SYNOPSYS®

DesignWare® DW_apb_ssi

Databook

DW_apb_ssi – *Product Code*

4.03a
December 2020

Copyright Notice and Proprietary Information

Contents

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 3.11b onward.

Version	Date	Description
4.03a	December 2020	**Added:** ■ "Endian Conversion Support" on page 91 ■ "BCM Library Components" on page 211 ■ "Synchronization Depth" on page 90 **Updated:** ■ Version changed for 2020.12a release ■ "Master SPI and SSP Serial Transfers" on page 38 ■ "Clocks and Resets" on page 90 ■ "Performance" on page 208 ■ "Parameter Descriptions", "Signal Descriptions", "Register Descriptions", "Internal Parameter Descriptions" are auto extracted with change bars from the RTL **Renamed:** ■ Synchronizer Methods to Appendix A, "Basic Core Module (BCM) Library" ■ Reset Signals to "Clocks and Resets" on page 90 **Removed:** ■ Index chapter

Version	Date	Description
4.02a	July 2018	**Added:**
		■ Added support for configurable synchronization depth through the following parameters: SSI_P2S_SYNC_DEPTH, and SSI_S2P_SYNC_DEPTH
		Updated:
		■ Version changed for 2018.07a release
		■ "Performance" on page 208
		■ "Parameter Descriptions", "Signal Descriptions", "Register Descriptions", "Internal Parameter Descriptions" are auto extracted with change bars from the RTL
		■ Updated Figure 2-50 and Figure 2-51
		■ The minimum frequency of ssi_clk is corrected from six times to eight times the maximum expected frequency of the bit-rate clock.
		Added:
		■ Added ssi_busy signal in debug interface
		Removed:
		■ Chapter 2, "Building and Verifying a Component or Subsystem" and added the contents in the newly created user guide.

Version	Date	Description
4.01a	October 2016	■ Version changed to 2016.10a ■ Updated "Features" on page 20 ■ Added following parameters to "Parameter Descriptions" on page 93 - SSI_IO_MAP_EN - SSI_HAS_DDR - SSI_HAS_RXDS ■ Updated fields for "CTRLR0" and "SPI_CTRLR0" registers in "Register Descriptions" on page 123. ■ Added register "TXD_DRIVE_EDGE" in "Register Descriptions" on page 123 ■ Added the "Advanced I/O Mapping for Enhanced SPI Modes" on page 71 section ■ Added the "Dual Data-Rate (DDR) Support in SPI Operation" on page 72 section ■ Added "Read Data Strobe Signal Support" on page 75 ■ Modified "Write Operation in Enhanced SPI Modes" on page 65 and "Read Operation in Enhanced SPI Modes" on page 68 ■ Added "XIP Mode Support in SPI Mode" on page 75 ■ Added "APB 3.0 Support" on page 89 ■ Added "APB 4.0 Support" on page 89 ■ Added "Data Mask Support for SPI" on page 77 ■ Added following parameters to "Parameter Descriptions" on page 93 for XIP and APB 4.0 support: - SSI_APBIF_TYPE - SSI_APB3_ERR_RESP_EN - SSI_XIP_EN ■ Added following signals in "Signal Descriptions" on page 105 for XIP support: - xip_en - pready - pslverr

Version	Date	Description
		■ Added following signals in "Signal Descriptions" on page 105 to support the data mask feature: - pstrb - prpot ■ Removed "Running Leda on Generated Code with coreConsultant", and reference to Leda directory in Table 2-1 ■ Removed "Running Leda on Generated Code with coreAssembler" section, and reference to Leda directory in Table 2-4 ■ Added an entry for the xprop directory in Table 2-1 and Table 2-4. ■ Added "Running VCS XPROP Analyzer" ■ Chapter 3, "Parameter Descriptions" and Chapter 5, "Register Descriptions" auto-extracted from the RTL ■ Moved "Internal Parameter Descriptions" to Appendix ■ Updated Appendix A.2.3, "Synchronizer 2: Synchronous (Dual-clock) FIFO Controller With Static Flags" ■ Moved Table 2-1 and Table 2-2 from Registers chapter to "Transmit and Receive FIFO Buffers" on page 29 ■ Moved Table 2-4 and Table 2-5 from Registers chapter to "DMA Controller Interface" on page 77
4.00a	June 2015	■ Modified the Clock Ratios section and added the following sub-sections: ❑ "SSI_ENH_CLK_RATIO = 0" on page 26 ❑ "SSI_ENH_CLK_RATIO = 1" on page 26 ❑ "Frequency Ratio Summary" on page 28 ■ Added the ""Enhanced SPI Modes" on page 65" section in the "Functional Description" chapter ■ Added "Interfacing DW_apb_ssi with Dual/Quad Capable Devices" on page 217 to support switching between standard and Dual/Quad modes of operation ■ Added the "Running SpyGlass® Lint and SpyGlass® CDC" ■ Added the "Running SpyGlass on Generated Code with coreAssembler" ■ Added Chapter C, "Internal Parameter Descriptions" ■ Added Appendix A, "Basic Core Module (BCM) Library" ■ Modified description for "Interrupt polarity", and added description for "Serial clock polarity" and "Serial clock phase", under "Features" on page 20 ■ Chapter 4, "Signal Descriptions" auto-extracted from the RTL ■ Updated area and power numbers in sections "Area" and "Power Consumption"

Version	Date	Description
3.23a	June 2014	■ Enhancement to support 32-bit frame size: - Added: - "Frame Size Support" section - SSI_MAX_XFER_SIZE parameter - Updated: - "Data Register APB Access" section - CTRLR0 and DR register ■ Added "Performance" section in "Integration Considerations" chapter ■ Updated Registers chapter: - Fixed typo in DR row of Memory Map Table 6-1 - Added more clarity on the Register reserved width - Corrected the offset in the RX_SAMPLE_DLY register - DCOL field description updated ■ Corrected External Input/Output Delay in Signald chapters
3.22b	May 2013	■ Added a section to describe Reset Signals ■ Updated the template.
3.22a	September 2012	Added the product code on the cover and in Table 1-1
3.22a	March 2012	■ Clarified conditions for asserting and clearing dma_tx_single and dma_rx_single signals ■ Added notes in Verification chapter clarifying that SSI master and slave BFMs are not VMT VIP models
3.21b	November 2011	Version change for 2011.11a release
3.21a	October 2011	Version change for 2011.10a release
3.20a	June 2011	Modified ssi_sleep signal description
3.20a	June 2011	■ Updated system diagram in Figure 1-1 ■ Enhanced "Related Documents" section in Preface
3.19a	April 2011	Added material for new SSI_SCPH0_SSTOGGLE parameter
3.18a	January 2011	Corrected descriptions of TXFTLR and RXFTLR registers
3.18a	December 2010	Version change for 2010.12a release
3.17a	November 2010	Corrected DW_ahb_dmac response in "Receive Watermark Level and Receive FIFO Overflow" section
3.17a	October 2010	Changes to clarify RTL enhancement for logic added to master state machine to prevent txd output from toggling when only receiving data frame
3.16a	September 2010	Corrected names of include files and vcs command used for simulation
3.15a	March 2010	Added a programmable delay register used to sample incoming data

Version	Date	Description
3.13a	December 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks
3.12a	July 2009	Corrected equations for avoiding underflow when programming a source burst transaction
3.12a	May 2009	Removed references to QuickStarts, as they are no longer supported
3.12a	April 2009	Enhanced overview and Figure 5-B
3.12a	October 2008	■ Changed DR address offset from 0x60-0x15c to 0x60-0xfc ■ Version change for 2008.10a release
3.11c	June 2008	Version change for 2008.06a release
3.11b	March 2008	DR register offset changed to 0x60 - 0x15c; occupies sixteen 32-bit addresses
3.11b	January 2008	■ Updated to revised installation guide and consolidated release notes ■ Changed references of "Designware AMBA" to simply "DesignWare"
3.11b	June 2007	Version change for 2007.06a release

Preface

This databook provides information that you need to interface the DesignWare® Synchronous Serial Interface (SSI), referred to as DW_apb_ssi throughout the remainder of this databook. This component conforms to the *AMBA Specification, Revision 2.0* from Arm®.

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

DatabookOrganization

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_ssi.

- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_apb_ssi.

- Chapter 4, "Signal Descriptions" provides a list and description of the DW_apb_ssi signals.

- Chapter 5, "Register Descriptions" describes the programmable registers of the DW_apb_ssi.

- Chapter 6, "Programming the DW_apb_ssi" provides information needed to program the configured DW_apb_ssi.

- Chapter 7, "Verification" provides information on verifying the configured DW_apb_ssi.

- Chapter 8, "Integration Considerations" includes information you need to integrate the configured DW_apb_ssi into your design.

- Appendix A "Basic Core Module (BCM) Library" documents the synchronizer methods (blocks of synchronizer functionality), and list of BCM library components used in DW_apb_ssi.

- Appendix B, "Application Notes" provides getting started information that allows you to walk through the process of using the DW_apb_ssi with Synopsys coreConsultant tool.

- Appendix C, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals chapter.

- Appendix D, "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 (Documentation Overview).*

Web Resources

- DesignWare IP product information: https://www.synopsys.com/designware-ip.html

- Your custom DesignWare IP page: https://www.synopsys.com/dw/mydesignware.php

- Documentation through SolvNetPlus: https://solvnetplus.synopsys.com (Synopsys password required)

- Synopsys Common Licensing (SCL): https://www.synopsys.com/keys

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

- Prepare the following debug information, if applicable:

  ❏ For environment set-up problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, select the following menu:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This option gathers all the Synopsys product data needed to begin debugging an issue and writes it to the *<core tool startup directory>*/debug.tar.gz file.

  ❏ 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.

- *For the fastest response*, enter a case through SolvNetPlus:

  a. https://solvnetplus.synopsys.com

    > **☞ Note**   SolvNetPlus does not support Internet Explorer. Use a supported browser such as Microsoft Edge, Google Chrome, Mozilla Firefox, or Apple Safari.

  b. Click the **Cases** menu and then click **Create a New Case** (below the list of cases).
  c. Complete the mandatory fields that are marked with an asterisk and click **Save**.

Ensure to include the following:

- **Product L1:** DesignWare Library IP
- **Product L2:** AMBA

    d.   After creating the case, attach any debug files you created.

For more information about general usage information, refer to the following article in SolvNetPlus:

https://solvnetplus.synopsys.com/s/article/SolvNetPlus-Usage-Help-Resources

- 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 L1 and Product L2 names, and Version number in your e-mail 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.

- 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

The DW_apb_ssi is a programmable Synchronous Serial Interface (SSI) peripheral. This 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. To access the product page and documentation for AMBA components, see the DesignWare IP Solutions for AMBA Interconnect page. (SolvNetPlus ID required)

Figure 1-1   Example of DW_apb_ssi in a Complete System
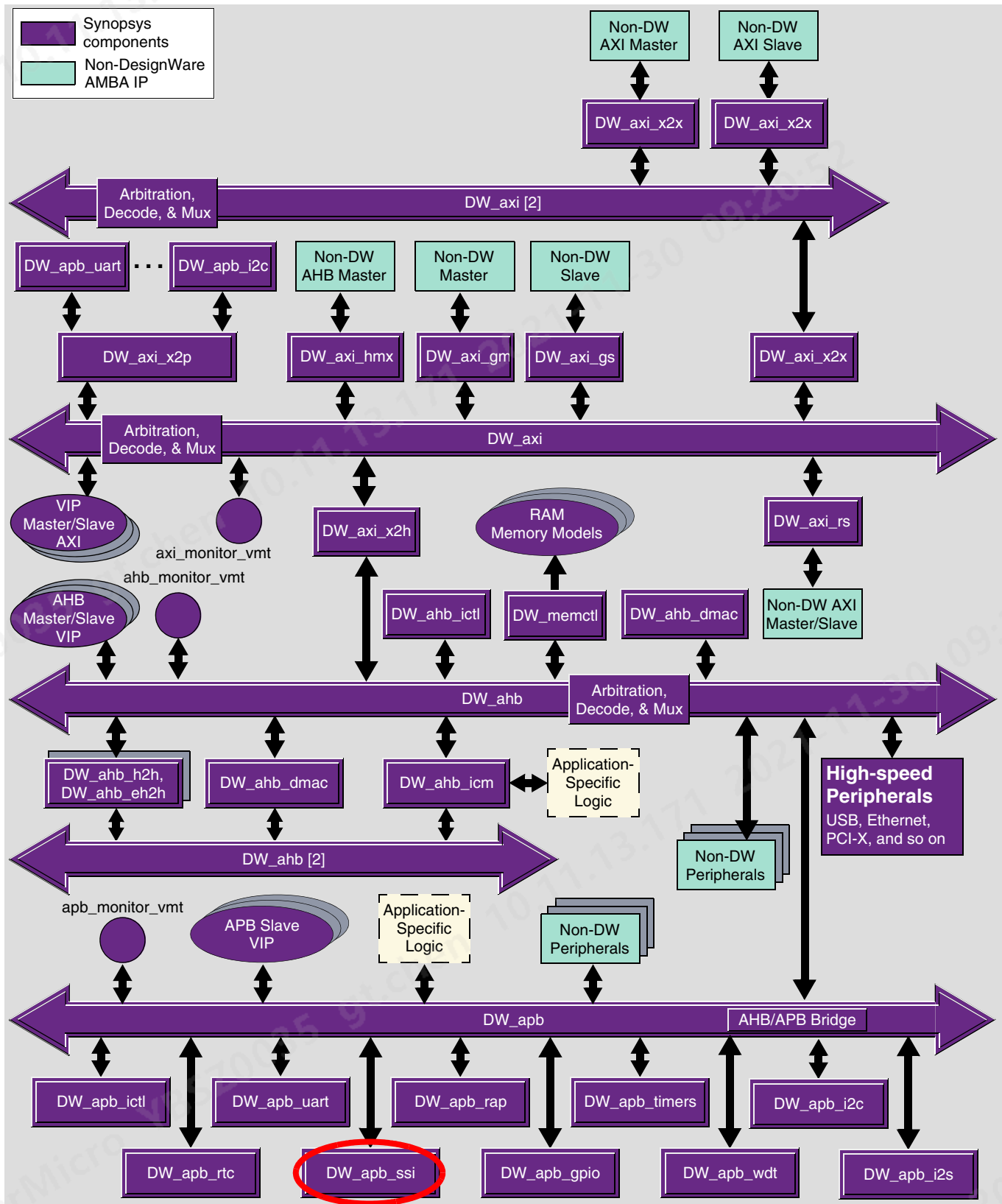
You can connect, configure, synthesize, and verify the DW_apb_ssi within a DesignWare subsystem using coreAssembler, documentation for which is available on the web in the *coreAssembler User Guide*.

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

1.2 General Product Description

The Synopsys DW_apb_ssi is a component of the DesignWare Advanced Peripheral Bus (DW_apb) and conforms to the *AMBA Specification, Revision 2.0* from Arm®.

1.2.1 DW_apb_ssi Block Diagram

Figure 1-2 shows the following functional groupings of the main interfaces to the DW_apb_ssi block:

- APB interface and DMA Controller Interface
- Transmit and receive FIFO controllers and an FSM controller
- Register block
- Shift control and interrupt logic

Figure 1-2   DW_apb_ssi Block Diagram
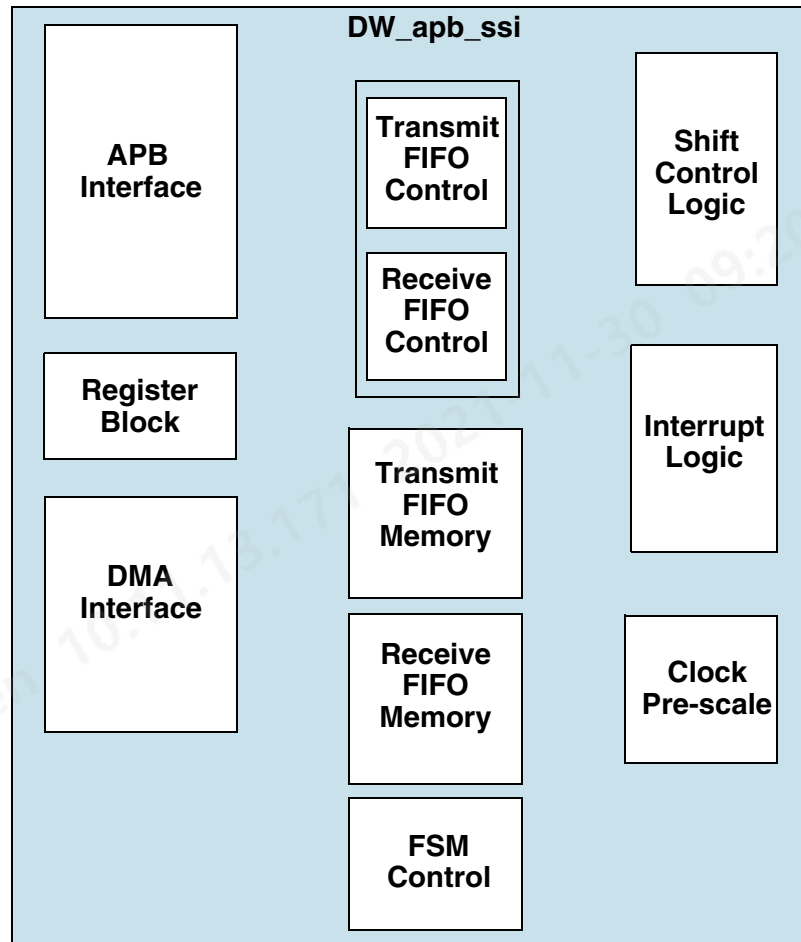
1.3   Features

DW_apb_ssi has the following features:

- APB interface – Allows for easy integration into a DesignWare Synthesizable Components for AMBA 2 implementation.

- APB3 and APB4 protocol support.

- Scalable APB data bus width – Supports APB data bus widths of 8, 16, and 32 bits.

- Serial-master or serial-slave operation – Enables serial communication with serial-master or serial-slave peripheral devices.

- Configurable and programmable Dual/Quad/Octal SPI support in Master Mode – Configure DW_apb_ssi to support Dual/Quad/Octal SPI mode, and then program DW_apb_ssi for dual/quad/octal SPI transfers.

- Dual Data Rate (DDR) and Read Data Strobe (RDS) Support - Enables the DW_apb_ssi master to perform operations with the device in DDR and RDS modes when working in Dual/Quad/Octal mode of operation.

- Data Mask Support - Enables the DW_apb_ssi to selectively update the bytes in the device. This feature is applicable only in enhanced SPI modes.

- eXecute-In-Place (XIP) support - Enables the DW_apb_ssi master to behave as a memory mapped I/O and fetches the data from the device based on the APB read request. This feature is applicable only in enhanced SPI modes.

- DMA Controller Interface – Enables the DW_apb_ssi to interface to a DMA controller over the bus using a handshaking interface for transfer requests.

- Independent masking of interrupts – Master collision, transmit FIFO overflow, transmit FIFO empty, receive FIFO full, receive FIFO underflow, and receive FIFO overflow interrupts can all be masked independently.

- Multi-master contention detection – Informs the processor of multiple serial-master accesses on the serial bus.

- Bypass of meta-stability flip-flops for synchronous clocks – When the APB clock (pclk) and the DW_apb_ssi serial clock (ssi_clk) are synchronous, meta-stable flip-flops are not used when transferring control signals across these clock domains.

- Programmable delay on the sample time of the received serial data bit (rxd), when configured in Master Mode; enables programmable control of routing delays resulting in higher serial data-bit rates.

- Programmable features:

  - Serial interface operation – Choice of Motorola SPI, Texas Instruments Synchronous Serial Protocol or National Semiconductor Microwire.

  - Clock bit-rate – Dynamic control of the serial bit rate of the data transfer; used in only serial-master mode of operation.

  - Data Item size (4 to 32 bits) – Item size of each data transfer under the control of the programmer.

- Configurable features:

  - FIFO depth – Configurable depth of the transmit and receive FIFO buffers from 2 to 256 words deep. The FIFO width is fixed at 16/32 bits, depending upon the SSI_MAX_XFER_SIZE parameter.

  - Number of slave select outputs – When operating as a serial master, 1 to 16 serial slave-select output signals can be generated.

  - Hardware/software slave-select – Dedicated hardware slave-select lines can be used or software control can be used to target the serial-slave device.

  - Combined or individual interrupt lines – You may choose to bring all individual interrupt lines or one combined interrupt line from the DW_apb_ssi to the interrupt controller.

  - Interrupt polarity – This configuration option selects the active level of the output interrupt lines.

  - Serial clock polarity – This configuration option selects the serial-clock polarity of the SPI format directly after reset.

  - Serial clock phase – This configuration option selects the serial-clock phase of the SPI format directly after reset.

- Endian conversion feature for Data register and XIP read transfers

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

1.4   Standards Compliance

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

1.5   Verification Environment Overview

The DW_apb_ssi 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 189 section discusses the specific procedures for verifying the DW_apb_ssi.

1.6   Licenses

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

1.7   Where To Go From Here

At this point, you may want to get started working with the DW_apb_ssi 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_ssi component, see "Overview of the coreConsultant Configuration and Integration Process" in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

For more information about implementing your DW_apb_ssi component within a DesignWare subsystem using coreAssembler, see "Overview of the coreAssembler Configuration and Integration Process" *DesignWare Synthesizable Components for AMBA 2 User Guide*.

2

Functional Description

The DW_apb_ssi is a configurable, synthesizable, and programmable component that is a full-duplex master or slave-synchronous serial interface. The host processor accesses data, control, and status information on the DW_apb_ssi through the APB interface. The DW_apb_ssi may also interface with a DMA Controller using an optional set of DMA signals, which can be selected at configuration time.

As described in more detail later, the DW_apb_ssi can be configured in one of two modes of operations: as a serial master or a serial slave. The DW_apb_ssi can connect to any serial-master or serial-slave peripheral device using one of the following interfaces:

- Motorola Serial Peripheral Interface (SPI)
- Texas Instruments Serial Protocol (SSP)
- National Semiconductor Microwire

2.1   DW_apb_ssi Overview

In order for the DW_apb_ssi to connect to a serial-master or serial-slave peripheral device, the peripheral must have a least one of the following interfaces:

- Motorola Serial Peripheral Interface (SPI) – A four-wire, full-duplex serial protocol from Motorola. There are four possible combinations for the serial clock phase and polarity. The clock phase (SCPH) determines whether the serial transfer begins with the falling edge of the slave select signal or the first edge of the serial clock. The slave select line is held high when the DW_apb_ssi is idle or disabled. For more information, see "Motorola Serial Peripheral Interface (SPI)" on page 47.

- Texas Instruments Serial Protocol (SSP) – A four-wire, full-duplex serial protocol. The slave select line used for SPI and Microwire protocols doubles as the frame indicator for the SSP protocol. For more information, see "Texas Instruments Synchronous Serial Protocol (SSP)" on page 53.

- National Semiconductor Microwire – A half-duplex serial protocol, which uses a control word transmitted from the serial master to the target serial slave. For more information, see "National Semiconductor Microwire" on page 54.

You can program the FRF (frame format) bit field in the Control Register 0 (CTRLR0) to select which protocol is used. You specify the FRF at configuration time to be hardcoded or programmable by setting the SSI_HC_FRF parameter. For more information about this configuration parameter, see "Parameter Descriptions" on page 93.

The serial protocols supported by the DW_apb_ssi allow for serial slaves to be selected or addressed using either hardware or software. When implemented in hardware, serial slaves are selected under the control of dedicated hardware select lines. The number of select lines generated from the serial master is equal to the number of serial slaves present on the bus. The serial-master device asserts the select line of the target serial slave before data transfer begins. This architecture is illustrated in Figure 2-1(A).

When implemented in software, the input select line for all serial slave devices should originate from a single slave select output on the serial master. In this mode it is assumed that the serial master has only a single slave select output. If there are multiple serial masters in the system, the slave select output from all masters can be logically ANDed to generate a single slave select input for all serial slave devices.

The main program in the software domain controls selection of the target slave device; this architecture is illustrated in Figure 2-1(B). Software would use the SSIENR register in all slaves in order to control which slave is to respond to the serial transfer request from the master device.

The following example is pseudo code that illustrates how to use software to select the target slave.

