz sclk: 200 MHz sclk_n: 200 MHz	6739 gates	0.0759uW	77.8uW

Master Minimum Configuration:
APB_DATA_WIDTH=8
I2S_MODE_EN=1
I2S_RECEIVER_BLOCK=1
I2S_TRANSMITTER_BLOCK=0
I2S_FIFO_DEPTH_GLOBAL=4
	pclk: 200 MHz
	sclk: 200 MHz
	sclk_n: 200 MHz
		2608 gates	0.0462uW	48.0uW

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption

Slave Minimum Configuration:
APB_DATA_WIDTH=8
I2S_MODE_EN=0
I2S_RECEIVER_BLOCK=1
I2S_TRANSMITTER_BLOCK=0
I2S_FIFO_DEPTH_GLOBAL=4
	pclk: 200 MHz
	sclk: 200 MHz
	sclk_n: 200 MHz
		2446 gates	0.0368uW	34.8uW

Master Maximum Combined Configuration:
APB_DATA_WIDTH=32
I2S_MODE_EN=1
I2S_RECEIVER_BLOCK=1
I2S_TRANSMITTER_BLOCK=1
I2S_FIFO_DEPTH_GLOBAL=16
I2S_HAS_DMA_INTERFACE=1
I2S_DMA_HS_TYPE=1
I2S_DMA_POL=1
I2S_RX_DMA=1
I2S_TX_DMA=1
	pclk: 200 MHz
	sclk: 200 MHz
	sclk_n: 200 MHz
		12143 gates	0.114uW	135uW

Slave Maximum Combined Configuration:
APB_DATA_WIDTH=32
I2S_MODE_EN=0
I2S_RECEIVER_BLOCK=1
I2S_TRANSMITTER_BLOCK=1
I2S_FIFO_DEPTH_GLOBAL=16
I2S_HAS_DMA_INTERFACE=1
I2S_DMA_HS_TYPE=1
I2S_DMA_POL=1
I2S_RX_DMA=1
I2S_TX_DMA=1
	pclk: 200 MHz
	sclk: 200 MHz
	sclk_n: 200 MHz
		11788 gates	0.106uW	105uW

Master Maximum Configuration:
APB_DATA_WIDTH=32
I2S_MODE_EN=1
I2S_RECEIVER_BLOCK=1
I2S_TRANSMITTER_BLOCK=1
I2S_FIFO_DEPTH_GLOBAL=16
I2S_HAS_DMA_INTERFACE=1
I2S_DMA_HS_TYPE=0
I2S_DMA_POL=1
I2S_RX_DMA=1
I2S_TX_DMA=1
	pclk: 200 MHz
	sclk: 200 MHz
	sclk_n: 200 MHz
		12036 gates	0.114uW	123uW

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption

Slave Maximum Configuration:
APB_DATA_WIDTH=32
I2S_MODE_EN=0
I2S_RECEIVER_BLOCK=1
I2S_TRANSMITTER_BLOCK=1
I2S_FIFO_DEPTH_GLOBAL=16
I2S_HAS_DMA_INTERFACE=1
I2S_DMA_HS_TYPE=0
I2S_DMA_POL=1
I2S_RX_DMA=1
I2S_TX_DMA=1
	pclk: 200 MHz
	sclk: 200 MHz
	sclk_n: 200 MHz
		11954 gates	0.0368uW	34.8uW

A
Synchronizer Methods

This appendix describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_apb_i2s to cross clock boundaries.

This appendix contains the following sections:

- "Synchronizers Used in DW_apb_i2s" on page 178
- "Synchronizer 1: Simple Double Register Synchronizer (DW_apb_i2s)" on page 179

Note The DesignWare Building Blocks (DWBB) contains several synchronizer components with functionality similar to methods documented in this appendix. For more information about the DWBB synchronizer components go to:
https://www.synopsys.com/dw/buildingblock.php

A.1 Synchronizers Used in DW_apb_i2s

Each of the synchronizers and synchronizer sub-modules are comprised of verified DesignWare Basic Core (BCM) RTL designs. The BCM synchronizer designs are identified by the synchronizer type. The corresponding RTL files comprising the BCM synchronizers used in the DW_apb_i2s are listed and cross referenced to the synchronizer type in Table A-1. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

Table A-1 Synchronizers Used in DW_apb_i2s

Synchronizer Module File	Synchronizer Type and Number
DW_apb_i2s_bcm21.v	Synchronizer 1: Simple Multiple register synchronizer