2.1.1 Example of Target Slave Selection Using Software

```
int main() {

// This function sets the SSI_EN bit to logic '0' in the SSIENR register
// of each device on the serial bus

disable_all_serial_devices();

// This function initializes the master device for the serial transfer
// 1. Write CTRLR0 to match the required transfer
// 2. If transfer is receive only write number of frames into CTRLR1
// 3. Write BAUDR to set the transfer baud rate.
// 4. Write TXFTLR and RXFTLR to set FIFO threshold levels
// 5. Write IMR register to set interrupt masks
// 6. Write SER register bit[0] to logic '1'
// 7. Write SSIENR register bit[0] to logic '1' to enable the master.

initialize_mst(ssi_mst_1);

// This function initializes the target slave device (slave 1 in this example)
// for the serial transfer.
// 1. Write CTRLR0 to match the required transfer
// 2. Write TXFTLR and RXFTLR to set FIFO threshold levels
// 3. Write IMR register to set interrupt masks
// 4. Write SSIENR register bit[0] to logic '1' to enable the slave.
// 5. If the slave is to transmit data, write data into TX FIFO
// Now the slave is enabled and awaiting an active level on its
// ss_in_n input port.  Note all other serial slaves are disabled (SSI_EN=0)
// and therefore does not respond to an active level on their ss_in_n port.

initialize_slv(ssi_slv_1);

// This function begins the serial transfer by writing transmit data into
// the master's TX FIFO.
```

```
start_serial_xfer(ssi_mst_1);

// You can poll the busy status with a function or use an ISR to determine
// when the serial transfer has completed.
}
```

The DW_apb_ssi does not enforce hardware or software control for serial-slave device selection. You can configure the DW_apb_ssi for either implementation, illustrated in Figure 2-1.

Figure 2-1   Hardware/Software Slave Selection

ss = slave select line
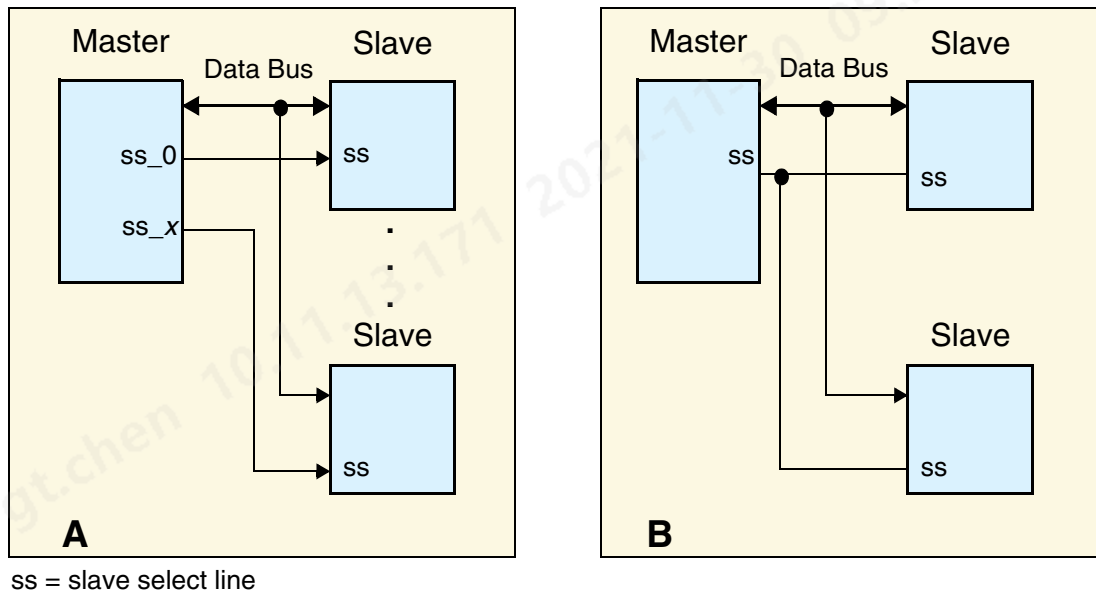
2.2   Clock Ratios

When DW_apb_ssi is configured as a master device, the maximum frequency of the bit-rate clock (sclk_out) is one-half the frequency of ssi_clk. This allows the shift control logic to capture data on one clock edge of sclk_out and propagate data on the opposite edge.

Figure 2-2 on page 25 illustrates the maximum ratio between sclk_out and ssi_clk.

Figure 2-2   Maximum sclk_out/ssi_clk Ratio
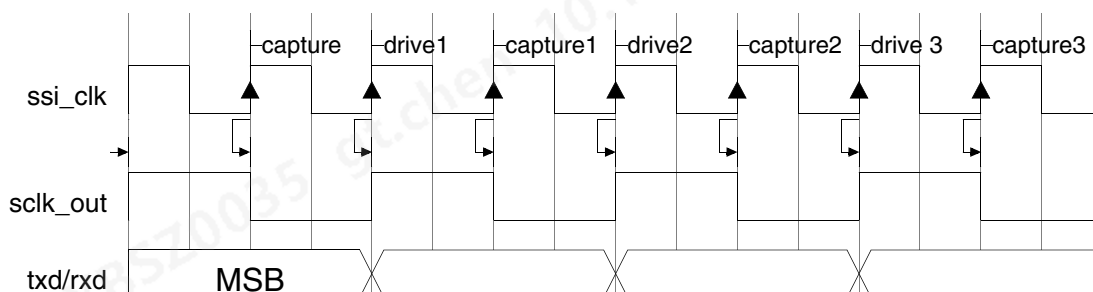
The sclk_out line toggles only when an active transfer is in progress. At all other times it is held in an inactive state, as defined by the serial protocol under which it operates.

The frequency of sclk_out can be derived from the following equation:

$$F_{sclk\_out} = \frac{F_{ssi\_clk}}{SCKDV}$$

SCKDV is a bit field in the programmable register BAUDR, holding any even value in the range 0 to 65,534. If SCKDV is 0, then sclk_out is disabled.

When DW_apb_ssi is configured as a slave device, the minimum frequency of ssi_clk depends on the SSI_ENH_CLK_RATIO configuration parameter and the operation of the slave peripheral.

2.2.1   SSI_ENH_CLK_RATIO = 0

If the slave device is *receive only,* the minimum frequency of ssi_clk is eight times the maximum expected frequency of the bit-rate clock from the master device (sclk_in). The sclk_in signal is double synchronized to the ssi_clk domain, and then it is edge detected.

If the slave device is *transmit and receive,* the minimum frequency of ssi_clk is 12 times the maximum expected frequency of the bit-rate clock from the master device (sclk_in). This minimum frequency is to ensure that data on the master's rxd line is stable before the master's shift control logic captures the data. The 12:1 ratio ensures that the slave has driven data onto the master's rxd line three ssi_clk cycles before the data is captured, which is indicated by tc (time before capture) in Figure 2-3.

Figure 2-3   Slave ssi_clk/sclk_in Ratio
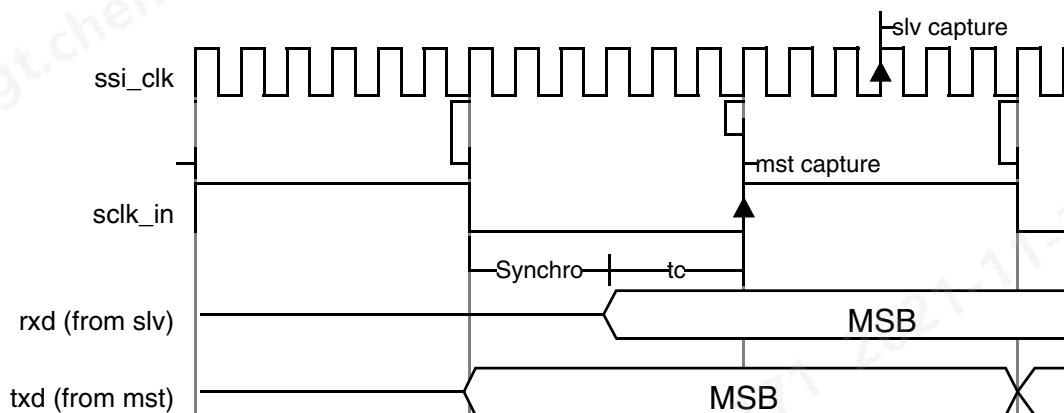
2.2.2   SSI_ENH_CLK_RATIO = 1

In this mode, the transmit and receive shift registers work directly with the bit-rate clock from the master device (sclk_in) to eliminate the need for synchronization.

To reduce the synchronization delay, the synchronization scheme uses two flip flops: one works on the positive edge of ssi_clk; and other works on the negative edge of ssi_clk. These flip flops reduce the synchronization delay to one ssi_clk cycle and enable DW_apb_ssi to work on lower clock ratios. When SSI_ENH_CLK_RATIO=1, the DW_apb_ssi slave device minimum frequency of ssi_clk is 4 times the maximum expected frequency of the bit-rate clock (sclk_in).

The bit-rate clock is gated according to different capture and driving edges for different frame formats. The capture clock (clk_cap) is used in the receive shifter to capture the data on the rxd line. The driving clock (clk_driv) is used in the transmit shifter to drive the data on the txd line. Figure 2-4, Figure 2-5, Figure 2-6

and Figure 2-7 illustrate how capture and driving clocks are derived from bit-rate clock for different frame formats.

Figure 2-4   Frame Format for Motorola Serial Peripheral Interface (SPI) with SCPH = 0
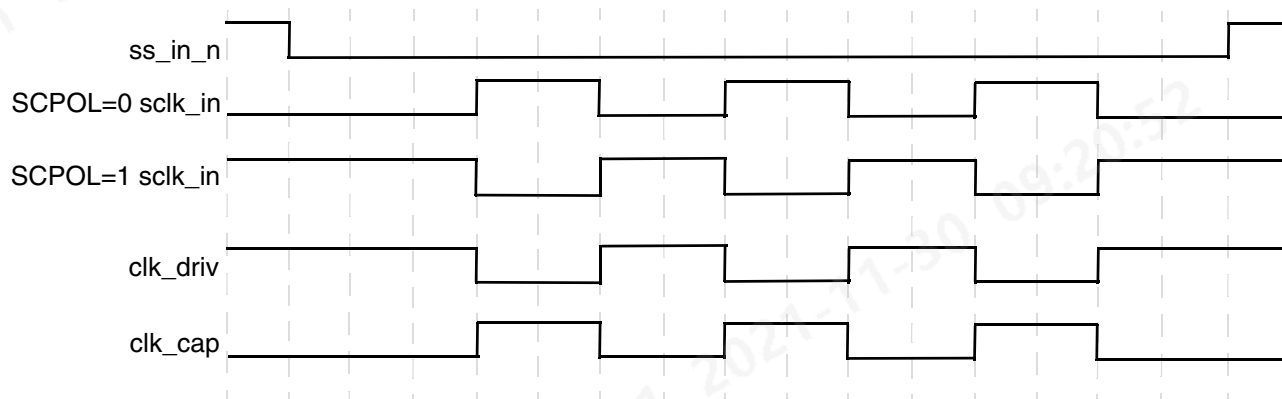
Figure 2-5   Frame Format for Motorola Serial Peripheral Interface (SPI) with SCPH = 1
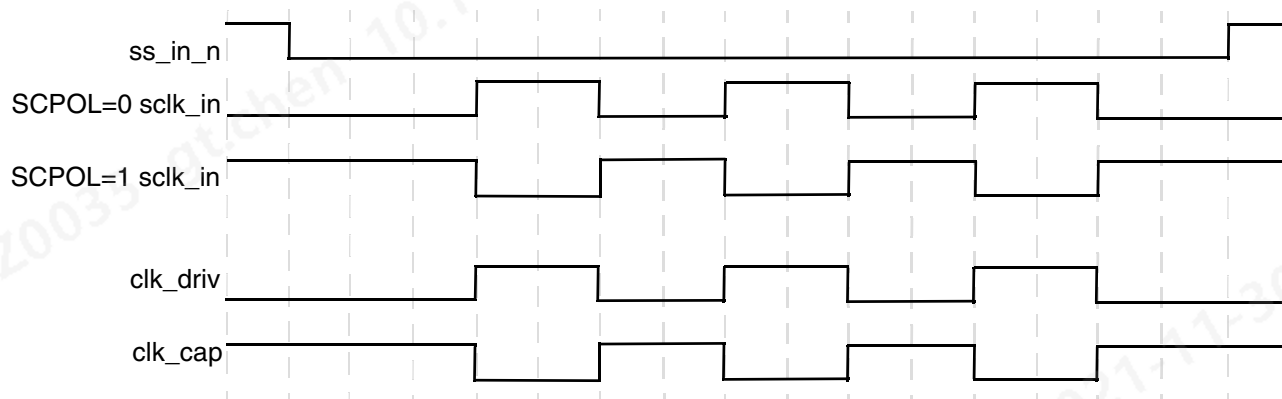
Figure 2-6   National Semiconductors Microwire Frame Format
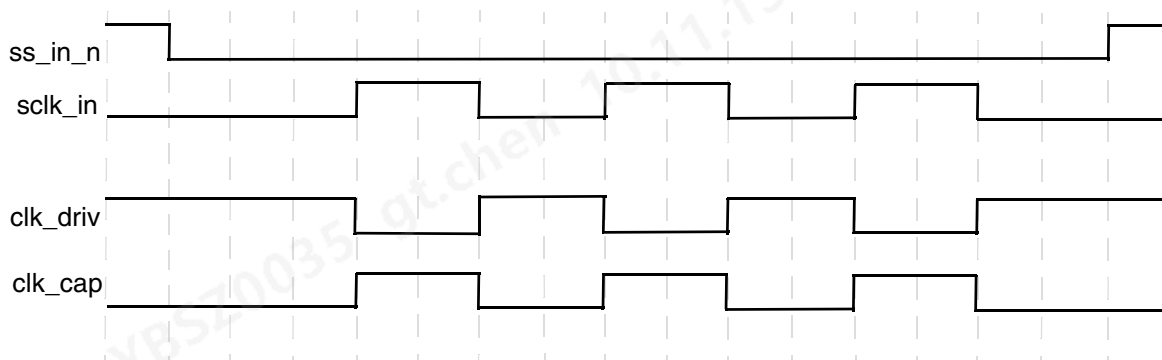
Figure 2-7   Texas Instruments Synchronous Serial Protocol (SSP)
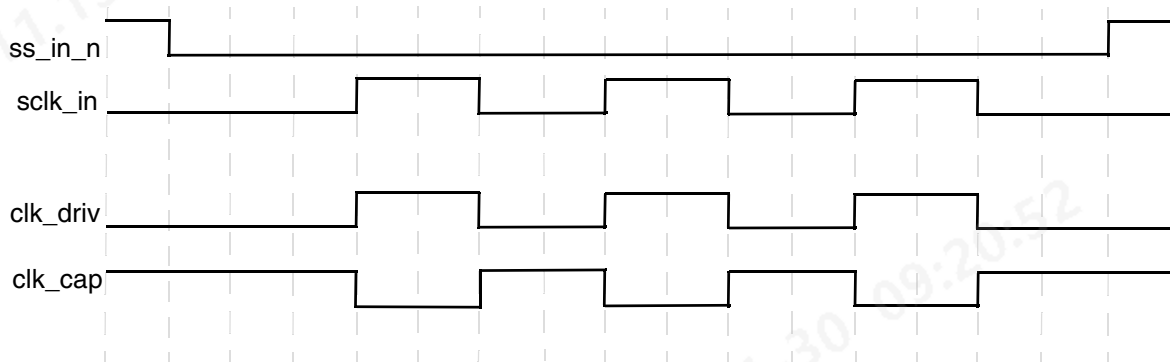
If the frame format is programmed for SPI with SCPH=0, then before the first edge of bit-rate clock (sclk_in) arrives, the data must be present on the txd line. Internally to drive data on 'txd' line the synchronized version of the slave select (ss_in_n) signal is used. This mechanism takes 2 cycles on ssi_clk to complete and the data is driven on the 3rd cycle of ssi_clk as shown in Figure 2-8. Therefore, to capture the data correctly, the first edge of bit-rate clock (sclk_in) must arrive after at least 4 ssi_clk cycles from when the slave is selected (ss_in_n).

Figure 2-8   Driving TXD When SCPH = 0
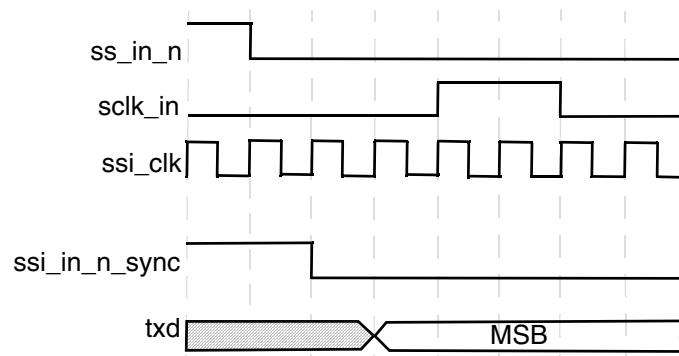
> **Note**   The SSI_ENH_CLK_RATIO mode is supported only when the *DWC-APB-Advanced-Source* source license exists.

2.2.3   Frequency Ratio Summary

A summary of the frequency ratio restrictions between the bit-rate clock (sclk_out/sclk_in) and the DW_apb_ssi peripheral clock (ssi_clk) are as follows:

- Master

  - $F_{ssi\_clk} >= 2 \times$ (maximum F_{sclk_out})

- Slave SSI_ENH_CLK_RATIO = 0

  - Receive only: $F_{ssi\_clk} >= 8 \times$ (maximum F_{sclk_in})

❑ Transmit and Receive: $F_{ssi\_clk} >= 12 \times$ (maximum F_{sclk_in})

■ Slave SSI_ENH_CLK_RATIO = 1

❑ $F_{ssi\_clk} >= 4 \times$ (maximum F_{sclk_in})

2.2.3.1   Design For Test

As explained "SSI_ENH_CLK_RATIO = 1" on page 26, when SSI_ENH_CLK_RATIO is set to 1, then sclk_in is used as capture and driving clock for rxd and txd lines, respectively. The polarity of clock is changed based on frame format being used. During the scan testing, these flops may remain uncovered. Therefore, you must connect scan_mode to chip-level scan mode. During scan mode (scan_mode =1), the clock input for these flip-flops are connected to ssi_clk, rather than the internally derived clk_driv and clk_cap. This makes register testable and subsequent downsteam points controllable.

2.3   Transmit and Receive FIFO Buffers

The FIFO buffers used by the DW_apb_ssi are internal D-type flip-flops that can be configured in depth between 2 to 256. The width of both transmit and receive FIFO buffers is fixed at 16/32-bits (depending upon SSI_MAX_XFER_SIZE), due to the serial specifications, which state that a serial transfer (data frame) can be 4 to 16/32 bits in length. Data frames that are less than 16/32-bits (depending upon SSI_MAX_XFER_SIZE) in size must be right-justified when written into the transmit FIFO buffer. The shift control logic automatically right-justifies receive data in the receive FIFO buffer.

Each data entry in the FIFO buffers contains a single data frame. It is impossible to store multiple data frames in a single FIFO location; for example, you may not store two 8-bit data frames in a single FIFO location. If an 8-bit data frame is required, the upper 8-bits of the FIFO entry are ignored or unused when the serial shifter transmits the data.

👉 **Note**   The transmit and receive FIFO buffers are cleared when the DW_apb_ssi is disabled (SSI_EN = 0) or when it is reset (presetn).

The transmit FIFO is loaded by APB write commands to the DW_apb_ssi data register (DR). Data are popped (removed) from the transmit FIFO by the shift control logic into the transmit shift register. The transmit FIFO generates a FIFO empty interrupt request (ssi_txe_intr) when the number of entries in the FIFO is less than or equal to the FIFO threshold value. The threshold value, set through the programmable register TXFTLR, determines the level of FIFO entries at which an interrupt is generated. The threshold value allows you to provide early indication to the processor that the transmit FIFO is nearly empty. A transmit FIFO overflow interrupt (ssi_txo_intr) is generated if you attempt to write data into an already full transmit FIFO.

Data are popped from the receive FIFO by APB read commands to the DW_apb_ssi data register (DR). The receive FIFO is loaded from the receive shift register by the shift control logic. The receive FIFO generates a FIFO-full interrupt request (ssi_rxf_intr) when the number of entries in the FIFO is greater than or equal to the FIFO threshold value plus 1. The threshold value, set through programmable register RXFTLR, determines the level of FIFO entries at which an interrupt is generated.

The threshold value allows you to provide early indication to the processor that the receive FIFO is nearly full. A receive FIFO overrun interrupt (ssi_rxo_intr) is generated when the receive shift logic attempts to load data into a completely full receive FIFO. However, this newly received data are lost. A receive FIFO

underflow interrupt (ssi_rxu_intr) is generated if you attempt to read from an empty receive FIFO. This alerts the processor that the read data are invalid.

Table 2-1 provides description for different Transmit FIFO Threshold values.

Table 2-1   Transmit FIFO Threshold (TFT) Decode Values

TFT Value	Description
0000_0000	ssi_txe_intr is asserted when 0 data entries are present in transmit FIFO
0000_0001	ssi_txe_intr is asserted when 1 or less data entry is present in transmit FIFO
0000_0010	ssi_txe_intr is asserted when 2 or less data entries are present in transmit FIFO
0000_0011	ssi_txe_intr is asserted when 3 or less data entries are present in transmit FIFO
...	...
...	...
1111_1100	ssi_txe_intr is asserted when 252 or less data entries are present in transmit FIFO
1111_1101	ssi_txe_intr is asserted when 253 or less data entries are present in transmit FIFO
1111_1110	ssi_txe_intr is asserted when 254 or less data entries are present in transmit FIFO
1111_1111	ssi_txe_intr is asserted when 255 or less data entries are present in transmit FIFO

Table 2-2 provides description for different Receive FIFO Threshold values.

Table 2-2   Receive FIFO Threshold (TFT) Decode Values

RFT Value	Description
0000_0000	ssi_rxf_intr is asserted when 1 or more data entry is present in receive FIFO
0000_0001	ssi_rxf_intr is asserted when 2 or more data entries are present in receive FIFO
0000_0010	ssi_rxf_intr is asserted when 3 or more data entries are present in receive FIFO
0000_0011	ssi_rxf_intr is asserted when 4 or more data entries are present in receive FIFO
...	...
...	...
1111_1100	ssi_rxf_intr is asserted when 253 or more data entries are present in receive FIFO
1111_1101	ssi_rxf_intr is asserted when 254 or more data entries are present in receive FIFO
1111_1110	ssi_rxf_intr is asserted when 255 or more data entries are present in receive FIFO
1111_1111	ssi_rxf_intr is asserted when 256 data entries are present in receive FIFO

2.4   32-Bit Frame Size Support

The SSI_MAX_XFER_SIZE configuration parameter is used to select the maximum programmable value in of data frame size. The following changes occur in the core when the SSI_MAX_XFER_SIZE is configured to 32 bits:

- dfs_32 (CTRLR0[20:16]) becomes valid, which contains the value of data frame size. The new register field holds the values 0 to 31. The dfs (CTRLR0[3:0]) becomes invalid and writing to this register has no effect.

- The receive and transmit FIFO widths increase from 16 to 32 bits.

- All 32 bits of the data register become valid. (For more information, see "Data Register APB Access" on page 88.)

2.5   SSI Interrupts

The DW_apb_ssi supports combined and individual interrupt requests, each of which can be masked. The combined interrupt request is the ORed result of all other DW_apb_ssi interrupts after masking. The system designer has the choice of routing individual interrupt requests or only the combined interrupt request to the Interrupt Controller. All DW_apb_ssi interrupts are level interrupts and have the same active polarity level; you can configure this polarity level as active-high or active-low.

The DW_apb_ssi interrupts are described as follows:

- Transmit FIFO Empty Interrupt (ssi_txe_intr) – Set when the transmit FIFO is equal to or below its threshold value and requires service to prevent an under-run. The threshold value, set through a software-programmable register, determines the level of transmit FIFO entries at which an interrupt is generated. This interrupt is cleared by hardware when data are written into the transmit FIFO buffer, bringing it over the threshold level.

- Transmit FIFO Overflow Interrupt (ssi_txo_intr) – Set when an APB access attempts to write into the transmit FIFO after it has been completely filled. When set, data written from the APB is discarded. This interrupt remains set until you read the transmit FIFO overflow interrupt clear register (TXOICR).

- Receive FIFO Full Interrupt (ssi_rxf_intr) – Set when the receive FIFO is equal to or above its threshold value plus 1 and requires service to prevent an overflow. The threshold value, set through a software-programmable register, determines the level of receive FIFO entries at which an interrupt is generated. This interrupt is cleared by hardware when data are read from the receive FIFO buffer, bringing it below the threshold level.

- Receive FIFO Overflow Interrupt (ssi_rxo_intr) – Set when the receive logic attempts to place data into the receive FIFO after it has been completely filled. When set, newly received data are discarded. This interrupt remains set until you read the receive FIFO overflow interrupt clear register (RXOICR).

- Receive FIFO Underflow Interrupt (ssi_rxu_intr) – Set when an APB access attempts to read from the receive FIFO when it is empty. When set, zeros are read back from the receive FIFO. This interrupt remains set until you read the receive FIFO underflow interrupt clear register (RXUICR).

- Multi-Master Contention Interrupt (ssi_mst_intr) – Present only when the DW_apb_ssi component is configured as a serial-master device. The interrupt is set when another serial master on the serial bus selects the DW_apb_ssi master as a serial-slave device and is actively transferring data. This informs the processor of possible contention on the serial bus. This interrupt remains set until you read the multi-master interrupt clear register (MSTICR).

- Combined Interrupt Request (ssi_intr) – ORed result of all the above interrupt requests after masking. To mask this interrupt signal, you must mask all other DW_apb_ssi interrupt requests.

For more information on interrupts, see Appendix 4, "Signal Descriptions".

2.6   Transfer Modes

When transferring data on the serial bus, the DW_apb_ssi operates in the modes discussed in this section.

The transfer mode (TMOD) is set by writing to control register 0 (CTRLR0), as described in "Register Descriptions" on page 123.

👉 **Note**   The transfer mode setting does not affect the duplex of the serial transfer. TMOD is ignored for Microwire transfers, which are controlled by the MWCR register.

2.6.1   Transmit and Receive

When TMOD = 2′b00, both transmit and receive logic are valid. The data transfer occurs as normal according to the selected frame format (serial protocol). Transmit data are popped from the transmit FIFO and sent through the txd line to the target device, which replies with data on the rxd line. The receive data from the target device is moved from the receive shift register into the receive FIFO at the end of each data frame.

2.6.2   Transmit Only

When TMOD = 2′b01, the receive data are invalid and should not be stored in the receive FIFO. The data transfer occurs as normal, according to the selected frame format (serial protocol). Transmit data are popped from the transmit FIFO and sent through the txd line to the target device, which replies with data on the rxd line. At the end of the data frame, the receive shift register does not load its newly received data into the receive FIFO. The data in the receive shift register is overwritten by the next transfer. You should mask interrupts originating from the receive logic when this mode is entered.

2.6.3   Receive Only

When TMOD = 2′b10, the transmit data are invalid. When configured as a slave, the transmit FIFO is never popped in Receive Only mode. The txd output remains at a constant logic level during the transmission. The data transfer occurs as normal according to the selected frame format (serial protocol). The receive data from the target device is moved from the receive shift register into the receive FIFO at the end of each data frame. You should mask interrupts originating from the transmit logic when this mode is entered.

2.6.4   EEPROM Read

> **☞ Note**   This transfer mode is only valid for master configurations.

When TMOD = 2'b11, the transmit data is used to transmit an opcode and/or an address to the EEPROM device. Typically this takes three data frames (8-bit opcode followed by 8-bit upper address and 8-bit lower address). During the transmission of the opcode and address, no data is captured by the receive logic (as long as the DW_apb_ssi master is transmitting data on its txd line, data on the rxd line is ignored). The DW_apb_ssi master continues to transmit data until the transmit FIFO is empty. Therefore, you should ONLY have enough data frames in the transmit FIFO to supply the opcode and address to the EEPROM. If more data frames are in the transmit FIFO than are needed, then read data is lost.

When the transmit FIFO becomes empty (all control information has been sent), data on the receive line (rxd) is valid and is stored in the receive FIFO; the txd output is held at a constant logic level. The serial transfer continues until the number of data frames received by the DW_apb_ssi master matches the value of the NDF field in the CTRLR1 register + 1.

> **☞ Note**   EEPROM read mode is not supported when the DW_apb_ssi is configured to be in the SSP mode.

2.7   Operation Modes

The DW_apb_ssi can be configured in the fundamental modes of operation discussed in this section.

2.7.1   Serial Master Mode

This mode enables serial communication with serial-slave peripheral devices. When configured as a serial-master device, the DW_apb_ssi initiates and controls all serial transfers. Figure 2-9 shows an example of the DW_apb_ssi configured as a serial master with all other devices on the serial bus configured as serial slaves.

Figure 2-9   DW_apb_ssi Configured as Master Device
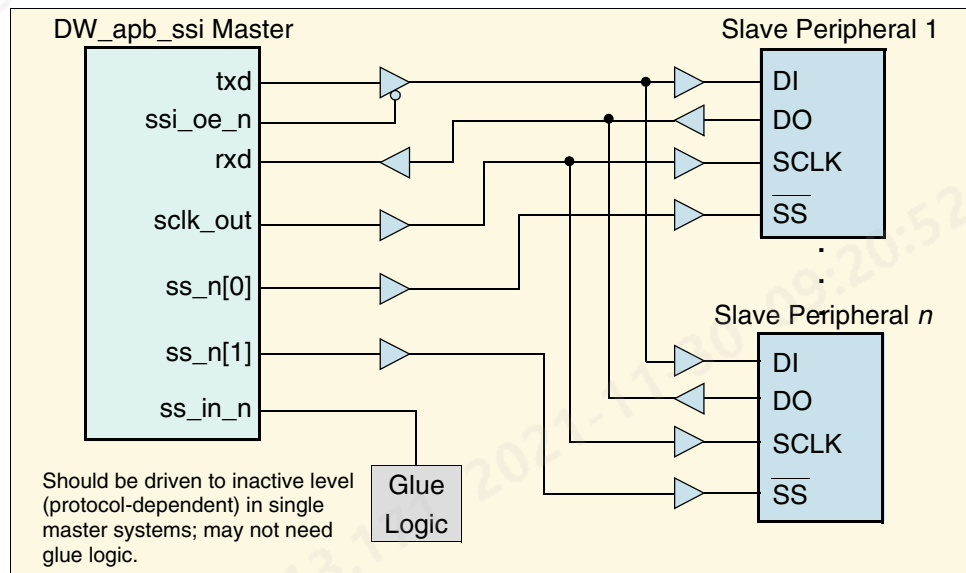
The serial bit-rate clock, generated and controlled by the DW_apb_ssi, is driven out on the sclk_out line. When the DW_apb_ssi is disabled (SSI_EN = 0), no serial transfers can occur and sclk_out is held in "inactive" state, as defined by the serial protocol under which it operates.

2.7.1.1   Master Contention Input

The DW_apb_ssi master configuration has a serial slave select input, ss_in_n, that can be used to inform the DW_apb_ssi master that another serial master is active on the bus. When this input is active—the active level depends on the serial protocol—the DW_apb_ssi master remains in an IDLE state and holds off any pending serial transfer until the ss_in_n input is returned to an in-active level.

You should use the ss_in_n input to assist arbitration between multiple serial bus masters. A simple usage example is shown in Figure 2-10.

Figure 2-10   Arbitration Between Multiple Serial Masters
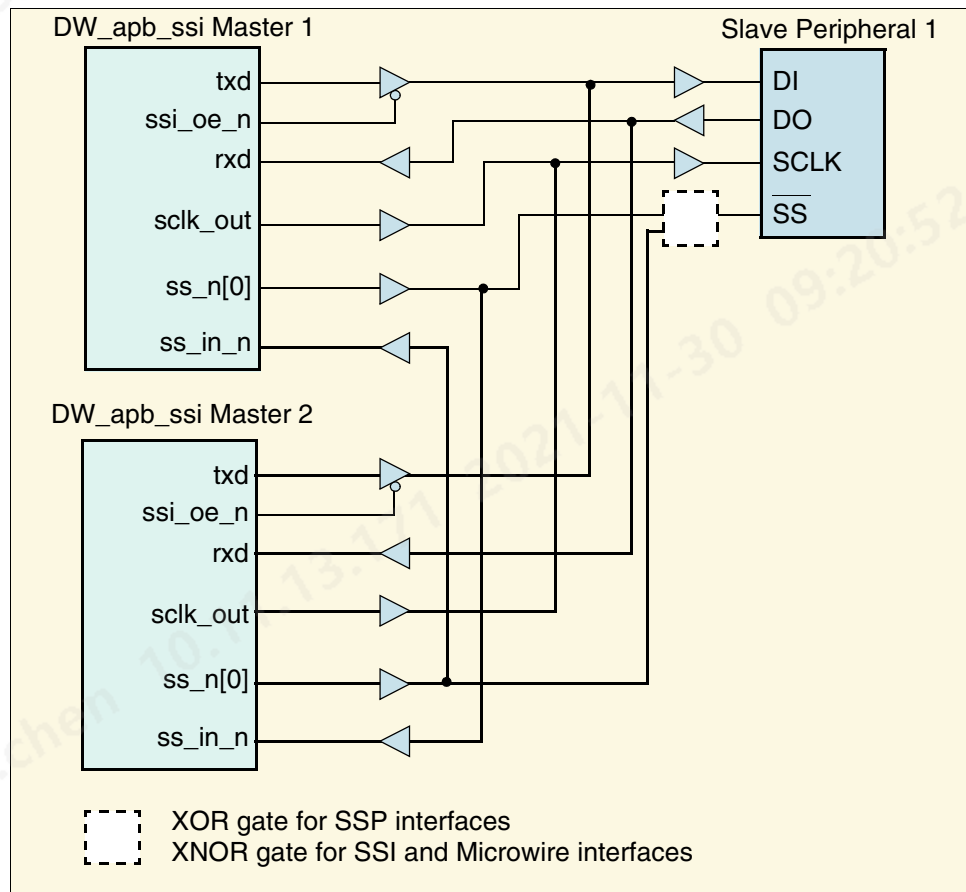
In this example, it is a case of "first-come-first-served" arbitration. Although the example does create the potential for locking the bus, if both masters assert their slave select outputs on the same clock edge, it shows how the ss_in_n signal can be used. A more complex arbiter block, that obeys the principles illustrated in the figure, should be used to arbitrate between master select outputs, slave select inputs and master select inputs, to prevent any potential bus locking occurrences.

If the DW_apb_ssi master is the only master device on the serial bus, you might need to insert some glue logic to control the logic level on the master ss_in_n input.

Glue logic is required if both of the following are true:

- You dynamically change the serial protocol
- One of the protocols being used is SSP

There are several methods for implementing this glue logic; Figure 2-11 illustrates an example architecture.

Figure 2-11   Glue Logic for Controlling Logic Level on Master ss_in_n Input
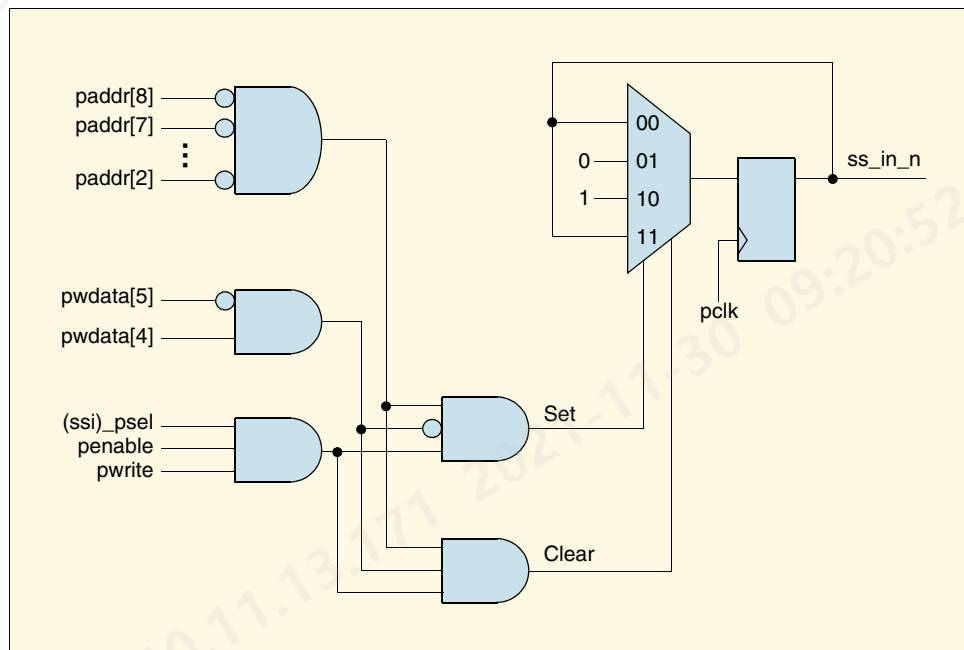
In this architecture, the ss_in_n signal is driven low when you writes 2'b01 (SSP) into the FRF bit field in control register 0 (CTRLR0). The ss_in_n signal is driven high for all other values written into the FRF bit field.

☞ **Note**   If the Default Frame Format in the DW_apb_ssi is not SSP, the register shown in the diagram below should reset to 1.

Glue logic is not required under either of the following conditions:

- If you never intend to dynamically change the serial protocol that the DW_apb_ssi master is operating under.

- If you do change the serial protocol dynamically but do not use the SSP protocol.

Under these conditions, the ss_in_n signal can be tied high or low depending on which serial protocol you use.

- If the serial protocol is SPI or MicroWire, the ss_in_n signal should be tied high.

- If the serial protocol being used is SSP, the ss_in_n signal should be tied low.

2.7.1.2   RXD Sample Delay

When the DW_apb_ssi is configured as a master, additional logic can be included in the design in order to delay the default sample time of the rxd signal. This additional logic can help to increase the maximum achievable frequency on the serial bus.

To include this additional logic, the SSI_HAS_RX_SAMPLE_DELAY parameter should have a value of 1.

Round trip routing delays on the sclk_out signal from the master and the rxd signal from the slave can mean that the timing of the rxd signal—as seen by the master—has moved away from the normal sampling time. Figure 2-12 illustrates this situation.

Figure 2-12   Effects of Round-Trip Routing Delays on sclk_out Signal
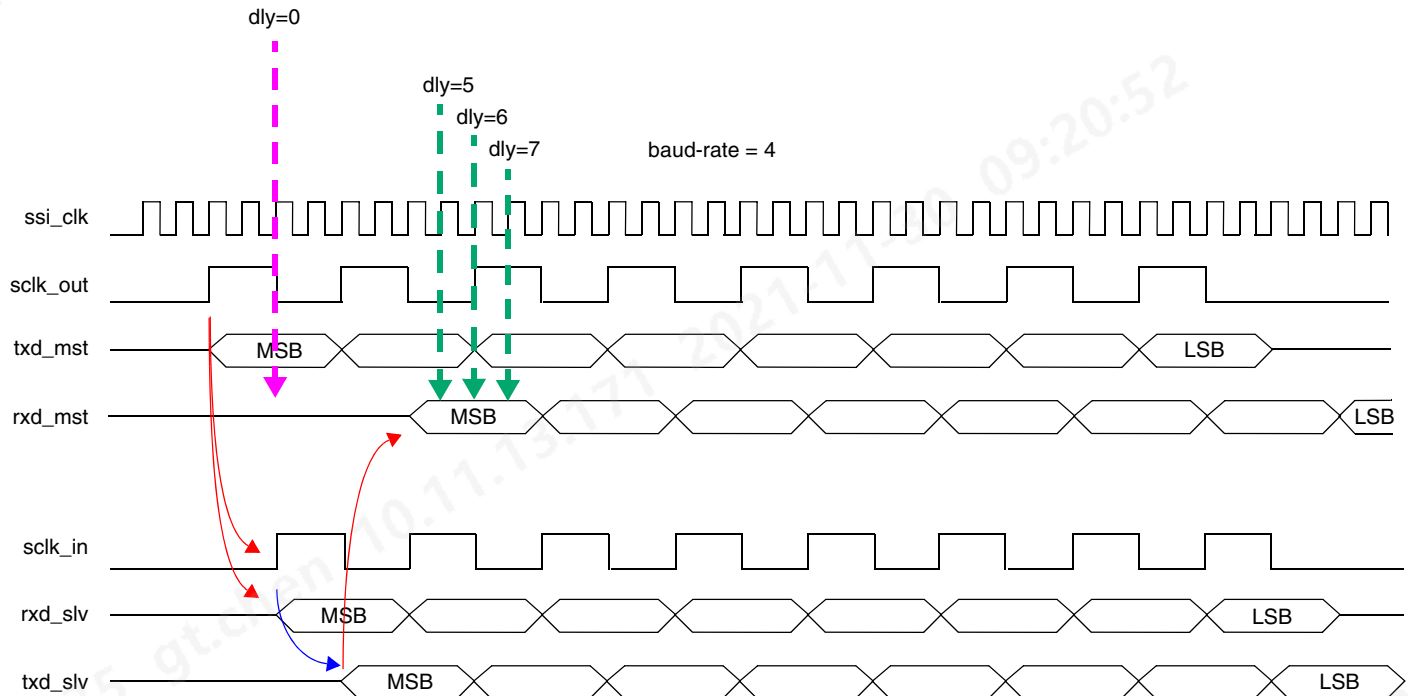
Red arrows indicate routing delay between master and slave devices

Blue arrow indicates sampling delay within slave from receiving slk_in to driving txd out

The Slave uses the sclk_out signal from the master as a strobe in order to drive rxd signal data onto the serial bus. Routing and sampling delays on the sclk_out signal by the slave device can mean that the rxd bit has not stabilized to the correct value before the master samples the rxd signal. Figure 2-12 shows an example of how a routing delay on the rxd signal can result in an incorrect rxd value at the default time when the master samples the port.

Without the RXD Sample Delay logic, you would have to increase the baud-rate for the transfer in order to ensure that the setup times on the rxd signal are within range; this results in reducing the frequency of the serial interface.

When the RXD Sample Delay logic is included, you can dynamically program a delay value in order to move the sampling time of the rxd signal equal to a number of ssi_clk cycles from the default.

The sample delay logic has a resolution of one ssi_clk cycle. Software can "train" the serial bus by coding a loop that continually reads from the slave and increments the master's RXD Sample Delay value until the correct data is received by the master.

2.7.1.3   Data Transfers

Data transfers are started by the serial-master device. When the DW_apb_ssi is enabled (SSI_EN=1), at least one valid data entry is present in the transmit FIFO and a serial-slave device is selected. When actively

transferring data, the busy flag (BUSY) in the status register (SR) is set. You must wait until the busy flag is cleared before attempting a new serial transfer.

☞ **Note**　　The BUSY status is not set when the data are written into the transmit FIFO. This bit gets set only when the target slave has been selected and the transfer is underway. After writing data into the transmit FIFO, the shift logic does not begin the serial transfer until a positive edge of the sclk_out signal is present. The delay in waiting for this positive edge depends on the baud rate of the serial transfer. Before polling the BUSY status, you should first poll the TFE status (waiting for 1) or wait for BAUDR * ssi_clk clock cycles.

2.7.1.4　　Master SPI and SSP Serial Transfers

The sections "Motorola Serial Peripheral Interface (SPI)" on page 47 and "Texas Instruments Synchronous Serial Protocol (SSP)" on page 53 describe the SPI and SSP serial protocols, respectively. They include timing diagrams and provide information as to how data are structured in the transmit and receive the FIFOs before and after the serial transfer.

When the transfer mode is "transmit and receive" or "transmit only" (TMOD = 2'b00 or TMOD = 2'b01, respectively), transfers are terminated by the shift control logic when the transmit FIFO is empty. For continuous data transfers, you must ensure that the transmit FIFO buffer does not become empty before all the data have been transmitted. The transmit FIFO threshold level (TXFTLR) can be used to early interrupt (ssi_txe_intr) the processor indicating that the transmit FIFO buffer is nearly empty. When a DMA is used for APB accesses, the transmit data level (DMATDLR) can be used to early request (dma_tx_req) the DMA Controller, indicating that the transmit FIFO is nearly empty. The FIFO can then be refilled with data to continue the serial transfer. You may also write a block of data (at least two FIFO entries) into the transmit FIFO before enabling a serial slave. This ensures that serial transmission does not begin until the number of data-frames that make up the continuous transfer are present in the transmit FIFO.

DW_apb_ssi starts the transfer once it receives the data in TX FIFO and keeps on sending the data until the TX FIFO becomes empty. In some of the cases if the slave interface (APB) is not able to keep up with the data rate of SPI interface the transmit FIFO may become empty during the transfer and the slave gets de-selected. That is what is being observed over here. The data availability on the transmit FIFO has to be made sure by the software so that this situation never occurs and transfer completes without any unexpected chip select de-assertion.

In order to make sure the data is present before the transfer starts, data register could be programmed earlier and then SER should be programmed. This ensures that all the required data is present in the FIFO before the transfer starts on the SPI interface. However, ideally software should make sure that the data rate on programming interface is sufficient as to keep up with data demands on serial interface.

When the transfer mode is "receive only" (TMOD = 2'b10), a serial transfer is started by writing one "dummy" data word into the transmit FIFO when a serial slave is selected. The txd output from the DW_apb_ssi is held at a constant logic level for the duration of the serial transfer. The transmit FIFO is popped only once at the beginning and may remain empty for the duration of the serial transfer. The end of the serial transfer is controlled by the "number of data frames" (NDF) field in control register 1 (CTRLR1).

If, for example, you want to receive 24 data frames from a serial-slave peripheral, you should program the NDF field with the value 23; the receive logic terminates the serial transfer when the number of frames received is equal to the NDF value + 1. This transfer mode increases the bandwidth of the APB bus as the transmit FIFO never needs to be serviced during the transfer. The receive FIFO buffer should be read each time the receive FIFO generates a FIFO full interrupt request to prevent an overflow.

When the transfer mode is "eeprom_read" (TMOD = 2'b11), a serial transfer is started by writing the opcode and/or address into the transmit FIFO when a serial slave (EEPROM) is selected. The opcode and address are transmitted to the EEPROM device, after which read data is received from the EEPROM device and stored in the receive FIFO. The end of the serial transfer is controlled by the NDF field in the control register 1 (CTRLR1).

👉 **Note**   EEPROM read mode is not supported when the DW_apb_ssi is configured to be in the SSP mode.

The receive FIFO threshold level (RXFTLR) can be used to give early indication that the receive FIFO is nearly full. When a DMA is used for APB accesses, the receive data level (DMARDLR) can be used to early request (dma_rx_req) the DMA Controller, indicating that the receive FIFO is nearly full.

A typical software flow for completing an SPI or SSP serial transfer from the DW_apb_ssi serial master is outlined as follows:

1. If the DW_apb_ssi is enabled, disable it by writing 0 to the SSI Enable register (SSIENR).

2. Set up the DW_apb_ssi control registers for the transfer; these registers can be set in any order.

    ❑ Write Control Register 0 (CTRLR0). For SPI transfers, the serial clock polarity and serial clock phase parameters must be set identical to target slave device.

    ❑ If the transfer mode is *receive only*, write CTRLR1 (Control Register 1) with the number of frames in the transfer minus 1; for example, if you want to receive four data frames, write this register with 3.

    ❑ Write the Baud Rate Select Register (BAUDR) to set the baud rate for the transfer.

    ❑ Write the Transmit and Receive FIFO Threshold Level registers (TXFTLR and RXFTLR, respectively) to set FIFO threshold levels.

    ❑ Write the IMR register to set up interrupt masks.

    ❑ The Slave Enable Register (SER) register can be written here to enable the target slave for selection. If a slave is enabled here, the transfer begins as soon as one valid data entry is present in the transmit FIFO. If no slaves are enabled prior to writing to the Data Register (DR), the transfer does not begin until a slave is enabled.

3. Enable the DW_apb_ssi by writing 1 to the SSIENR register.

4. Write data for transmission to the target slave into the transmit FIFO (write DR).

    If no slaves were enabled in the SER register at this point, enable it now to begin the transfer.

5. Poll the BUSY status to wait for completion of the transfer. The BUSY status cannot be polled immediately; for more information, see the note on .

    If a transmit FIFO empty interrupt request is made, write the transmit FIFO (write DR). If a receive FIFO full interrupt request is made, read the receive FIFO (read DR).

6. The transfer is stopped by the shift control logic when the transmit FIFO is empty. If the transfer mode is *receive only* (TMOD = 2'b10), the transfer is stopped by the shift control logic when the specified number of frames have been received. When the transfer is done, the BUSY status is reset to 0.

7. If the transfer mode is not *transmit only* (TMOD != 01), read the receive FIFO until it is empty.

8. Disable the DW_apb_ssi by writing 0 to SSIENR.

Figure 2-13 shows a typical software flow for starting a DW_apb_ssi master SPI/SSP serial transfer. The diagram also shows the hardware flow inside the serial-master component.