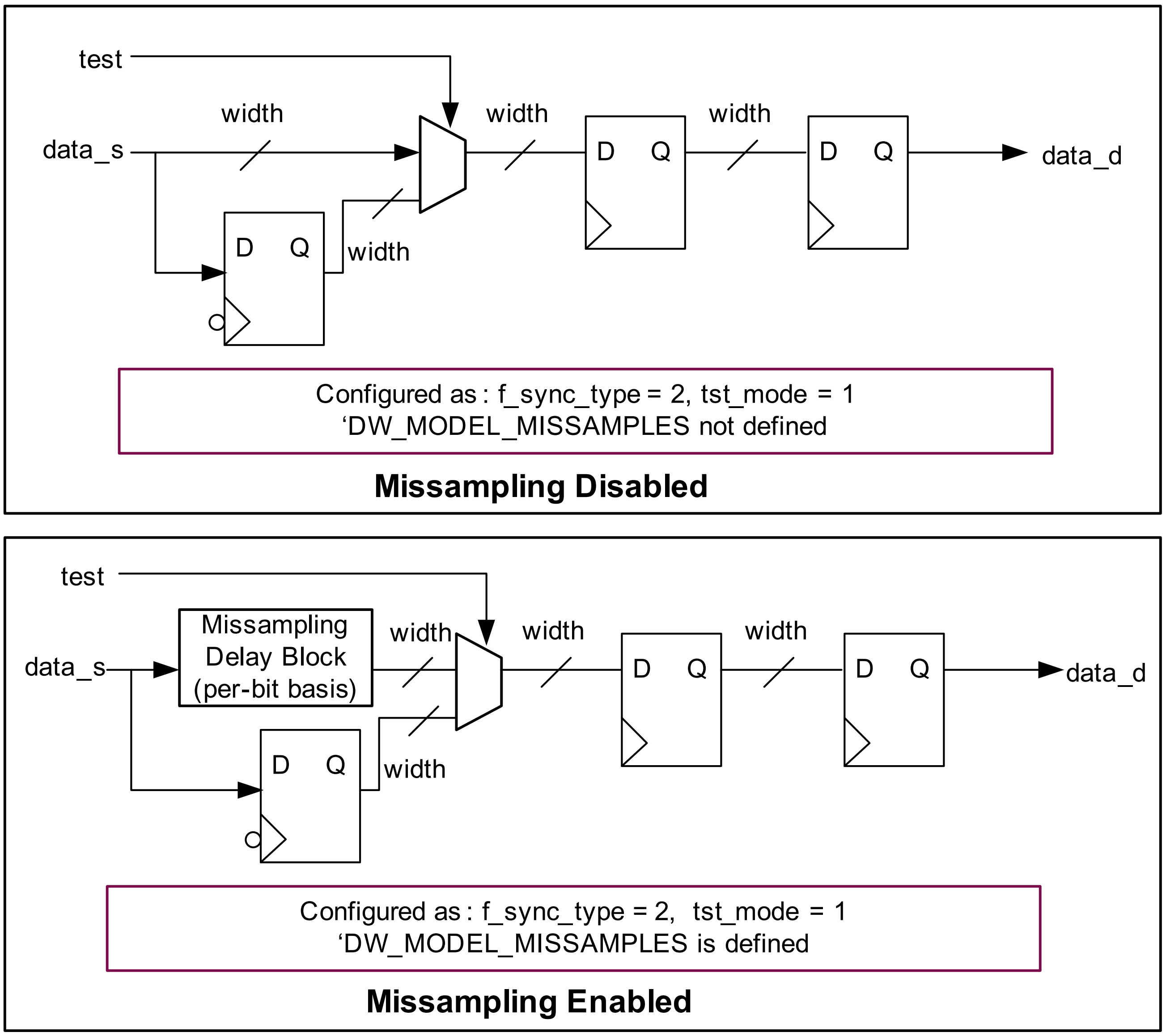
Note The BCM21 is a basic multiple register based synchronizer module used in the design. It can be replaced with equivalent technology specific synchronizer cell.

Caution Be cautious while choosing the depth (I2S_SYNC_DEPTH) for different synchronization mechanisms as 1 (where for the first stage negative-edge flip-flop is used and for the second stage positive-edge flip-flop is used). At higher frequencies, as the maximum time available for meta-stability resolution is halved with respect to the available clock period, this can lead to meta-stability - when synchronizer depth is selected as 1.
It is recommended to use the synchronizer depths that are greater than or equal to 2. The depth of 1 will be deprecated in the future releases.