Figure 2-13   DW_apb_ssi Master SPI/SSP Transfer Flow
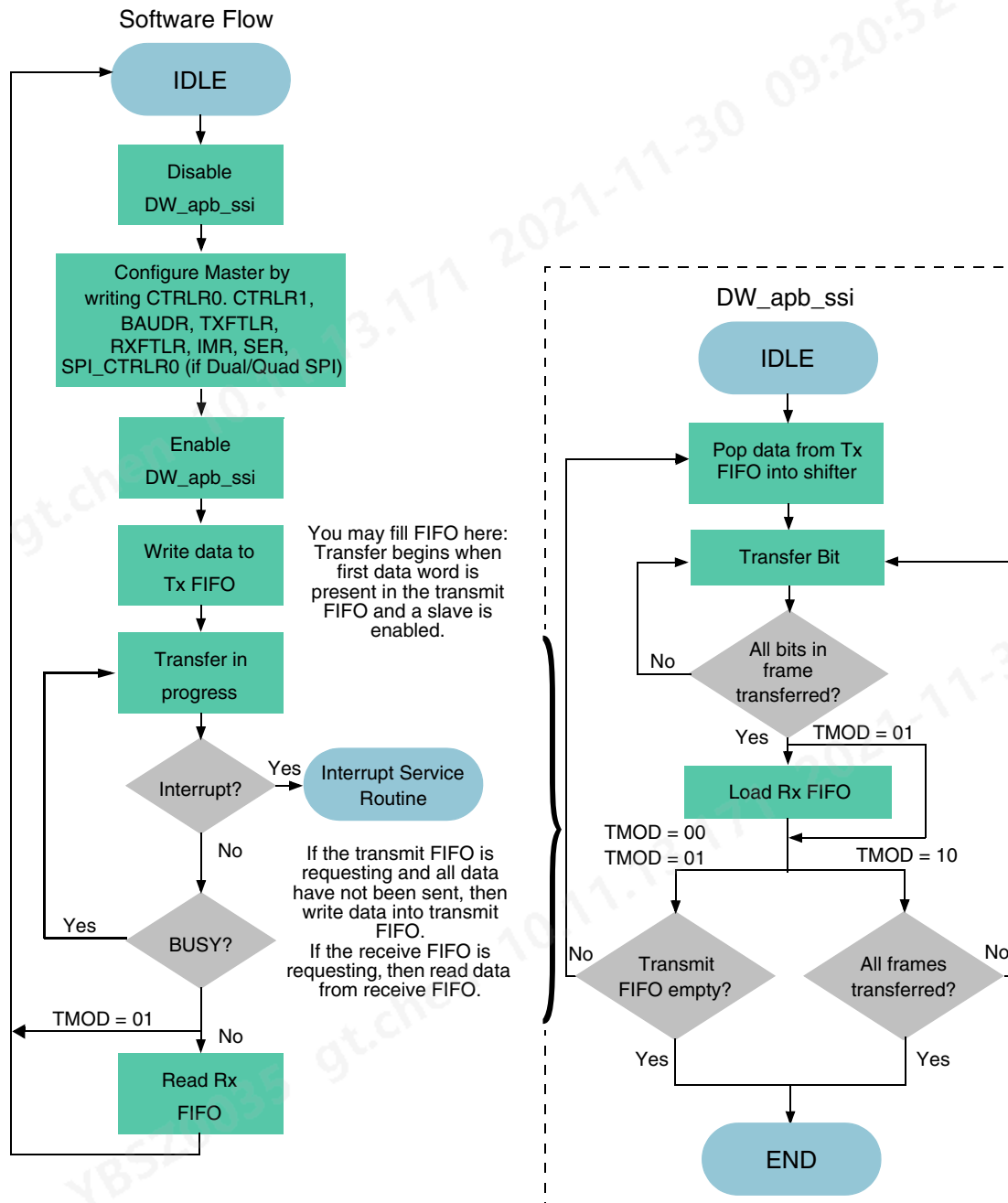
DW_apb_ssi starts the transfer after it receives the data in the TX FIFO. DW_apb_ssi continues to send the data until the TX FIFO is empty.

If the application interface (APB) is unable to keep up the pace with the data transfer rate of the SPI interface, then the transmit FIFO may become empty, and thereby the slave is de-selected.

In such scenarios, the application should make sure that the TX FIFO does not become empty in the middle of the transfer, and the transfer completes without any unexpected chip select de-selection.

Data Register (DR) can be programmed to make sure that the required data is available before the transfer starts. Then, application can select the slave by programming the SER register.

This ensures that the required data is present in the FIFO before the transfer starts on the SPI interface. However, the software must make sure that the data rate on the programming interface (APB) is sufficient to keep up with the data demand on the serial interface.

2.7.1.5   Master Microwire Serial Transfers

"National Semiconductor Microwire" on page 54 describes the Microwire serial protocol in detail, including timing diagrams and explaining how data are structured in the transmit and receive FIFOs before and after a serial transfer.

Microwire serial transfers from the DW_apb_ssi serial master are controlled by the Microwire Control Register (MWCR). The MWHS bit field enables and disables the Microwire handshaking interface. The MDD bit field controls the direction of the data frame (the control frame is always transmitted by the master and received by the slave). The MWMOD bit field defines whether the transfer is sequential or nonsequential.

All Microwire transfers are started by the DW_apb_ssi serial master when there is at least one control word in the transmit FIFO and a slave is enabled. When the DW_apb_ssi master transmits the data frame (MDD = 1), the transfer is terminated by the shift logic when the transmit FIFO is empty. When the DW_apb_ssi master receives the data frame (MDD = 1), the termination of the transfer depends on the setting of the MWMOD bit field. If the transfer is nonsequential (MWMOD = 0), it is terminated when the transmit FIFO is empty after shifting in the data frame from the slave. When the transfer is sequential (MWMOD = 1), it is terminated by the shift logic when the number of data frames received is equal to the value in the CTRLR1 register + 1.

When the handshaking interface on the DW_apb_ssi master is enabled (MWHS =1), the status of the target slave is polled after transmission. Only when the slave reports a *ready status* does the DW_apb_ssi master complete the transfer and clear its BUSY status. If the transfer is continuous, the next control/data frame is not sent until the slave device returns a *ready status*.

A typical software flow for completing a Microwire serial transfer from the DW_apb_ssi serial master is outlined as follows:

1. If the DW_apb_ssi is enabled, disable it by writing 0 to SSIENR.

2. Set up the DW_apb_ssi control registers for the transfer. These registers can be set in any order. Write CTRLR0 to set transfer parameters.

   ❑ If the transfer is sequential and the DW_apb_ssi master receives data, write CTRLR1 with the number of frames in the transfer minus 1; for instance, if you want to receive four data frames, write this register with 3.

   ❑ Write BAUDR to set the baud rate for the transfer.

   ❑ Write TXFTLR and RXFTLR to set FIFO threshold levels.

   ❑ Write the IMR register to set up interrupt masks.

You can write the SER register to enable the target slave for selection. If a slave is enabled here, the transfer begins as soon as one valid data entry is present in the transmit FIFO. If no slaves are enabled prior to writing to the DR register, the transfer does not begin until a slave is enabled.

3. Enable the DW_apb_ssi by writing 1 to the SSIENR register.

4. If the DW_apb_ssi master transmits data, write the control and data words into the transmit FIFO (write DR). If the DW_apb_ssi master receives data, write the control words into the transmit FIFO.

   If no slaves were enabled in the SER register at this point, enable now to begin the transfer.

5. Poll the BUSY status to wait for completion of the transfer. The BUSY status cannot be polled immediately; for more information, see the note on page 38.

   If a transmit FIFO empty interrupt request is made, write the transmit FIFO (write DR). If a receive FIFO full interrupt request is made, read the receive FIFO (read DR).

6. The transfer is stopped by the shift control logic when the transmit FIFO is empty. If the transfer mode is sequential and the DW_apb_ssi master receives data, the transfer is stopped by the shift control logic when the specified number of data frames is received. When the transfer is done, the BUSY status is reset to 0.

7. If the DW_apb_ssi master receives data, read the receive FIFO until it is empty.

8. Disable the DW_apb_ssi by writing 0 to SSIENR.

Figure 2-14 shows a typical software flow for starting a DW_apb_ssi master Microwire serial transfer. The diagram also shows the hardware flow inside the serial-master component.

Figure 2-14   DW_apb_ssi Master Microwire Transfer Flow
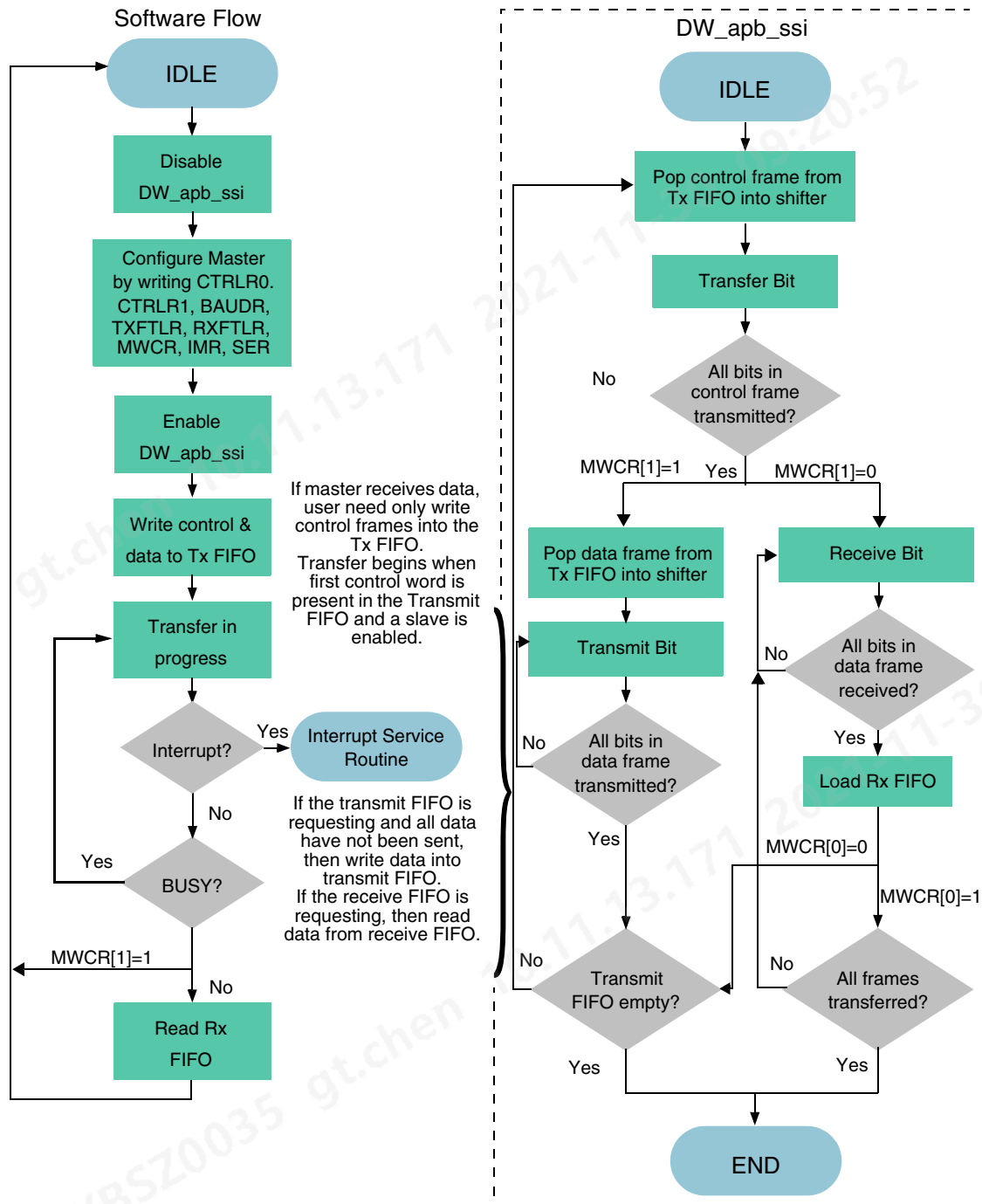
2.7.2   Serial-Slave Mode

This mode enables serial communication with master peripheral devices. When the DW_apb_ssi is configured as a slave device, all serial transfers are initiated and controlled by the serial bus master. Figure 2-15 shows an example of the DW_apb_ssi configured as a serial slave in a single-master bus system.

When the DW_apb_ssi serial slave is selected during configuration, it enables its txd data onto the serial bus. All data transfers to and from the serial slave are regulated on the serial clock line (sclk_in), driven from the serial-master device. Data are propagated from the serial slave on one edge of the serial clock line and sampled on the opposite edge.

Figure 2-15   DW_apb_ssi Configured as Slave Device
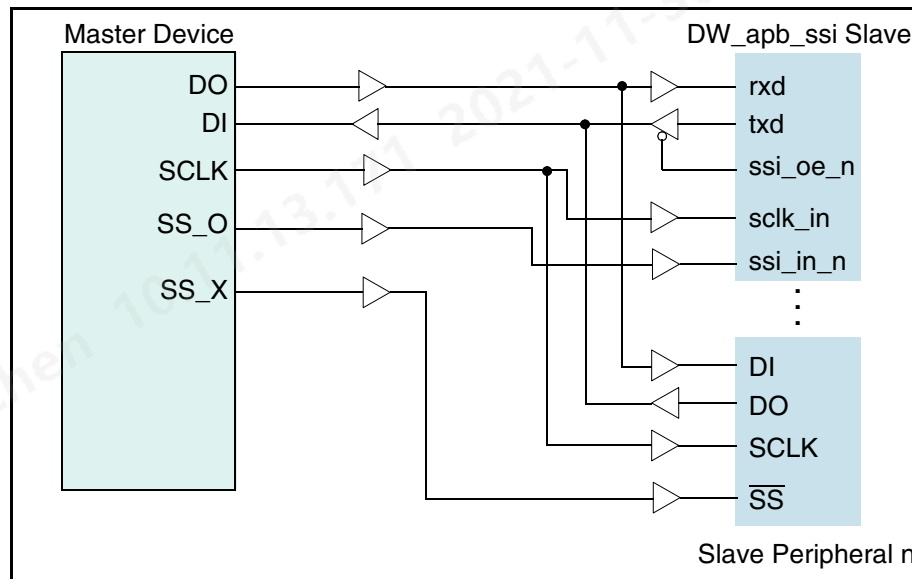
When the DW_apb_ssi serial slave is not selected, it must not interfere with data transfers between the serial-master and other serial-slave devices. When the serial slave is not selected, its txd output is buffered, resulting in a high impedance drive onto the serial master rxd line. The buffers shown in Figure 2-15 are external to DW_apb_ssi.

The serial clock that regulates the data transfer is generated by the serial-master device and input to the DW_apb_ssi slave on sclk_in. The slave remains in an idle state until selected by the bus master. When not actively transmitting data, the slave must hold its txd line in a high impedance state to avoid interference with serial transfers to other slave devices. The ssi_oe_n line is available for use to control the txd output buffer. The slave continues to transfer data to and from the master device as long as it is selected. If the master transmits to all serial slaves, a control bit (SLV_OE) in the DW_apb_ssi control register 0 (CTRLR0) can be programmed to inform the slave if it should respond with data from the its txd line.

2.7.2.1   Slave SPI and SSP Serial Transfers

"Motorola Serial Peripheral Interface (SPI)" on page 47 and "Texas Instruments Synchronous Serial Protocol (SSP)" on page 53 contain a description of the SPI and SSP serial protocols, respectively. The sections also provide timing diagrams and information on how data are structured in the transmit and receive FIFOs before and after the serial transfer.

If the DW_apb_ssi slave is *receive only* (TMOD=10), the transmit FIFO need not contain valid data because the data currently in the transmit shift register is resent each time the slave device is selected. The TXE error flag in the status register (SR) is not set when TMOD=01. You should mask the transmit FIFO empty interrupt when this mode is used.

If the DW_apb_ssi slave transmits data to the master, you must ensure that data exists in the transmit FIFO before a transfer is initiated by the serial-master device. If the master initiates a transfer to the DW_apb_ssi slave when no data exists in the transmit FIFO, an error flag (TXE) is set in the DW_apb_ssi status register, and the previously transmitted data frame is resent on txd. For continuous data transfers, you must ensure that the transmit FIFO buffer does not become empty before all the data have been transmitted. The transmit FIFO threshold level register (TXFTLR) can be used to early interrupt (ssi_txe_intr) the processor, indicating that the transmit FIFO buffer is nearly empty. When a DMA Controller is used for APB accesses, the DMA transmit data level register (DMATDLR) can be used to early request (dma_tx_req) the DMA Controller, indicating that the transmit FIFO is nearly empty. The FIFO can then be refilled with data to continue the serial transfer. The receive FIFO buffer should be read each time the receive FIFO generates a FIFO full interrupt request to prevent an overflow. The receive FIFO threshold level register (RXFTLR) can be used to give early indication that the receive FIFO is nearly full. When a DMA Controller is used for APB accesses, the DMA receive data level register (DMARDLR) can be used to early request (dma_rx_req) the DMA controller, indicating that the receive FIFO is nearly full.

A typical software flow for completing a continuous serial transfer from a serial master to the DW_apb_ssi slave is described as follows:

1. If the DW_apb_ssi is enabled, disable it by writing 0 to SSIENR.

2. Set up the DW_apb_ssi control registers for the transfer. These registers can be set in any order.

   a. Write CTRLR0 (for SPI transfers SCPH and SCPOL must be set identical to the master device).

   b. Write TXFTLR and RXFTLR to set FIFO threshold levels.

   c. Write the IMR register to set up interrupt masks.

3. Enable the DW_apb_ssi by writing 1 to the SSIENR register.

4. If the transfer mode is *transmit and receive* (TMOD=2'b00) or *transmit only* (TMOD=2'b01), write data for transmission to the master into the transmit FIFO (Write DR).

   If the transfer mode is *receive only* (TMOD=2'b10), there is no need to write data into the transmit FIFO; the current value in the transmit shift register is retransmitted.

5. The DW_apb_ssi slave is now ready for the serial transfer. The transfer begins when the DW_apb_ssi slave is selected by a serial-master device.

6. When the transfer is underway, the BUSY status can be polled to return the transfer status. If a transmit FIFO empty interrupt request is made, write the transmit FIFO (write DR). If a receive FIFO full interrupt request is made, read the receive FIFO (read DR).

7. The transfer ends when the serial master removes the select input to the DW_apb_ssi slave. When the transfer is completed, the BUSY status is reset to 0.

8. If the transfer mode is not *transmit only* (TMOD != 01), read the receive FIFO until empty.

9. Disable the DW_apb_ssi by writing 0 to SSIENR.

Figure 2-16 shows a typical software flow for a DW_apb_ssi slave SPI or SSP serial transfer. The diagram also shows the hardware flow inside the serial-slave component.

Figure 2-16   DW_apb_ssi Slave SPI/SSP Transfer Flow
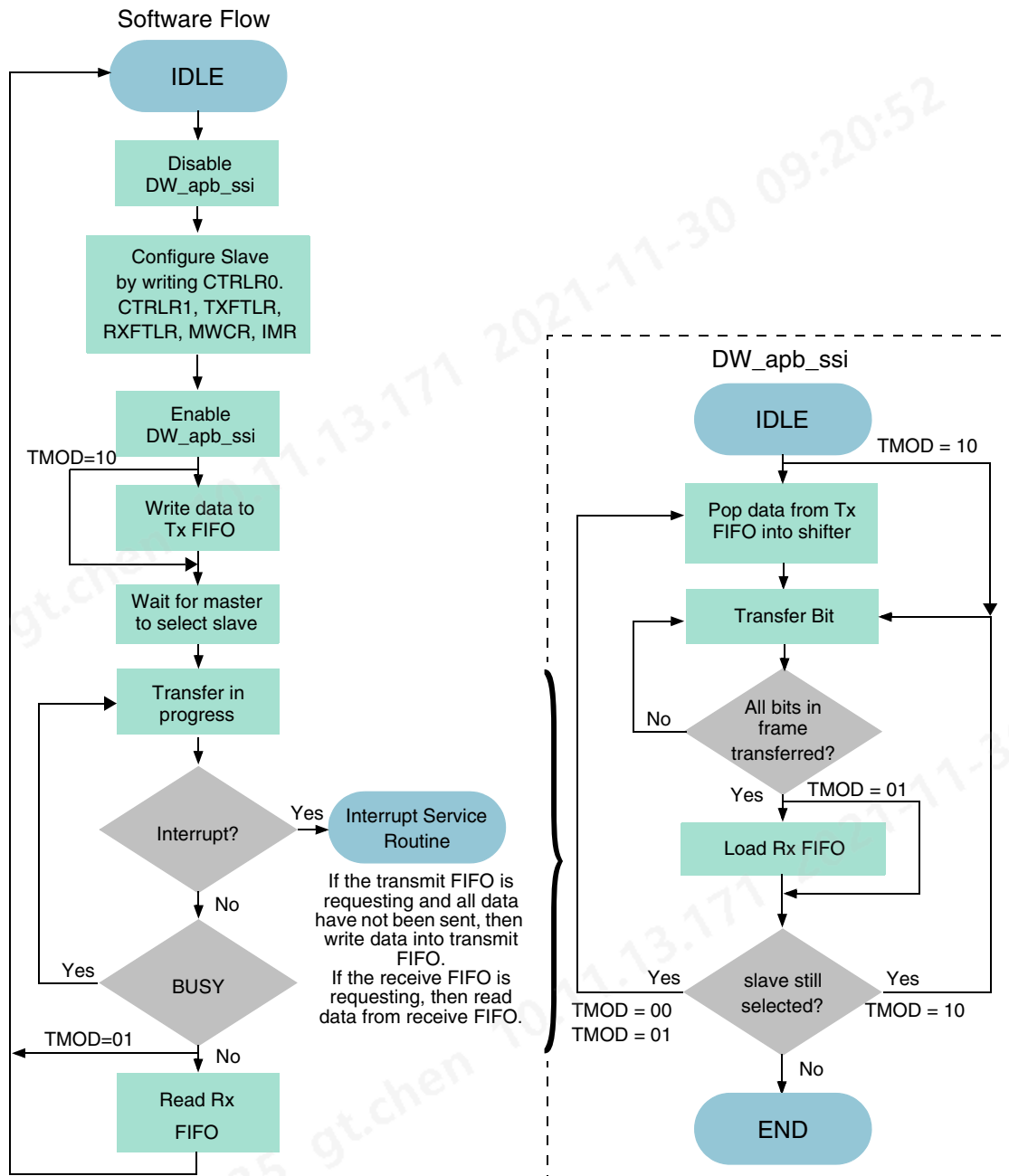
2.7.2.2   Slave Microwire Serial Transfers

"National Semiconductor Microwire" on page 54 describes the Microwire serial protocol in detail, including timing diagrams and information on how data are structured in the transmit and receive FIFOs before and after a serial transfer. When the DW_apb_ssi is configured as a slave device, the Microwire protocol

operates in much the same way as the SPI protocol. There is no decode of the control frame by the DW_apb_ssi slave device.

2.8 Partner Connection Interfaces

The DW_apb_ssi can connect to any serial-master or serial-slave peripheral device using one of the interfaces discussed in the following sections.

2.8.1 Motorola Serial Peripheral Interface (SPI)

With the SPI, the clock polarity (SCPOL) configuration parameter determines whether the inactive state of the serial clock is high or low. To transmit data, both SPI peripherals must have identical serial clock phase (SCPH) and clock polarity (SCPOL) values. The data frame can be 4 to 16/32-bits (depending upon SSI_MAX_XFER_SIZE) in length.

When the configuration parameter SCPH = 0, data transmission begins on the falling edge of the slave select signal. The first data bit is captured by the master and slave peripherals on the first edge of the serial clock; therefore, valid data must be present on the txd and rxd lines prior to the first serial clock edge.

Figure 2-17 shows a timing diagram for a single SPI data transfer with SCPH = 0. The serial clock is shown for configuration parameters SCPOL = 0 and SCPOL = 1.

Figure 2-17   SPI Serial Format (SCPH = 0)
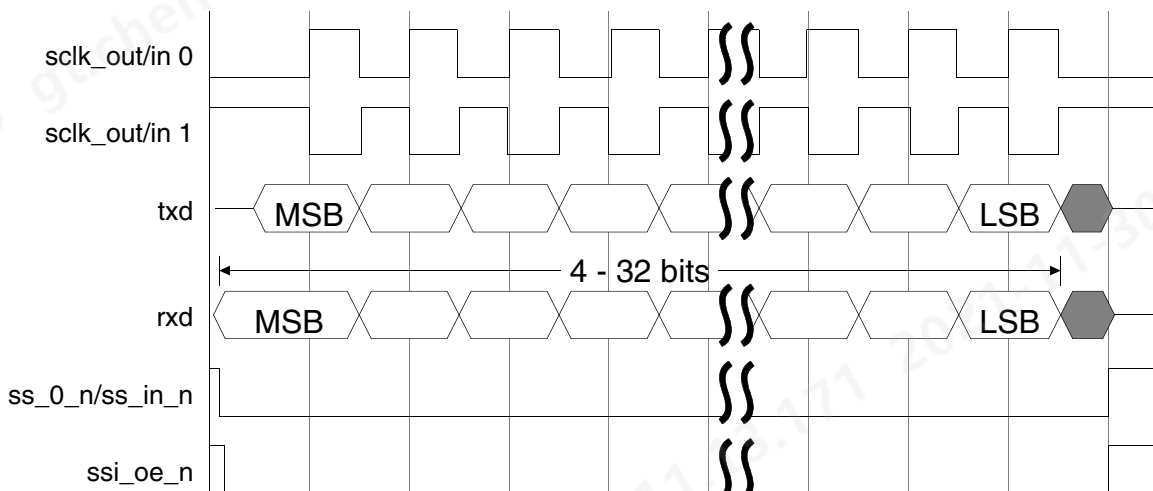
The following signals are illustrated in the timing diagrams in this section:

■ sclk_out – serial clock from DW_apb_ssi master (master configuration only)

■ sclk_in – serial clock from DW_apb_ssi slave (slave configuration only)

■ ss_0_n – slave select signal from DW_apb_ssi master (master configuration only)

■ ss_in_n – slave select input to the DW_apb_ssi slave

■ ss_oe_n – output enable for the DW_apb_ssi master/slave

■ txd – transmit data line for the DW_apb_ssi master/slave

■ rxd – receive data line for the DW_apb_ssi master/slave

Two different modes of continuous data transfers are supported when SCPH = 0; the selection of the desired operation mode is done using the configuration of the SSI_SCPH0_SSTOGGLE parameter:

■ When SSI_SCPH0_SSTOGGLE is configured as True and CTRLR0. SSTE is set to 1, the DW_apb_ssi toggles the slave select signal between frames and the serial clock is held to its default value while the slave select signal is active; this operating mode is illustrated in Figure 2-18.

■ When SSI_SCPH0_SSTOGGLE is configured as True and CTRLR0. SSTE is set to 0 or when SSI_SCPH0_SSTOGGLE is configured as False, the slave select signal stays low and the serial clock runs continuously for the duration of the transfer; this operating mode is illustrated in Figure 2-19.

Figure 2-18   Serial Format Continuous Transfers (SCPH = 0) when SSI_SCPH0_SSTOGGLE = 1
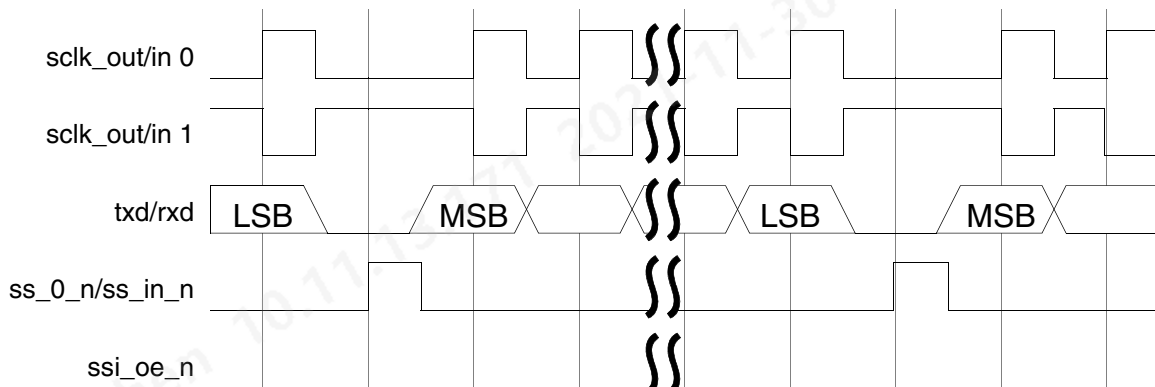
Figure 2-19   Serial Format Continuous Transfers (SCPH=0) when SSI_SCPH0_SSTOGGLE = 0
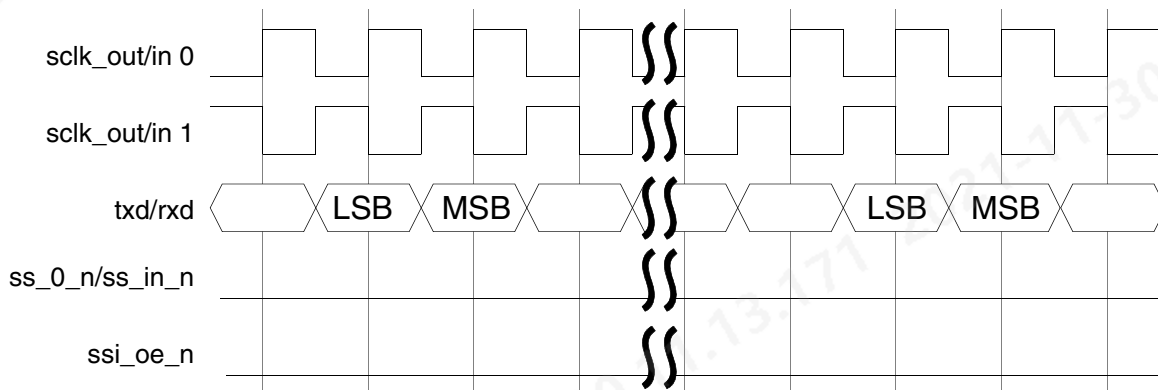
When the configuration parameter SCPH = 1, both master and slave peripherals begin transmitting data on the first serial clock edge after the slave select line is activated. The first data bit is captured on the second (trailing) serial clock edge. Data are propagated by the master and slave peripherals on the leading edge of the serial clock. During continuous data frame transfers, the slave select line may be held active-low until the last bit of the last frame has been captured.

Figure 2-19 shows the timing diagram for the SPI format when the configuration parameter SCPH = 1.

Figure 2-20   SPI Serial Format (SCPH = 1)
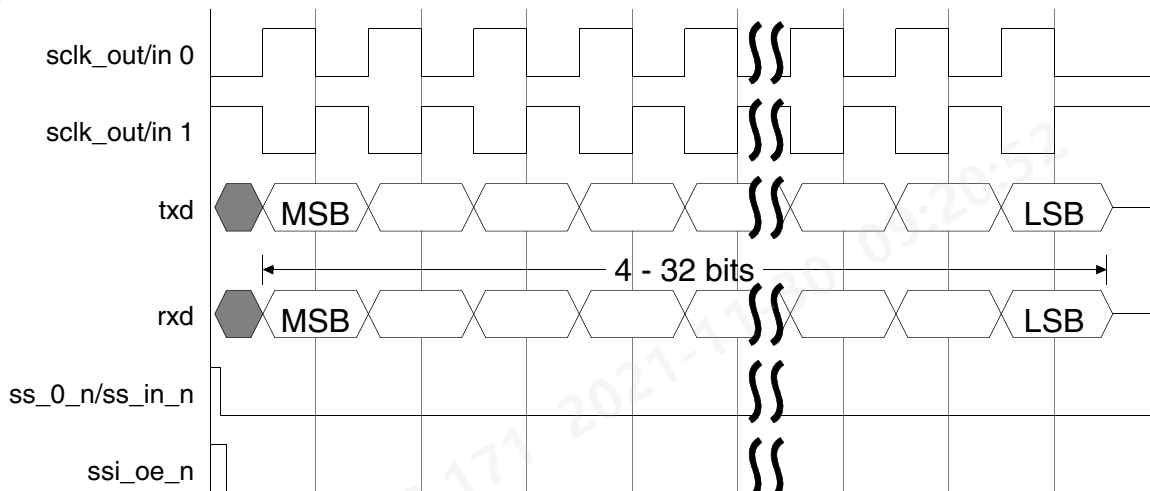
Continuous data frames are transferred in the same way as single frames, with the MSB of the next frame following directly after the LSB of the current frame. The slave select signal is held active for the duration of the transfer.

Figure 2-21 shows the timing diagram for continuous SPI transfers when the configuration parameter SCPH = 1.

Figure 2-21   SPI Serial Format Continuous Transfer (SCPH = 1)
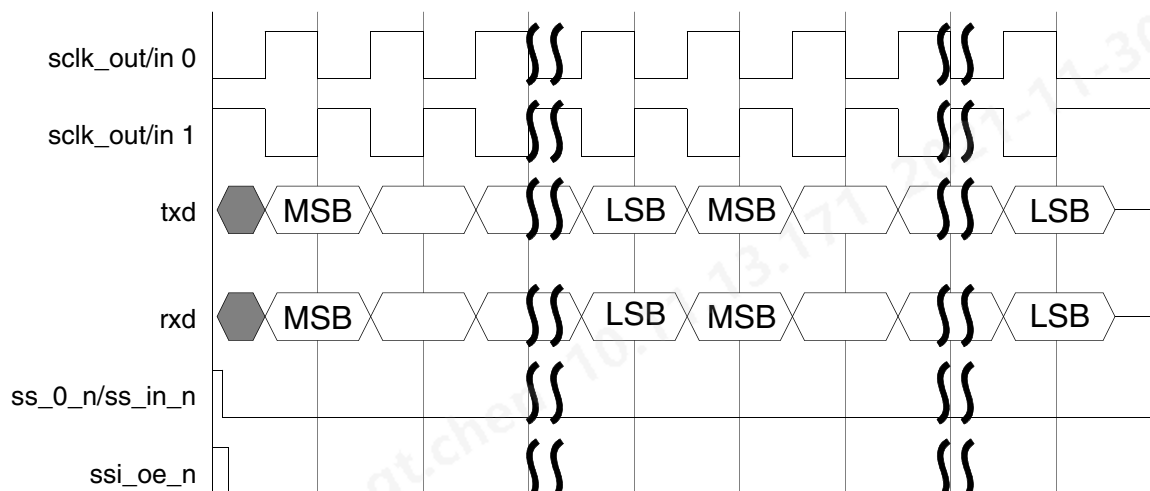
There are four possible transfer modes on the DW_apb_ssi for performing SPI serial transactions; see "Transfer Modes" on page 32. For *transmit and receive transfers* (transfer mode field (9:8) of the Control Register 0 = 2'b00), data transmitted from the DW_apb_ssi to the external serial device is written into the transmit FIFO. Data received from the external serial device into the DW_apb_ssi is pushed into the receive FIFO.

Figure 2-22 shows the FIFO levels prior to the beginning of a serial transfer and the FIFO levels on completion of the transfer. In this example, two data words are transmitted from the DW_apb_ssi to the external serial device in a continuous transfer. The external serial device also responds with two data words for the DW_apb_ssi.

Figure 2-22   FIFO Status for *Transmit & Receive* SPI and SSP Transfers
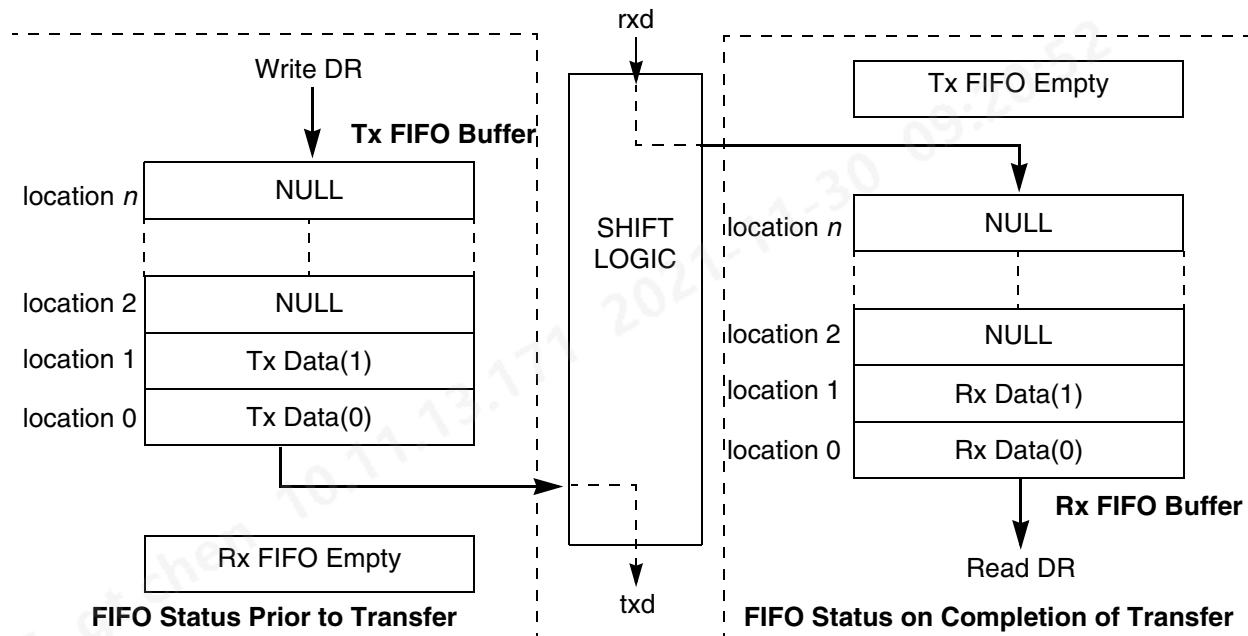
For *transmit only* transfers (transfer mode field (9:8) of the Control Register 0 = 2'b01), data transmitted from the DW_apb_ssi to the external serial device is written into the transmit FIFO. As the data received from the external serial device is deemed invalid, it is not stored in the DW_apb_ssi receive FIFO.

Figure 2-23 shows the FIFO levels prior to the beginning of a serial transfer and the FIFO levels on completion of the transfer. In this example, two data words are transmitted from the DW_apb_ssi to the external serial device in a continuous transfer.

Figure 2-23   FIFO Status for *Transmit Only* SPI and SSP Transfers
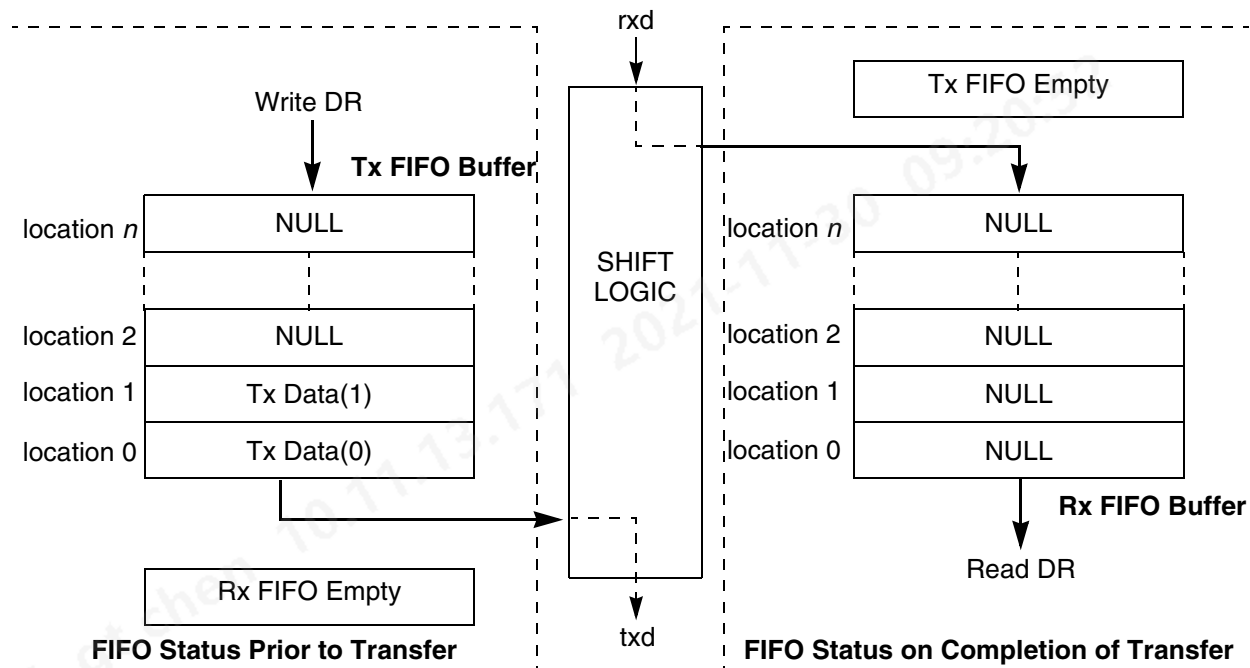
For *receive only* transfers (transfer mode field (9:8) of the Control Register 0 = 2'b10), data transmitted from the DW_apb_ssi to the external serial device is invalid, so a single dummy word is written into the transmit FIFO to begin the serial transfer. The txd output from the DW_apb_ssi is held at a constant logic level for the duration of the serial transfer. Data received from the external serial device into the DW_apb_ssi is pushed into the receive FIFO.

Figure 2-24 shows the FIFO levels prior to the beginning of a serial transfer and the FIFO levels on completion of the transfer. In this example, two data words are received by the DW_apb_ssi from the external serial device in a continuous transfer.

Figure 2-24   FIFO Status for *Receive Only* SPI and SSP Transfers
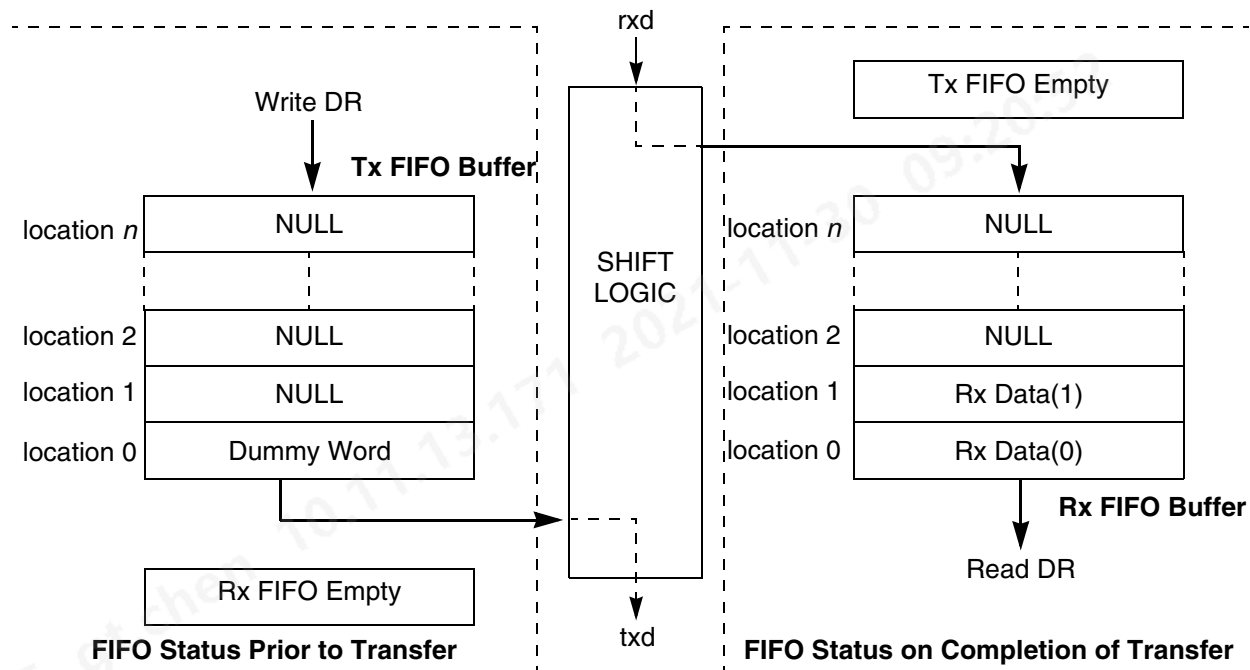
For eeprom_read transfers (transfer mode field [9:8] of the Control Register 0 = 2'b11), opcode and/or EEPROM address are written into the transmit FIFO. During transmission of these control frames, received data is not captured by the DW_apb_ssi master. After the control frames have been transmitted, receive data from the EEPROM is stored in the receive FIFO.

Figure 2-25 shows the FIFO levels prior to the beginning of a serial transfer and the FIFO levels on completion of the transfer. In this example, one opcode and an upper and lower address are transmitted to

the EEPROM, and eight data frames are read from the EEPROM and stored in the receive FIFO of the
DW_apb_ssi master.

Figure 2-25   FIFO Status for *EEPROM Read* Transfer Mode
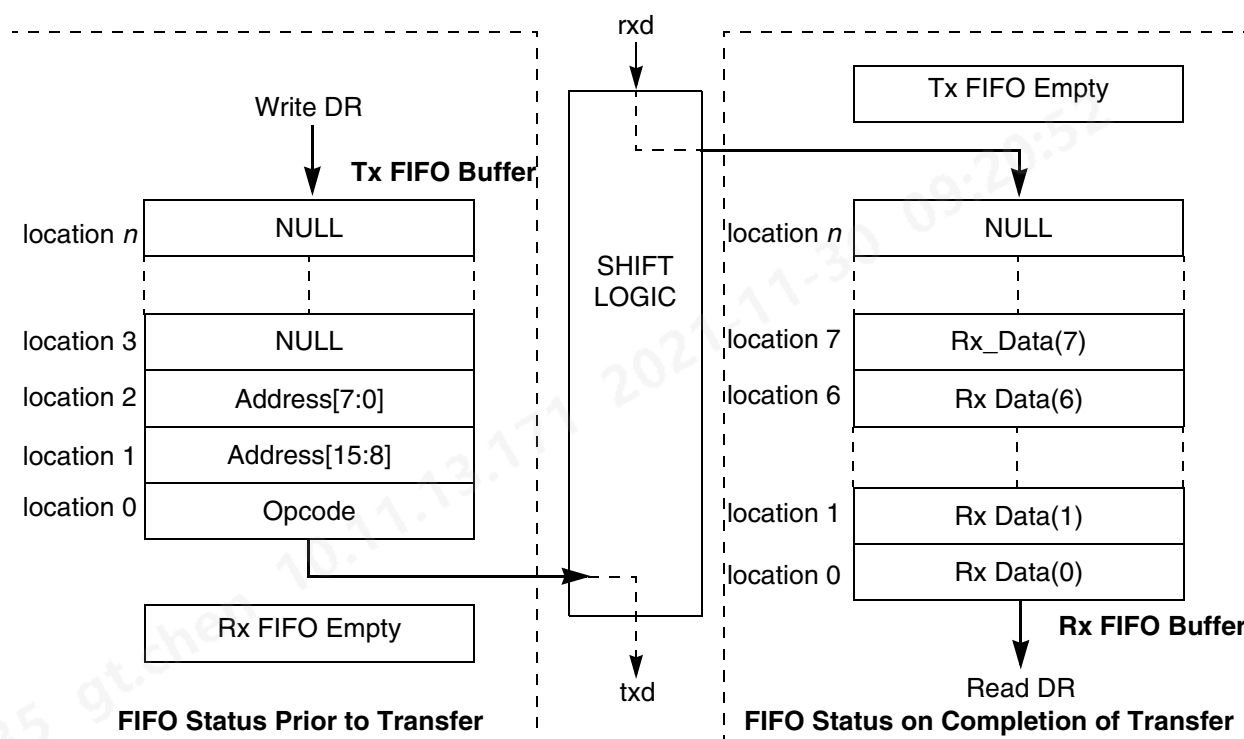
2.8.2   Texas Instruments Synchronous Serial Protocol (SSP)

Data transfers begin by asserting the frame indicator line (ss_0_n/ss_in_n) for one serial clock period. Data
to be transmitted are driven onto the txd line one serial clock cycle later; similarly data from the slave are
driven onto the rxd line. Data are propagated on the rising edge of the serial clock (sclk_out/sclk_in) and
captured on the falling edge. The length of the data frame ranges from 4 to 16/32-bits (depending upon
SSI_MAX_XFER_SIZE).

Figure 2-26 shows the timing diagram for a single SSP serial transfer.