A.2 Synchronizer 1: Simple Double Register Synchronizer (DW_apb_i2s)

This is a single clock data bus synchronizer for synchronizing control signals that crosses asynchronous clock boundaries. The synchronization scheme uses two stage synchronization process (Figure A-1) both using positive edge of clock.

Figure A-1 Block Diagram of Synchronizer 1 with Two-Stage Synchronization (Both Positive Edges)


test
width
width
width
data_s
D Q
D Q
data_d
D Q
width
Configured as : f_sync_type = 2, tst_mode = 1
'DW_MODEL_MISSAMPLES not defined
Missampling Disabled
test
Missampling Delay Block (per-bit basis)
width
width
width
data_s
D Q
D Q
data_d
D Q
width
Configured as : f_sync_type = 2, tst_mode = 1
'DW_MODEL_MISSAMPLES is defined
Missampling Enabled

B
Internal Parameter Descriptions

Provides a description of the internal parameters that might be indirectly referenced in expressions in the Signals, Parameters, or Registers chapters. These parameters are not visible in the coreConsultant GUI and most of them are derived automatically from visible parameters. **You must not set any of these parameters directly.**

Some expressions might refer to TCL functions or procedures (sometimes identified as **function_of**) that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Table B-1 Internal Parameters

Parameter Name	Equals To
ENCODED_APB_DATA_WIDTH	{[function_of: APB_DATA_WIDTH]}
ENCODED_I2S_FIFO_DEPTH_GLOBAL	{[function_of: I2S_FIFO_DEPTH_GLOBAL]}
ENCODED_I2S_RX_CHANNELS	{[function_of: I2S_RX_CHANNELS]}
ENCODED_I2S_RX_WORDSIZE_0	{[function_of: I2S_RX_WORDSIZE_0]}
ENCODED_I2S_RX_WORDSIZE_1	{[function_of: I2S_RX_WORDSIZE_1]}
ENCODED_I2S_RX_WORDSIZE_2	{[function_of: I2S_RX_WORDSIZE_2]}
ENCODED_I2S_RX_WORDSIZE_3	{[function_of: I2S_RX_WORDSIZE_3]}
ENCODED_I2S_TX_CHANNELS	{[function_of: I2S_TX_CHANNELS]}
ENCODED_I2S_TX_WORDSIZE_0	{[function_of: I2S_TX_WORDSIZE_0]}
ENCODED_I2S_TX_WORDSIZE_1	{[function_of: I2S_TX_WORDSIZE_1]}
ENCODED_I2S_TX_WORDSIZE_2	{[function_of: I2S_TX_WORDSIZE_2]}
ENCODED_I2S_TX_WORDSIZE_3	{[function_of: I2S_TX_WORDSIZE_3]}
I2S_ADDR_SLICE_LHS	10

Table B-1 Internal Parameters (Continued)

Parameter Name	Equals To
I2S_COMP_VERSION	32'h3131312a

C Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
activity	A set of functions in coreConsultant that step you through configuration, verification, and synthesis of a selected core.
AHB	Advanced High-performance Bus — high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces (Arm® Limited specification).
AMBA	Advanced Microcontroller Bus Architecture — a trademarked name by Arm® Limited that defines an on-chip communication standard for high speed microcontrollers.
APB	Advanced Peripheral Bus — optimized for minimal power consumption and reduced interface complexity to support peripheral functions (Arm® Limited specification).
APB bridge	DW_apb submodule that converts protocol between the AHB bus and APB bus.
application design	Overall chip-level design into which a subsystem or subsystems are integrated.
arbiter	AMBA bus submodule that arbitrates bus activity between masters and slaves.
BFM	Bus-Functional Model — A simulation model used for early hardware debug. A BFM simulates the bus cycles of a device and models device pins, as well as certain on-chip functions. See also Full-Functional Model.
big-endian	Data format in which most significant byte comes first; normal order of bytes in a word.
blocked command stream	A command stream that is blocked due to a blocking command issued to that stream; see also command stream, blocking command, and non-blocking command.
blocking command	A command that prevents a testbench from advancing to next testbench statement until this command executes in model. Blocking commands typically return data to the testbench from the model.