Figure 2-26   SSP Serial Format
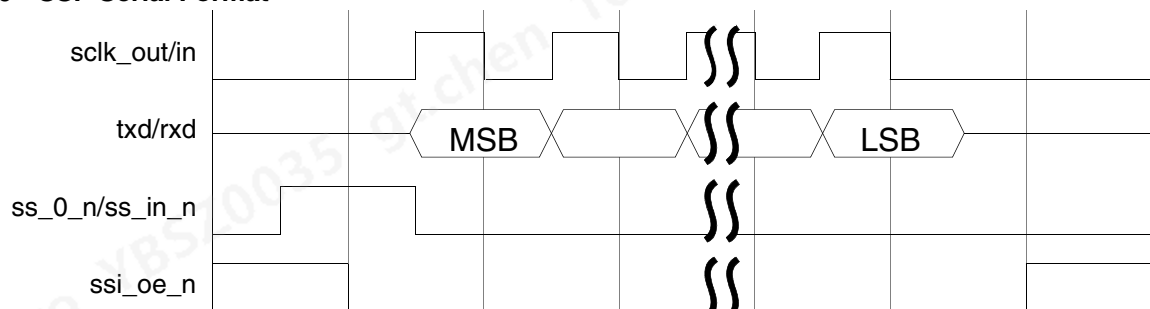
Continuous data frames are transferred in the same way as single data frames. The frame indicator is asserted for one clock period during the same cycle as the LSB from the current transfer, indicating that another data frame follows.

Figure 2-27 shows the timing for a continuous SSP transfer.

Figure 2-27   SSP Serial Format Continuous Transfer
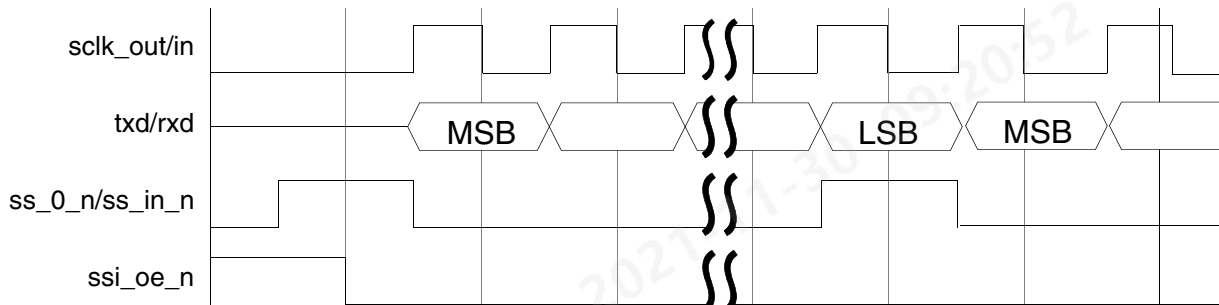
2.8.3   National Semiconductor Microwire

When the DW_apb_ssi is configured as a serial master, data transmission begins with the falling edge of the slave-select signal (ss_0_n). One-half serial clock (sclk_out) period later, the first bit of the control is sent out on the txd line. The length of the control word can be in the range 1 to 16 bits and is set by writing bit field CFS (bits 15:12) in CTRLR0. The remainder of the control word is transmitted (propagated on the falling edge of sclk_out) by the DW_apb_ssi serial master. During this transmission, no data are present (high impedance) on the serial master's rxd line.

The direction of the data word is controlled by the MDD bit field (bit 1) in the Microwire Control Register (MWCR). When MDD=0, this indicates that the DW_apb_ssi serial master receives data from the external serial slave. One clock cycle after the LSB of the control word is transmitted, the slave peripheral responds with a dummy 0 bit, followed by the data frame, which can be 4 to 16/32-bits (depending upon SSI_MAX_XFER_SIZE) in length. Data are propagated on the falling edge of the serial clock and captured on the rising edge.

The slave-select signal is held active-low during the transfer and is de-asserted one-half clock cycle later, after the data are transferred. Figure 2-28 shows the timing diagram for a single DW_apb_ssi serial master read from an external serial slave.

Figure 2-28   Single DW_apb_ssi Master Microwire Serial Transfer (MDD=0)
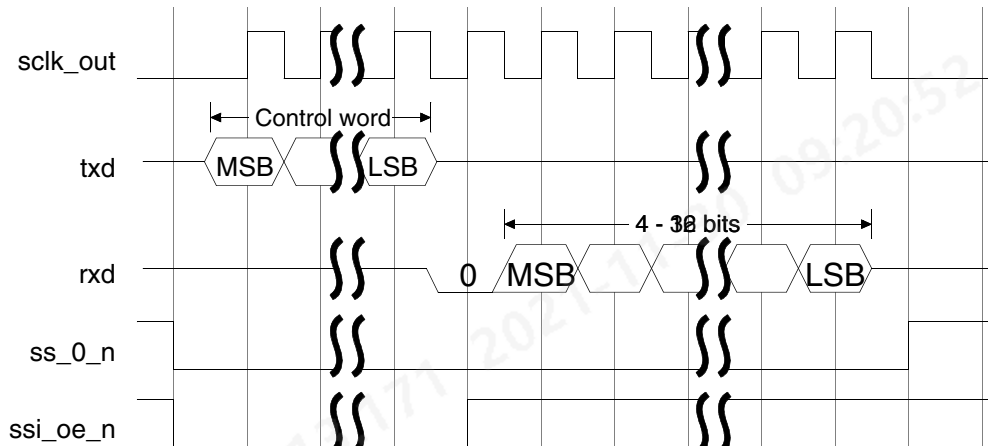
Figure 2-29 shows how the data and control frames are structured in the transmit FIFO prior to the transfer; the value programmed into the MWCR register is also shown.

Figure 2-29   FIFO Status for Single Microwire Transfer (Receiving Data Frame)
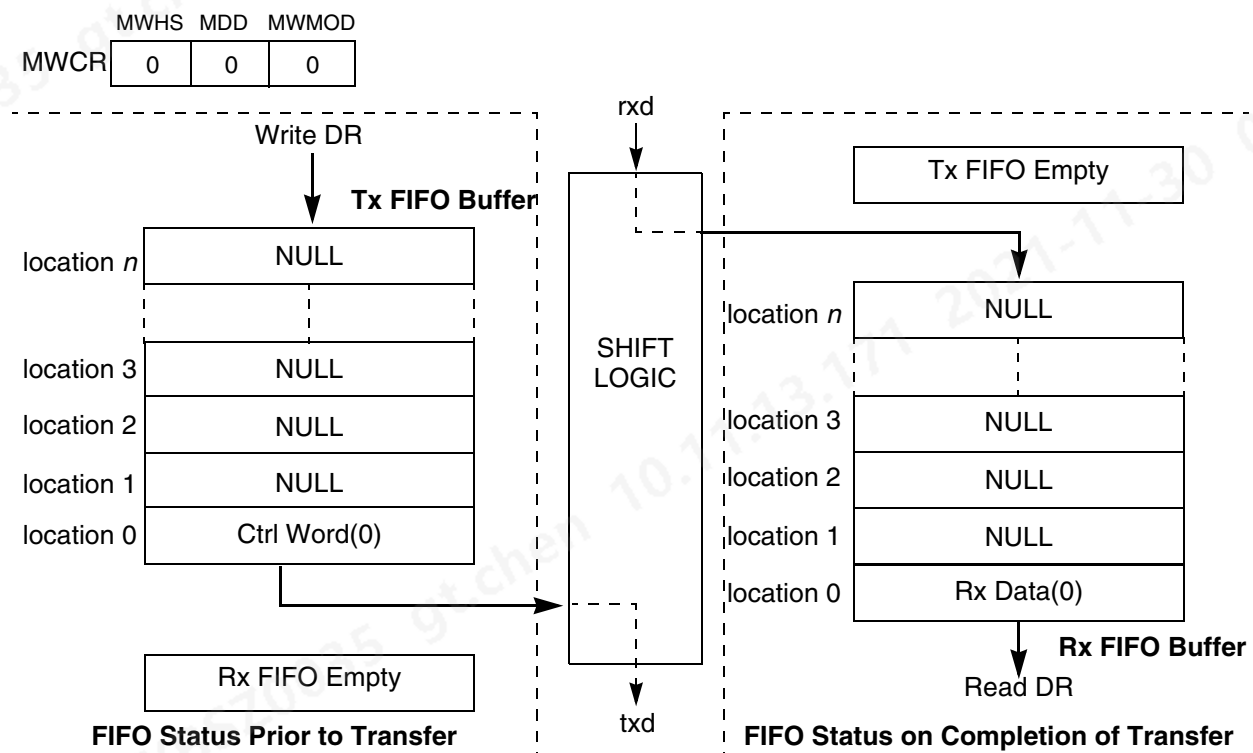
Continuous transfers from the Microwire protocol can be sequential or nonsequential, and are controlled by the MWMOD bit field (bit 0) in the MWCR register.

Non-sequential continuous transfers occur as illustrated in Figure 2-30, with the control word for the next transfer following immediately after the LSB of the current data word.

Figure 2-30   Continuous Non-sequential Microwire Transfer (Receiving Data Frame)
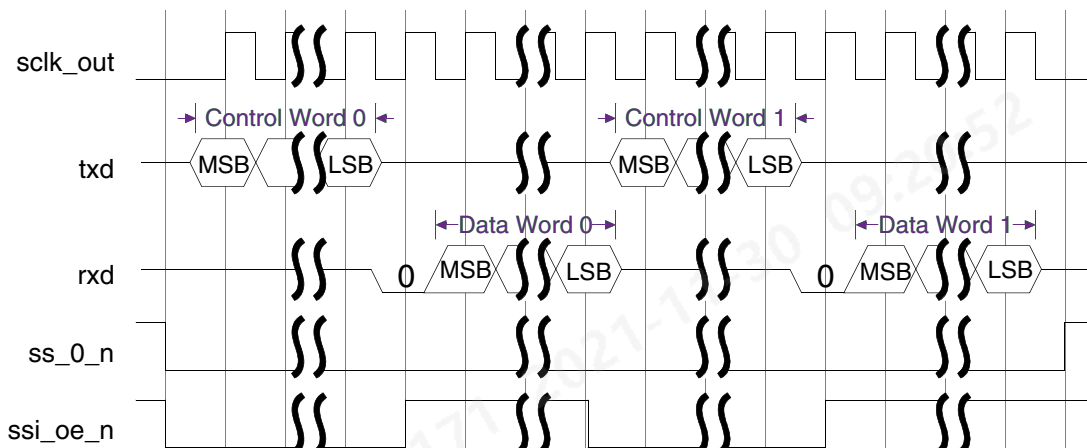
The only modification needed to perform a continuous non-sequential transfer is to write more control words into the transmit FIFO buffer; this is illustrated in Figure 2-31. In this example, two data words are read from the external serial-slave device.

Figure 2-31   FIFO Status for Non-sequential Microwire Transfer (Receiving Data Frame)
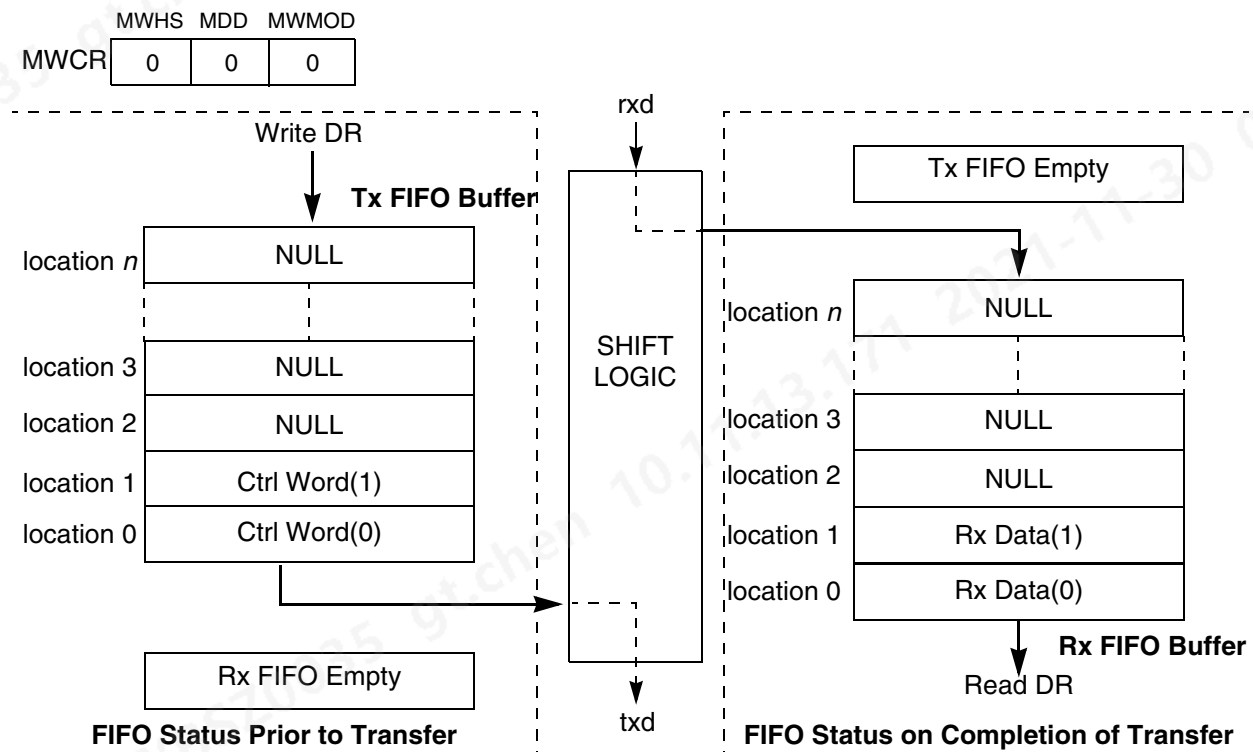
During sequential continuous transfers, only one control word is transmitted from the DW_apb_ssi master. The transfer is started in the same manner as with non-sequential read operations, but the cycle is continued

to read further data. The slave device automatically increments its address pointer to the next location and continues to provide data from that location. Any number of locations can be read in this manner; the DW_apb_ssi master terminates the transfer when the number of words received is equal to the value in the CTRLR1 register plus 1.

The timing diagram in Figure 2-32 and example in Figure 2-33 show a continuous sequential read of two data frames from the external slave device.

Figure 2-32   Continuous Sequential Microwire Transfer (Receiving Data Frame)
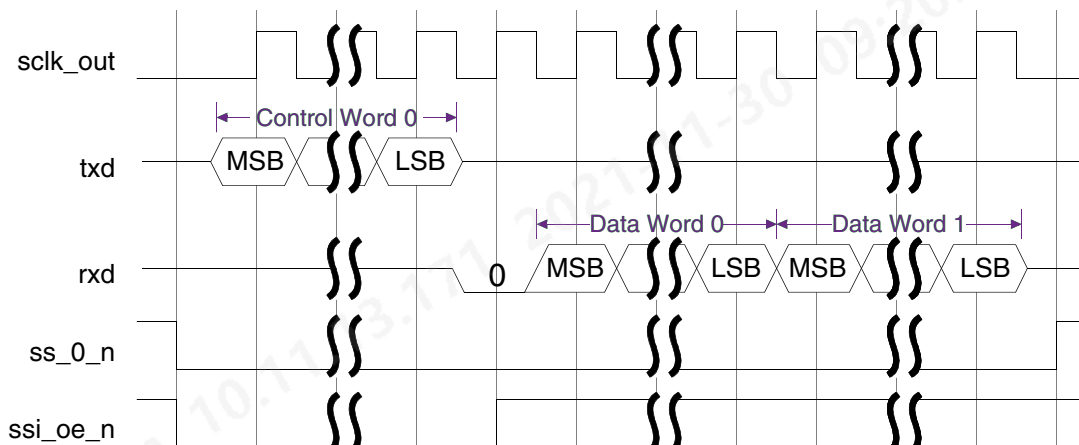
Figure 2-33   FIFO Status for Sequential Microwire Transfer (Receiving Data Frame)
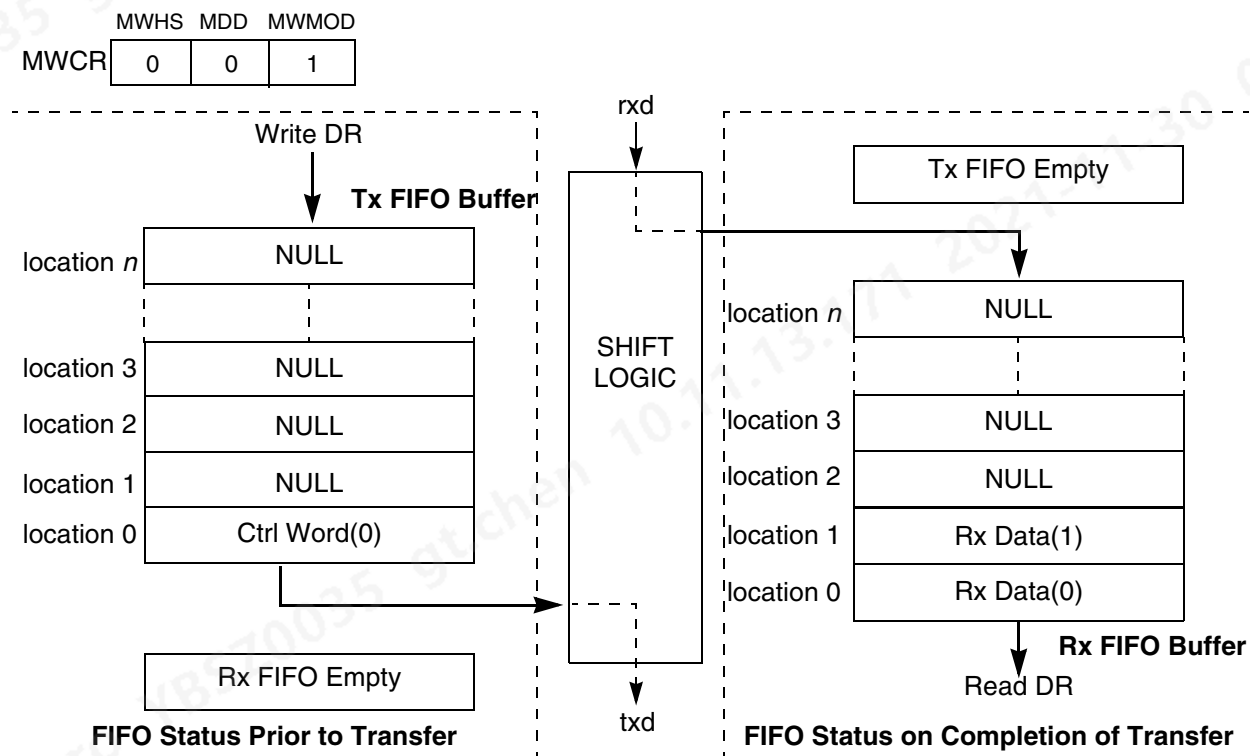
When MDD = 1, this indicates that the DW_apb_ssi serial master transmits data to the external serial slave. Immediately after the LSB of the control word is transmitted, the DW_apb_ssi master begins transmitting the data frame to the slave peripheral.

Figure 2-34 shows the timing diagram for a single DW_apb_ssi serial master write to an external serial slave.

Figure 2-34   Single Microwire Transfer (Transmitting Data Frame)
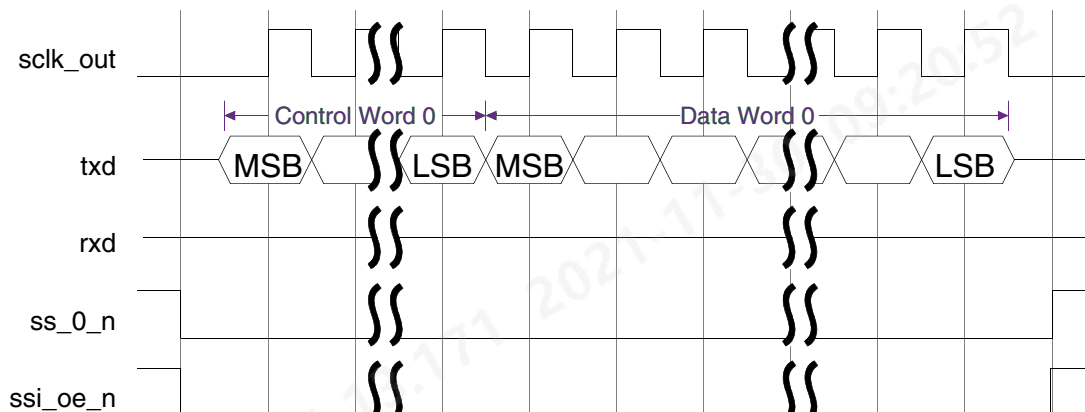
> **Note**   The DW_apb_ssi does not support continuous sequential Microwire writes, where MDD = 1 and MWMOD = 1.

Figure 2-35 shows how the data and control frames are structured in the transmit FIFO prior to the transfer, also shown is the value programmed into the MWCR register.

Figure 2-35   FIFO Status for Single Microwire Transfer (Transmitting Data Frame)
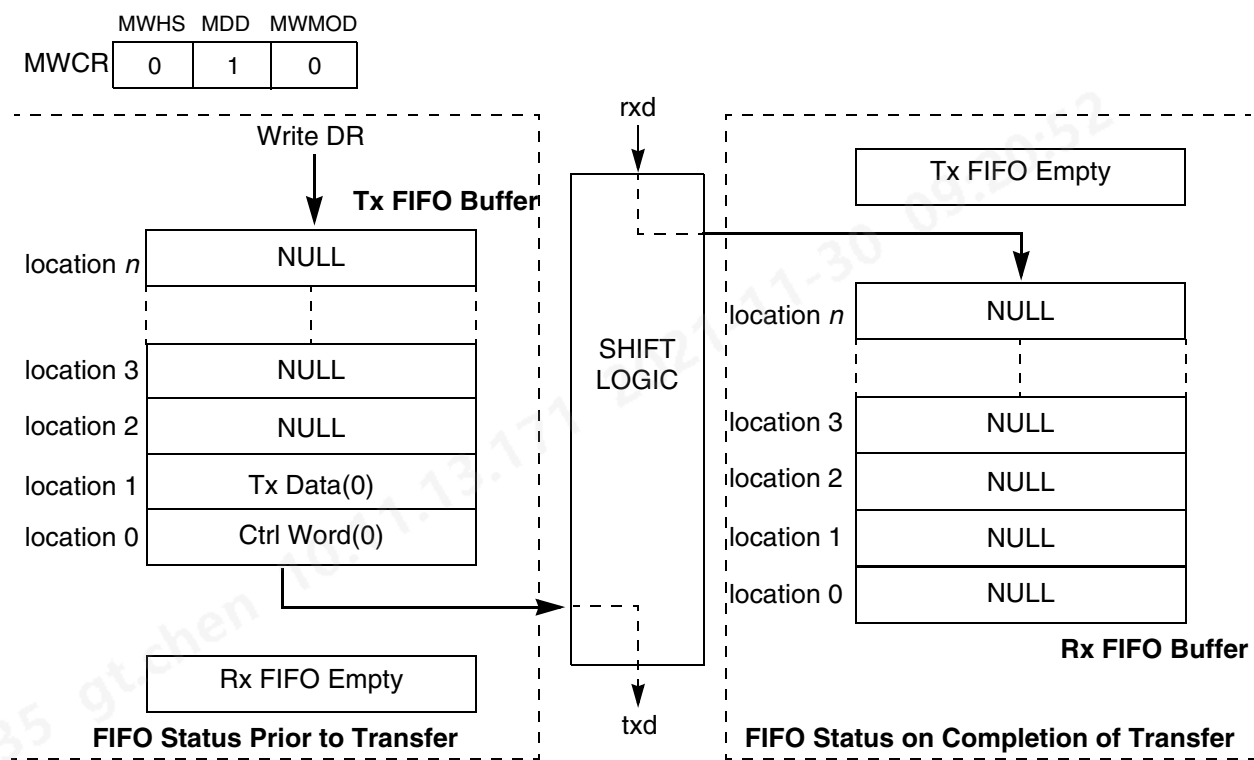
Continuous transfers occur as shown in Figure 2-36, with the control word for the next transfer following immediately after the LSB of the current data word.

Figure 2-36   Continuous Microwire Transfer (Transmitting Data Frame)
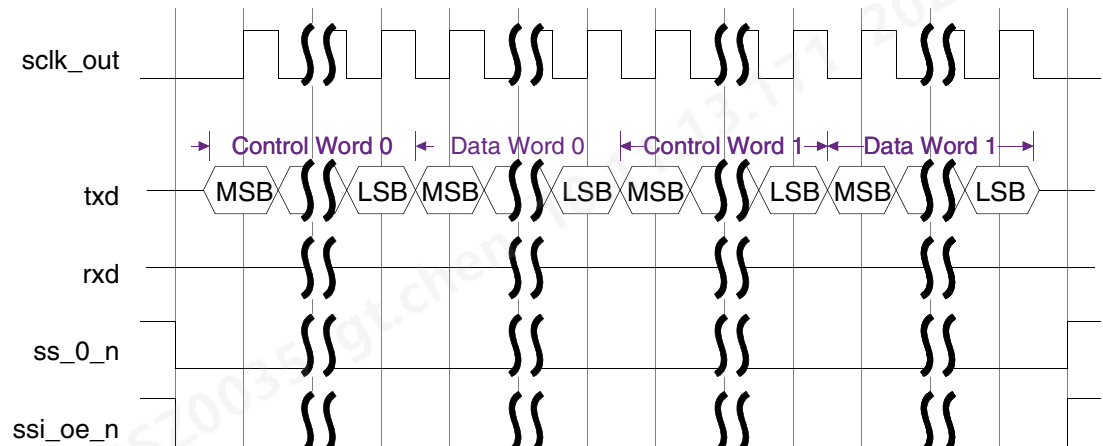
The only modification you need to make to perform a continuous transfer is to write more control and data words into the transmit FIFO buffer, shown in Figure 2-37. This example shows two data words are written to the external serial slave device.

Figure 2-37   FIFO Status for Continuous Microwire Transfer (Transmitting Data Frame)
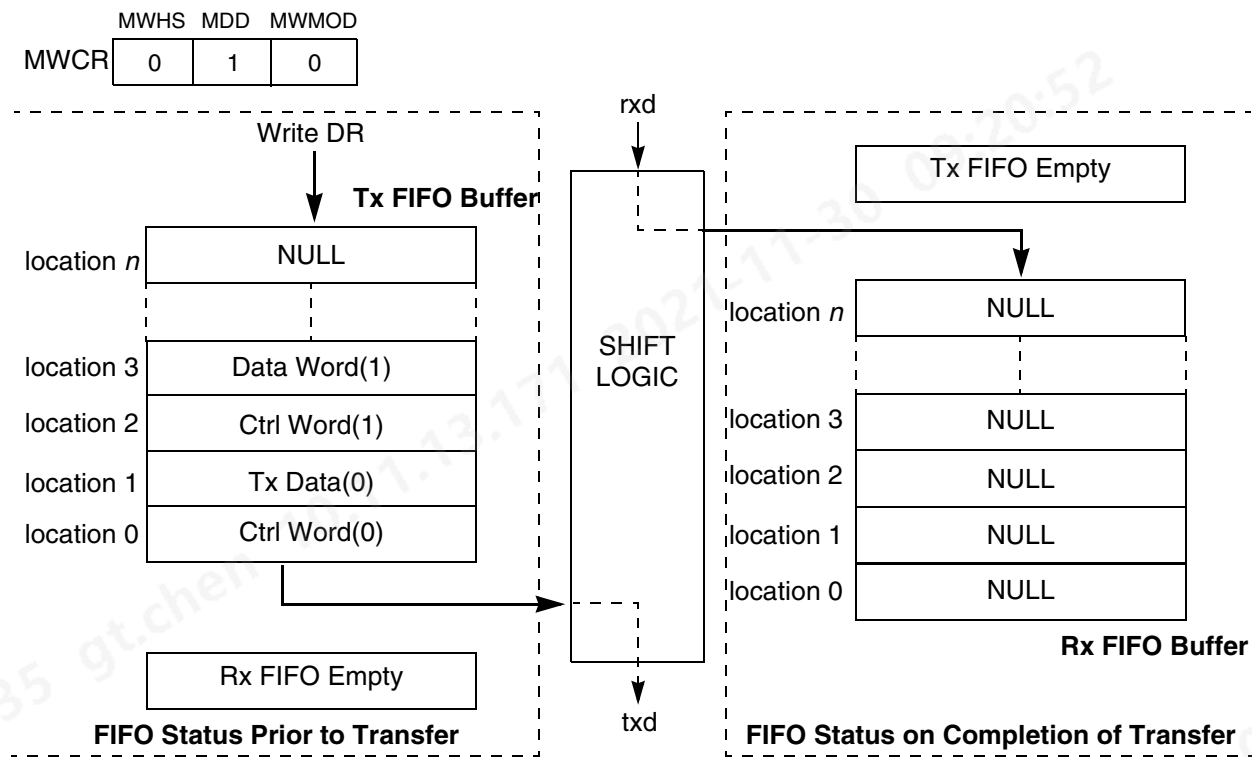
The Microwire handshaking interface can also be enabled for DW_apb_ssi master write operations to external serial-slave devices. To enable the handshaking interface, you must write 1 into the MHS bit field (bit 2) on the MWCR register. When MHS is set to 1, the DW_apb_ssi serial master checks for a ready status from the slave device before completing the transfer, or transmitting the next control word for continuous transfers.

Figure 2-38 shows an example of a continuous Microwire transfer with the handshaking interface enabled.

Figure 2-38   Continuous Microwire Transfer with Handshaking (Transmitting Data Frame)
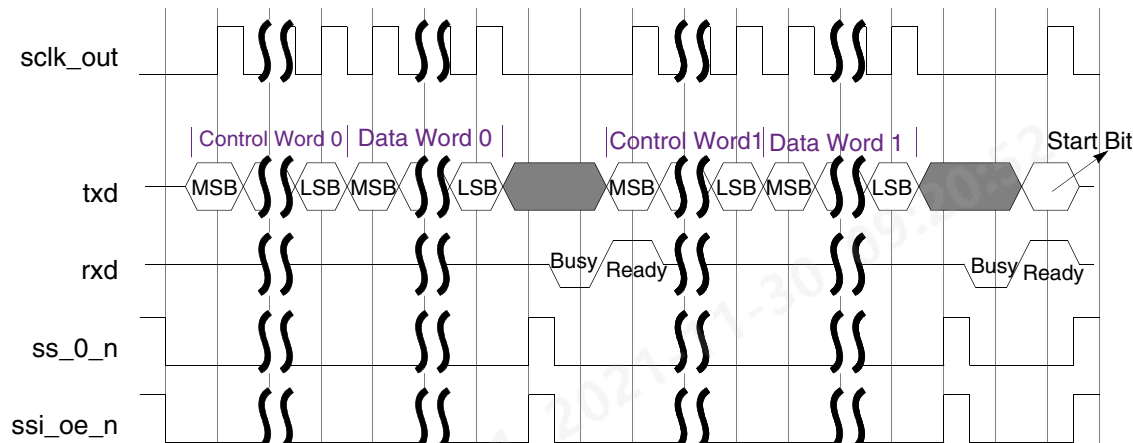
After the first data word has been transmitted to the serial-slave device, the DW_apb_ssi master polls the rxd input waiting for a ready status from the slave device. Upon reception of the ready status, the DW_apb_ssi master begins transmission of the next control word. It is assumed that slave will keep the READY status until it receives any confirmation from the Master, which is provided when the clock is resumed by the Master. After transmission of the last data frame has completed, the DW_apb_ssi master transmits a start bit to clear the ready status of the slave device before completing the transfer. The FIFO status for this transfer is the same as in Figure 2-37, except that the MWHS bit field is set (1).

To transmit a control word (not followed by data) to a serial-slave device from the DW_apb_ssi master, there must be only one entry in the transmit FIFO buffer. It is impossible to transmit two control words in a continuous transfer, as the shift logic in the DW_apb_ssi treats the second control word as a data word. When the DW_apb_ssi master transmits only a control word, the MDD bit field (bit 1 of MWCR register) must be set (1).

In the example shown in Figure 2-39 and in the timing diagram in Figure 2-40, the handshaking interface is enabled. If the handshaking interface is disabled (MHS=0), the transfer is terminated by the DW_apb_ssi master one sclk_out cycle after the LSB of the control word is captured by the slave device.

Figure 2-39 FIFO Status for Microwire Control Word Transfer
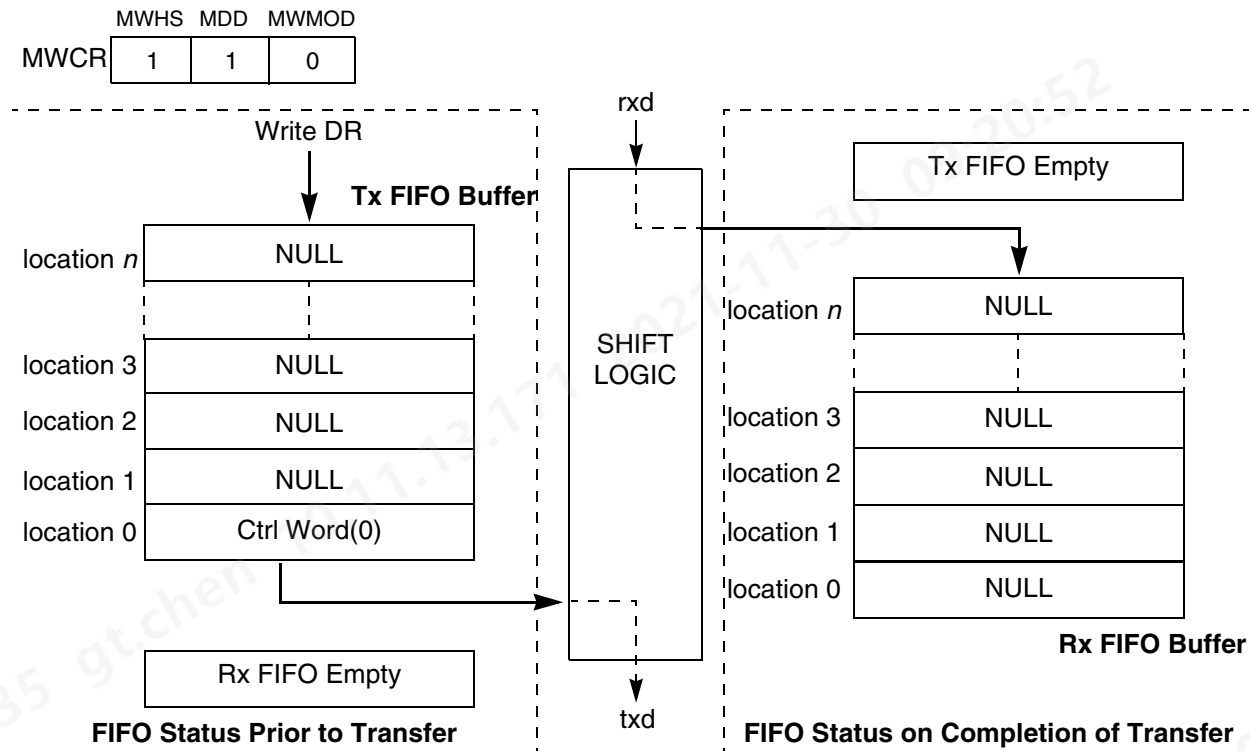
Figure 2-40 Microwire Control Word
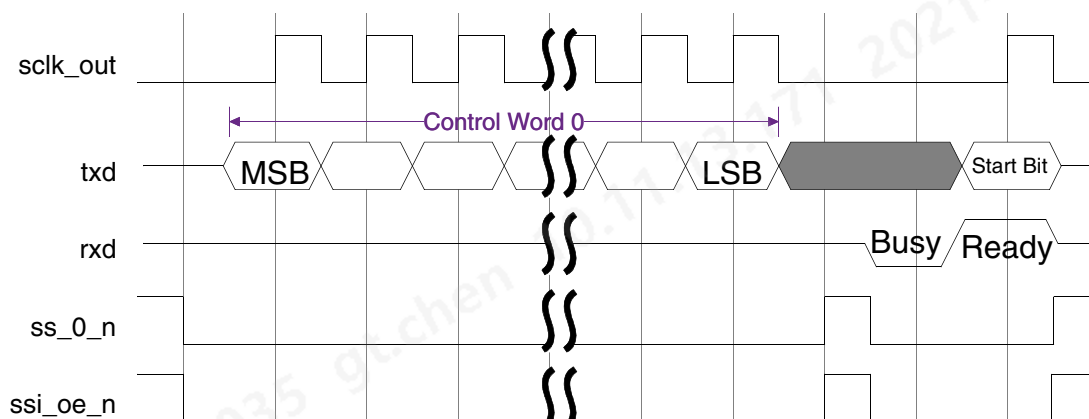
When the DW_apb_ssi is configured as a serial slave, data transmission begins with the falling edge of the slave select signal (ss_in_n). One-half serial clock (sclk_in) period later, the first bit of the control is present on the rxd line. The length of the control word can be in the range of 1 to 16 bits and is set by writing bit field CFS in the CTRLR0 register. The CFS bit field must be set to the size of the expected control word from the

serial master. The remainder of the control word is received (captured on the rising edge of sclk_in) by the DW_apb_ssi serial slave. During this reception, no data are driven (high impedance) on the serial slave's txd line.

The direction of the data word is controlled by the MDD bit field (bit 1) MWCR register. When MDD=0, this indicates that the DW_apb_ssi serial slave is to receive data from the external serial master. Immediately after the control word is transmitted, the serial master begins to drives the data frame onto the DW_apb_ssi slave rxd line. Data are propagated on the falling edge of the serial clock and captured on the rising edge. The slave-select signal is held active-low during the transfer and is de-asserted one-half clock cycle later after the data are transferred. The DW_apb_ssi slave output enable signal (ssi_oe_n) is held inactive for the duration of the transfer.

Figure 2-41 shows the timing diagram for single DW_apb_ssi serial slave read from an external serial master.

**Figure 2-41   Single DW_apb_ssi Slave Microwire Serial Transfer (MDD=0)**
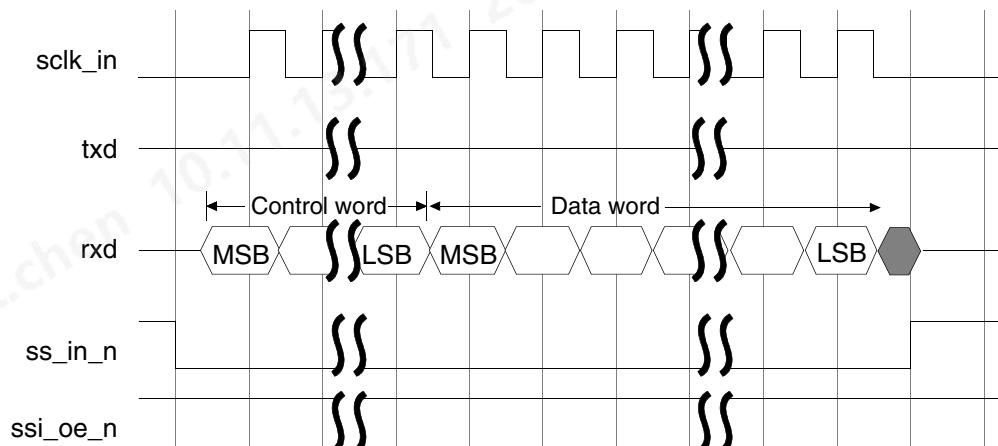
Figure 2-42 shows how the data and control frames are stored in the receive FIFO on completion of the transfer; also shown is the value programmed into the MWCR register.

Figure 2-42   FIFO Status for Single Microwire Transfer (Receiving Data Frame)

When MDD=1, this indicates that the DW_apb_ssi serial slave transmits data to the external serial master. Immediately after the LSB of the control word is transmitted, the DW_apb_ssi slave transmits a dummy 0 bit, followed by the 4- to 16/32-bit (depending upon SSI_MAX_XFER_SIZE) data frame on the txd line.

Figure 2-43 shows the timing diagram for a single DW_apb_ssi serial slave write to an external serial master.

Figure 2-43   Single DW_apb_ssi Slave Microwire Serial Transfer (MDD=1)
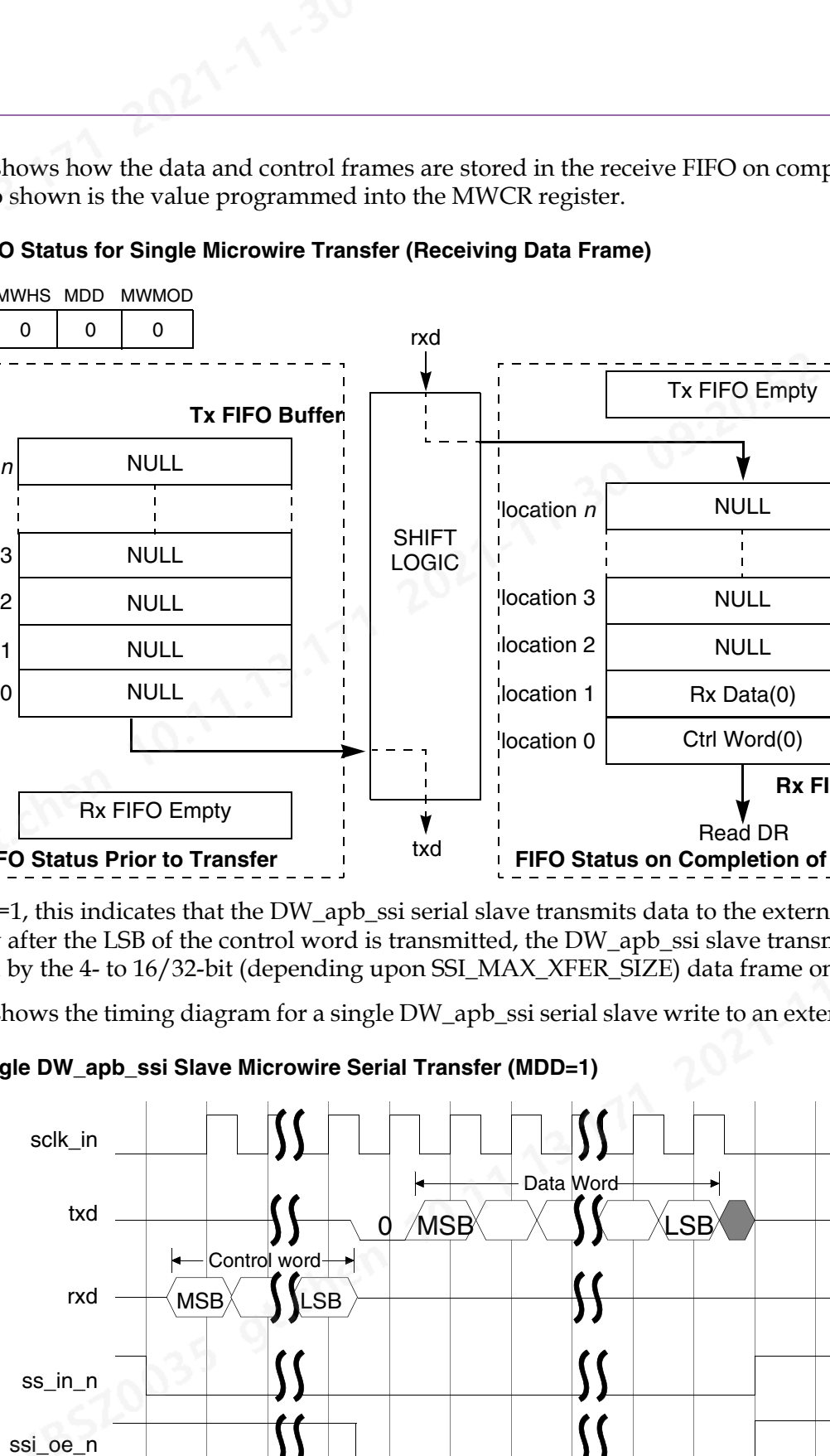
Figure 2-44   FIFO Status for Single Microwire Transfer (Transmitting Data Frame)
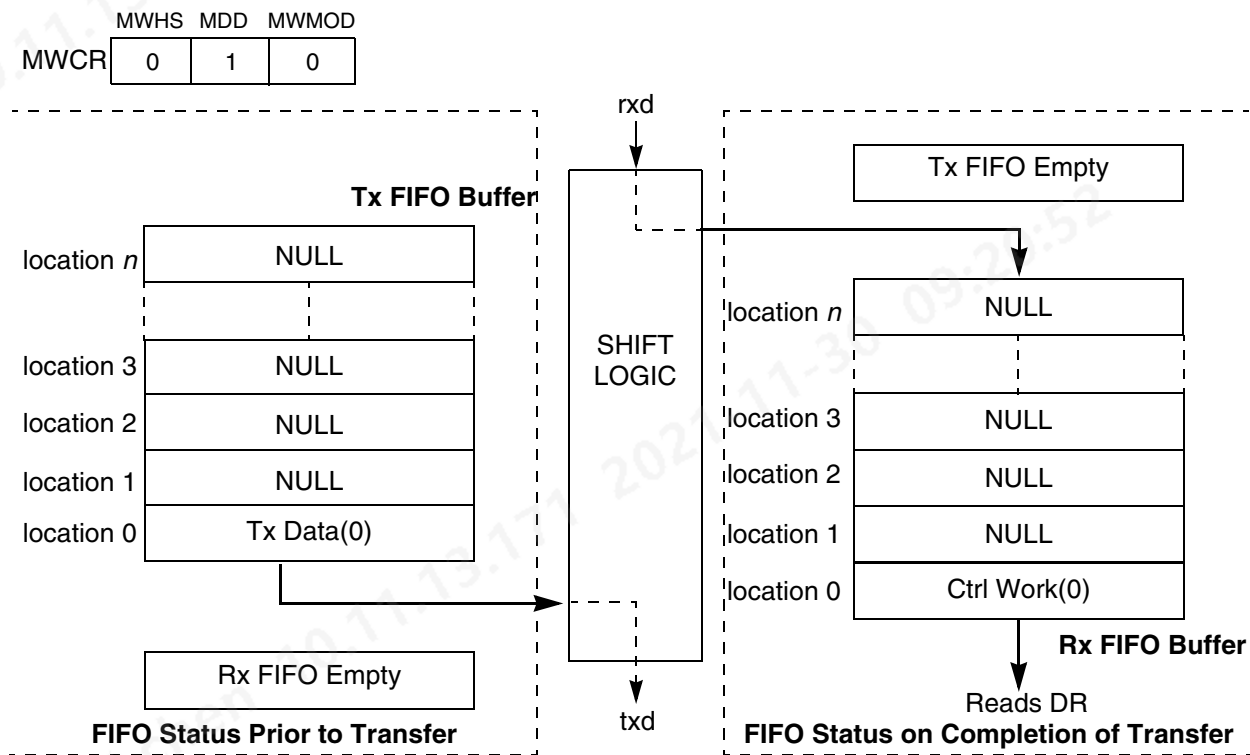
Continuous transfers for a DW_apb_ssi slave occur in the same way as those specified for the DW_apb_ssi master configuration. The DW_apb_ssi slave configuration does not support the handshaking interface, as there is never a busy period.

2.8.4   Enhanced SPI Modes

DW_apb_ssi supports the dual, quad, and octal modes of SPI using the SSI_SPI_MODE configuration parameter. The possible values for this parameter are Standard, Dual SPI, Quad SPI and Octal SPI modes. When dual, quad, or octal mode is selected for this parameter, the width of txd, rxd and ssi_oe_n signals change to 2, 4, or 8, respectively. Hence, the data is shifted out/in on more than one line, increasing the overall throughput. All four combinations of the serial clock's polarity and phase are valid in this mode and it works same as in normal SPI mode as described in "Motorola Serial Peripheral Interface (SPI)" on page 47. Dual SPI, Quad or Octal SPI modes function similarly except for the width of txd, rxd and ssi_oe_n signals. The mode of operation (write/read) can be selected using the CTRLR0.TMOD field.

The following sections describe the read and write operations in Dual SPI and Quad SPI modes in detail:

- "Write Operation in Enhanced SPI Modes" on page 65
- "Read Operation in Enhanced SPI Modes" on page 68

2.8.4.1   Write Operation in Enhanced SPI Modes

Dual, Quad, or Octal SPI write operations can be divided into three parts:

- Instruction phase
- Address phase

■ Data phase

The following register fields are used for a write operation:

■ CTRLR0.SPI_FRF - Specifies the format in which the transmission happens for the frame.

■ SPI_CTRLR0 (Control Register 0 register) – Specifies length of instruction, address, and data.

■ SPI_CTRLR0.INST_L – Specifies length of an instruction (possible values for an instruction are 0, 4, 8, or 16 bits.)

■ SPI_CTRLR0.ADDR_L – Specifies address length (See Table 2-3 for decode values)

■ CTRLR0.DFS or CTRLR0.DFS_32 – Specifies data length.

An instruction takes one FIFO location and address can take more than one FIFO locations. Both the instruction and address must be programmed in the data register (DR). DW_apb_ssi waits until both have been programmed to start the write operation.

The instruction, address and data can be programmed to send in dual/quad mode, which can be selected from the SPI_CTRLR0.TRANS_TYPE and CTRLR0.SPI_FRF fields.

	☞ Note	■ If CTRLR0.SPI_FRF is selected to be "Standard SPI Format", everything is sent in Standard SPI mode and SPI_CTRLR0.TRANS_TYPE field is ignored.
		■ CTRLR0.SPI_FRF is only applicable if CTRLR0.FRF is programmed to 00.