bus bridge	Logic that handles the interface and transactions between two bus standards, such as AHB and APB. See APB bridge.
command channel	Manages command streams. Models with multiple command channels execute command streams independently of each other to provide full-duplex mode function.
command stream	The communication channel between the testbench and the model.
component	A generic term that can refer to any synthesizable IP or verification IP in the DesignWare Library. In the context of synthesizable IP, this is a configurable block that can be instantiated as a single entity (VHDL) or module (Verilog) in a design.
configuration	The act of specifying parameters for a core prior to synthesis; can also be used in the context of VIP.
configuration intent	Range of values allowed for each parameter associated with a reusable core.
core	Any configurable block of synthesizable IP that can be instantiated as a single entity (VHDL) or module (Verilog) in a design. Core is the preferred term for a big piece of IIP. Anything that requires coreConsultant for configuration, as well as anything in the DesignWare Cores library, is a core.
core developer	Person or company who creates or packages a reusable core. All the cores in the DesignWare Library are developed by Synopsys.
core integrator	Person who uses coreConsultant or coreAssembler to incorporate reusable cores into a system-level design.
coreAssembler	Synopsys product that enables automatic connection of a group of cores into a subsystem. Generates RTL and gate-level views of the entire subsystem.
coreConsultant	A Synopsys product that lets you configure a core and generate the design views and synthesis views you need to integrate the core into your design. Can also synthesize the core and run the unit-level testbench supplied with the core.
coreKit	An unconfigured core and associated files, including the core itself, a specified synthesis methodology, interfaces definitions, and optional items such as verification environment files and core-specific documentation.
cycle command	A command that executes and causes HDL simulation time to advance.
decoder	Software or hardware subsystem that translates from and “encoded” format back to standard format.
design context	Aspects of a component or subsystem target environment that affect the synthesis of the component or subsystem.
design creation	The process of capturing a design as parameterized RTL.
Design View	A simulation model for a core generated by coreConsultant.
DesignWare Synthesizable Components	The Synopsys name for the collection of AMBA-compliant coreKits and verification models delivered with DesignWare and used with coreConsultant or coreAssembler to quickly build DesignWare Synthesizable Component designs.

DesignWare cores	A specific collection of synthesizable cores that are licensed individually. For more information, refer to www.synopsys.com/designware.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the DesignWare Synthesizable Components.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multi-byte word; see also little-endian and big-endian.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
GPIO	General Purpose Input Output.
GTECH	A generic technology view used for RTL simulation of encrypted source code by non-Synopsys simulators.
hard IP	Non-synthesizable implementation IP.
HDL	Hardware Description Language – examples include Verilog and VHDL.
IIP	Implementation Intellectual Property — A generic term for synthesizable HDL and non-synthesizable “hard” IP in all of its forms (coreKit, component, core, MacroCell, and so on).
implementation view	The RTL for a core. You can simulate, synthesize, and implement this view of a core in a real chip.
instantiate	The act of placing a core or model into a design.
interface	Set of ports and parameters that defines a connection point to a component.
IP	Intellectual property — A term that encompasses simulation models and synthesizable blocks of HDL code.
little-endian	Data format in which the least-significant byte comes first.
MacroCell	Bigger IP blocks (6811, 8051, memory controller) available in the DesignWare Library and delivered with coreConsultant.
master	Device or model that initiates and controls another device or peripheral.
model	A Verification IP component or a Design View of a core.
monitor	A device or model that gathers performance statistics of a system.
non-blocking command	A testbench command that advances to the next testbench statement without waiting for the command to complete.
peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.

RTL	Register Transfer Level. A higher level of abstraction that implies a certain gate-level structure. Synthesis of RTL code yields a gate-level design.
SDRAM	Synchronous Dynamic Random Access Memory; high-speed DRAM adds a separate clock signal to control signals.
SDRAM controller	A memory controller with specific connections for SDRAMs.
slave	Device or model that is controlled by and responds to a master.
SoC	System on a chip.
soft IP	Any implementation IP that is configurable. Generally referred to as synthesizable IP.
static controller	Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.
subsystem	In relation to coreAssembler, highest level of RTL that is automatically generated.
synthesis intent	Attributes that a core developer applies to a top-level design, ports, and core.
synthesizable IP	A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.
technology-independent	Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.
Testsuite Regression Environment (TRE)	A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
workspace	A network location that contains a personal copy of a component or subsystem. After you configure the component or subsystem (using coreConsultant or coreAssembler), the workspace contains the configured component/subsystem and generated views needed for integration of the component/subsystem at the top level.
wrap, wrapper	Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.
zero-cycle command	A command that executes without HDL simulation time advancing.

Index