Figure 2-45 shows a typical write operation in Dual, Quad, or Octal SPI Mode. The value of N is: 7 if SSI_SPI_MODE is set to 3, 3 if SSI_SPI_MODE is set to 2, and 1 if SSI_SPI_MODE is set to 1. For one write operation, the instruction and address are sent only once followed by data frames programmed in DR until the transmit FIFO becomes empty.

Figure 2-45   Typical Write Operation Dual/Quad/Octal SPI Mode
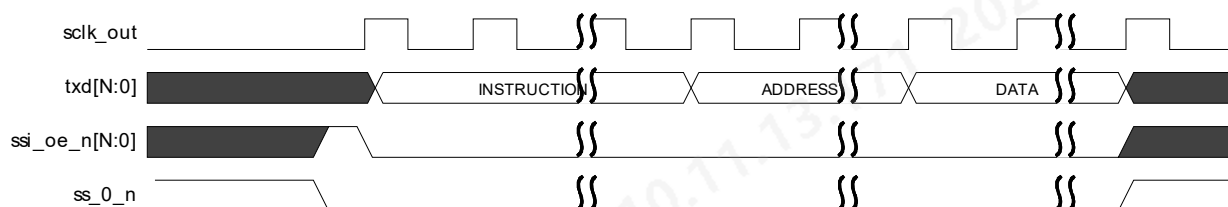
To initiate a Dual/Quad/Octal write operation, CTRLR0.SPI_FRF must be set to 01/10/11, respectively. This sets the transfer type, and for each write command, data is transferred in the format specified in CTLR0.SPI_FRF field.

Following are the possible cases of write operation in enhanced SPI modes

■ Case A: Instruction and address both transmitted in standard SPI format

For this, SPI_CTRLR0.TRANS_TYPE field must be set to 00. Figure 2-46 show the timing diagram when both instruction and address are transmitted in standard SPI format. The value of N is: 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01.

Figure 2-46   Instruction and Address Transmitted in Standard SPI Format
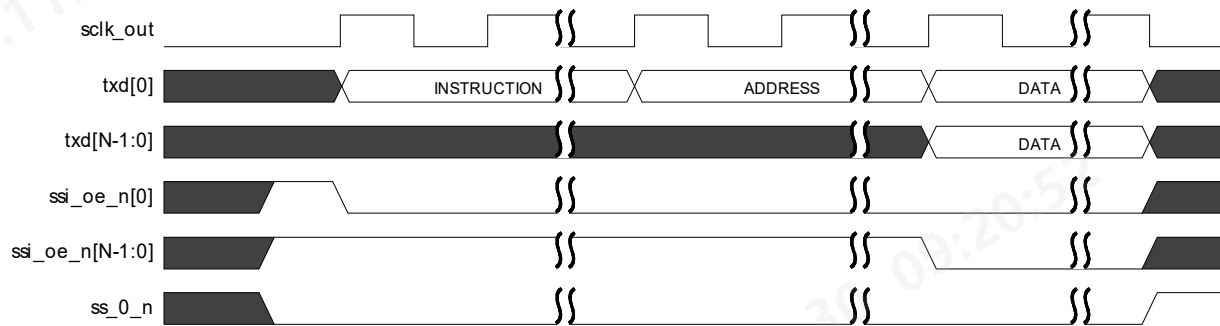
- **Case B: Instruction transmitted in standard and address transmitted in Enhanced SPI format**

    For this, SPI_CTRLR0.TRANS_TYPE field must be set to 01. Figure 2-47 shows the timing diagram when an instruction is transmitted in standard format and address is transmitted in dual SPI format specified in the CTRLR0.SPI_FRF field. The value of N is: 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01.

Figure 2-47   Instruction Transmitted in Standard and Address Transmitted in Enhanced SPI Format
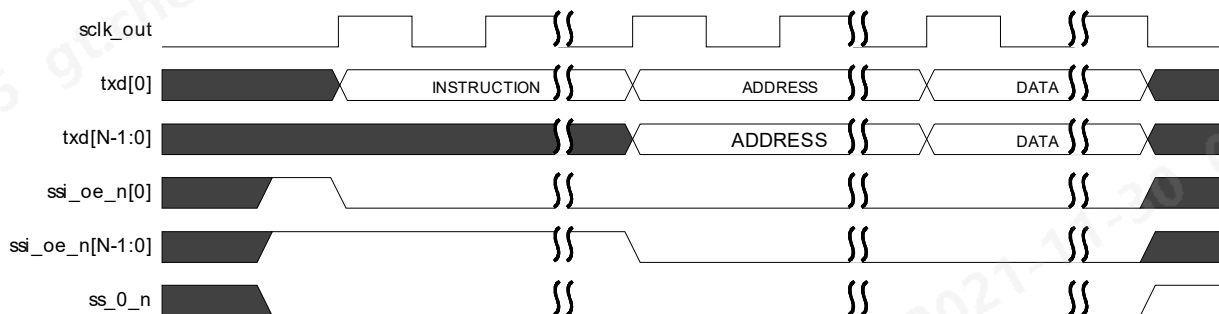
- Case C: Instruction and Address both transmitted in Enhanced SPI format

    For this, SPI_CTRLR0.TRANS_TYPE field must be set to 10. Figure 2-48 shows the timing diagram in which instruction and address are transmitted in SPI format specified in the CTRLR0.SPI_FRF field. The value of N is: 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01.

Figure 2-48   Instruction and Address Both Transmitted in Enhanced SPI Format
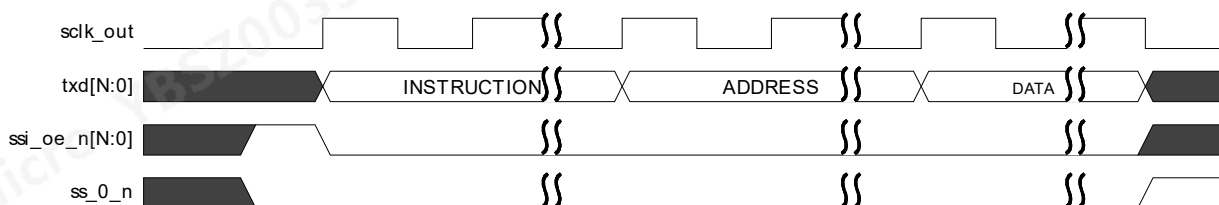
■ Case D: Instruction only transfer in enhanced SPI format

For this, SPI_CTRLR0.TRANS_TYPE field must be set to 10. Figure 2-49 shows the timing diagram for such a transfer. The value of N is: 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01.

Figure 2-49   Instruction only transfer in enhanced SPI Format
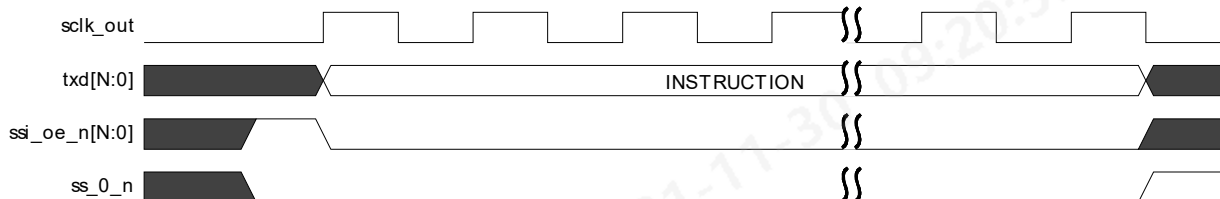
2.8.4.2   Read Operation in Enhanced SPI Modes

A Dual, Quad, or Octal SPI read operation can be divided into four phases:

■ Instruction phase

■ Address phase

■ Wait cycles

■ Data phase

Wait Cycles can be programmed using SPI_CTRLR0.WAIT_CYCLES field. The value programmed into SPI_CTRLR0.WAIT_CYCLES is mapped directly to sclk_out times. For example, WAIT_CYCLES=0 indicates no Wait, WAIT_CYCLES=1, indicates 1 wait cycle and so on. The wait cycles are introduced for target slave to change their mode from input to output and the wait cycles can vary for different devices.

For a READ operation, DW_apb_ssi sends instruction and control data once and waits until it receives NDF (CTRLR1 register) number of data frames and then de-asserts slave select signal.

Figure 2-50 shows a typical read operation in dual quad SPI mode. The value of N is: 7 if SSI_SPI_MODE is set to Octal mode, 3 if SSI_SPI_MODE is set to Quad mode, and 1 if SSI_SPI_MODE is set to Dual mode.

Figure 2-50   Typical Read Operation in Enhanced SPI Mode
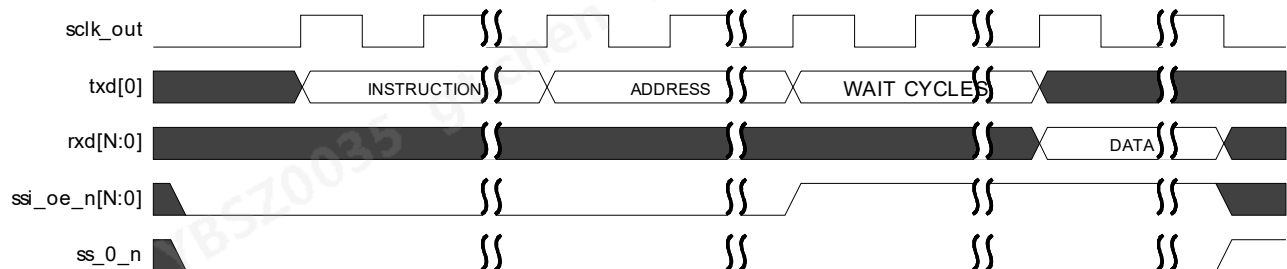
To initiate a dual/quad/octal read operation, CTRLR0.SPI_FRF must be set to 01/10/11 respectively. This sets the transfer type, now for each read command data is transferred in the format specified in CTLR0.SPI_FRF field.

Following are the possible cases of write operation in enhanced SPI modes:

■   Case A: Instruction and address both transmitted in standard SPI format

    For this, SPI_CTRLR0.TRANS_TYPE field should be set to 00. Figure 2-51 shows the timing diagram when both instruction and address are transferred in standard SPI format. The figure also shows WAIT cycles after address, which can be programmed in the SPI_CTRLR0.WAIT_CYCLES field. The value of N is 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01.

Figure 2-51   Instruction and Address Transmitted in Standard SPI Format
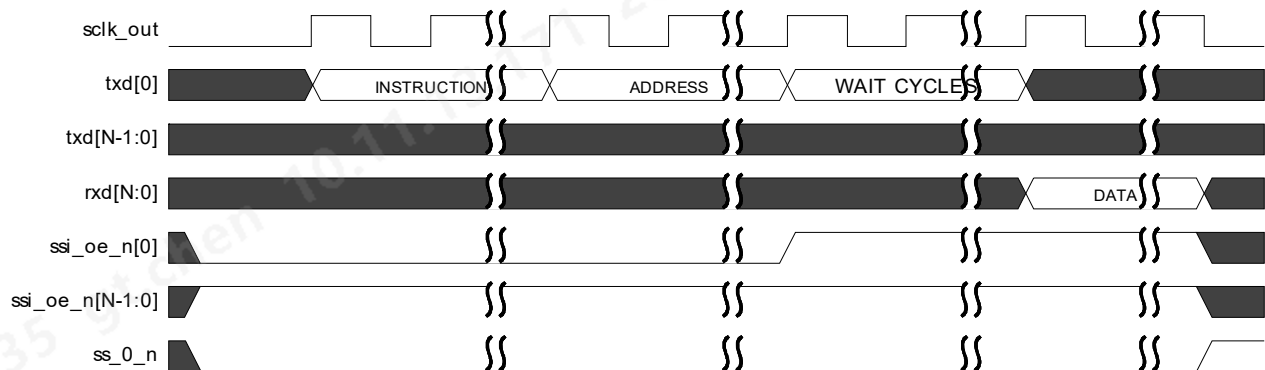
■   Case B: Instruction transmitted in standard and address transmitted in dual SPI format

    For this, SPI_CTRLR0.TRANS_TYPE field should be set to 01. Figure 2-52 shows the timing diagram in which instruction is transmitted in standard format and address is transmitted in dual SPI format. The value of N is 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01.

Figure 2-52   Instruction Transmitted in Standard and Address Transmitted in Enhanced SPI Format
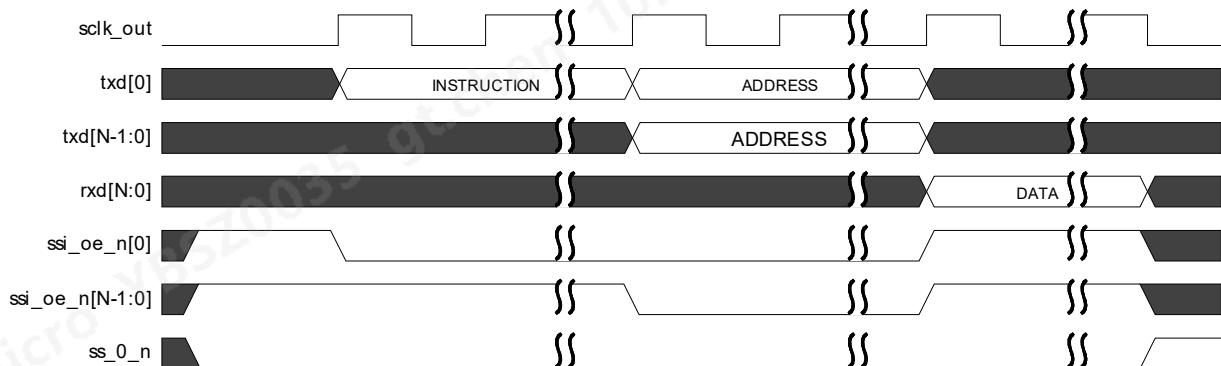
■ Case C: Instruction and Address both transmitted in Dual SPI format

For this, SPI_CTRLR0.TRANS_TYPE field must be set to 10. Figure 2-53 shows the timing diagram in which both instruction and address are transmitted in dual SPI format. The value of N is: 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01.

Figure 2-53   Instruction and Address Transmitted in Enhanced SPI Format
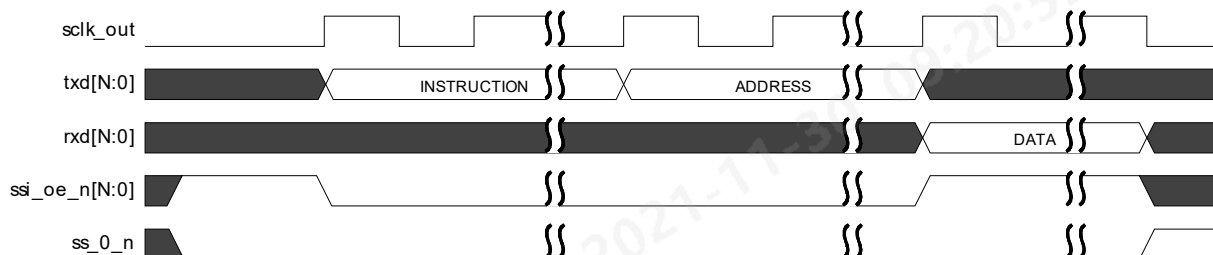
■ Case D: No Instruction, No Address READ transfer

For this, SPI_CTRLR0.ADDR_L and SPI_CTRLR0.INST_L must be set to 0 and SPI_CTRLR0.WAIT_CYCLES must be set to a non-zero value. Table 2-3 lists the ADDR_L decode value and the respective description for enhanced (Dual/Quad/Octal) SPI modes.

Table 2-3   ADDR_L Decode in Enhanced SPI Mode

ADDR_L Decode Value	Description
0000	0-bit Address Width
0001	4-bit Address Width
0010	8-bit Address Width
0011	12-bit Address Width
0100	16-bit Address Width
0101	20-bit Address Width
0110	24-bit Address Width
0111	28-bit Address Width
1000	32-bit Address Width
1001	36-bit Address Width
1010	40-bit Address Width
1011	44-bit Address Width
1100	48-bit Address Width

Table 2-3   ADDR_L Decode in Enhanced SPI Mode

ADDR_L Decode Value	Description
1101	52-bit Address Width
1110	56-bit Address Width
1111	60-bit Address Width

Figure 2-54 shows the timing diagram for such type of transfer. The value of N is: 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01. To initiate this transfer, the software has to perform a dummy write in the data register (DR), DW_apb_ssi waits for programmed wait cycles and then fetch the amount of data specified in NDF field.

Figure 2-54   No Instruction and No Address READ Transfer
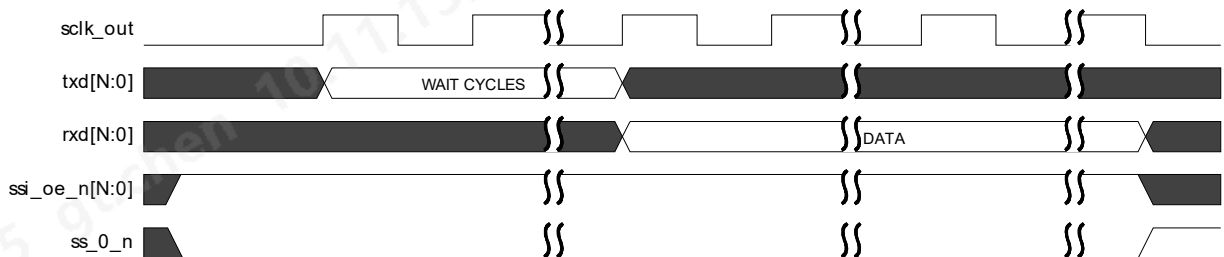
2.8.4.3   Advanced I/O Mapping for Enhanced SPI Modes

The Input/Output mapping for enhanced SPI modes (dual, quad, and octal) is configurable using the SSI_IO_EN parameter, which configures whether the I/O mapping of a slave device is hardcoded inside the DW_apb_ssi. When SSI_IO_MAP_EN is set to 1, the rxd[1] signal is used to sample incoming data in standard SPI mode of operation.

For other protocols (such as SSP and Microwire), the I/O mapping remains the same. Therefore, it is easy for other protocols to connect with any device that supports Dual/Quad SPI operation because other protocols do not require a multiplex logic to exist outside the design.

Figure 2-55 shows the I/O mapping of DW_apb_ssi in Quad mode with another SPI device that supports the Quad mode. As illustrated in Figure 2-55, the IO[1] pin is used as DO in standard SPI mode of operation and it is connected to rxd[1] pin, which is sampling the input in the standard mode of operation.

Figure 2-55 Advanced I/O Mapping in Quad SPI Modes
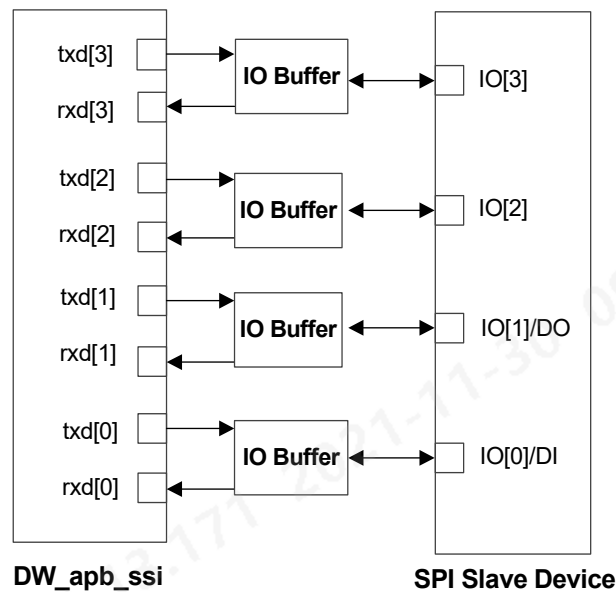
2.8.5 Dual Data-Rate (DDR) Support in SPI Operation

In standard operations, data transfer in SPI modes occur on either the positive or negative edge of the clock. For improved throughput, the dual data-rate transfer can be used for reading or writing to the memories.

The DDR mode supports the following modes of SPI protocol:

- SCPH=0 & SCPOL=0 (Mode 0)
- SCPH=1 & SCPOL=1 (Mode 3)

DDR commands enable data to be transferred on both edges of clock. Following are the different types of DDR commands:

- Address and data are transmitted (or received in case of data) in DDR format, while instruction is transmitted in standard format.
- Instruction, address, and data are all transmitted or received in DDR format.

The DDR_EN (SPI_CTRLR0[16]) bit is used to determine if the Address and data have to be transferred in DDR mode and INST_DDR_EN (SPI_CTRLR0[17]) bit is used to determine if Instruction must be transferred in DDR format. These bits are only valid when the CTRLR0.SPI_FRF bit is set to be in Dual, Quad or Octal mode.

Figure 2-56 describes a DDR write transfer where instructions are continued to be transmitted in standard format. In Figure 2-56, the value of N is 7 if CTRLR0.SPI_FRF is set to 11, 3 if CTRLR0.SPI_FRF is set to 10, and 1 if CTRLR0.SPI_FRF is set to 01.

Figure 2-56   DDR Transfer with SCPH=0 and SCPOL=0

INST = Instruction Phase
A3,A2,A1,A0 - Address Bytes
D3,D2,D1,D0 - Data Bytes
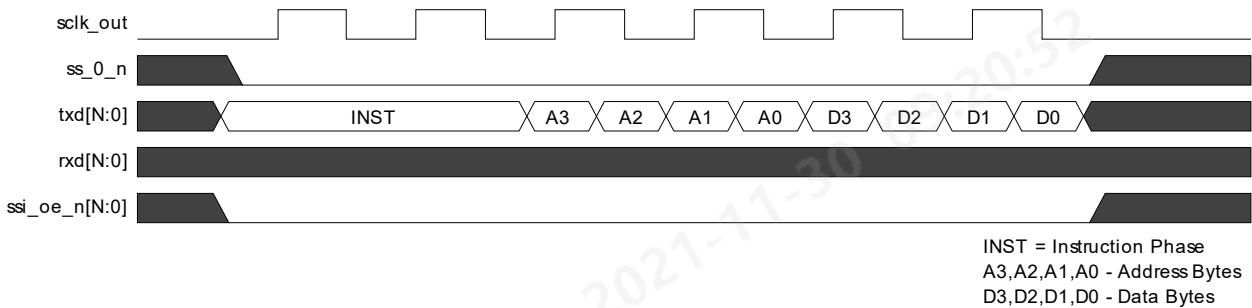
Figure 2-57 describes a DDR write transfer where instruction, address and data all are transferred in DDR format.

Figure 2-57   DDR Transfer with Instruction, Address and Data Transmitted in DDR Format

INST-1,INST-2 = Instruction Bytes
A3,A2,A1,A0 - Address Bytes
D3,D2,D1,D0 - Data Bytes
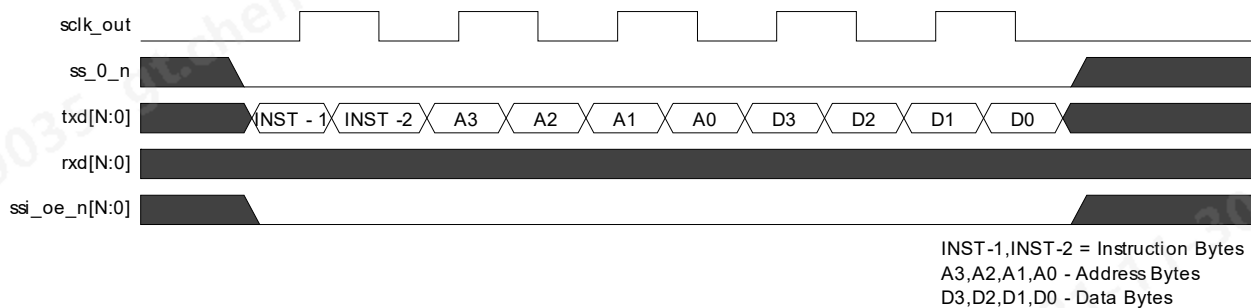
👉**Note**   In the DDR transfer, address and instruction cannot be programmed to a value of 0

2.8.5.1   Transmitting Data in DDR Mode

In DDR mode, data is transmitted on both edges so that it is difficult to sample data correctly. DW_apb_ssi uses an internal register to determine the edge on which the data should be transmitted. This ensures that the receiver is able to get a stable data while sampling. The internal register (DDR_DRIVE_EDGE) determines the edge on which the data is transmitted. DW_apb_ssi sends data with respect to baud clock, which is an integral multiple of the internal clock (ssi_clk * BAUDR). The data needs to be transmitted within half clock cycle (BAUDR/2), therefore the maximum value for DDR_DRIVE_EDGE is equal to [(BAUDR/2)-1]. If the programmed value of DDR_DRIVE_EDGE is 0 then data is transmitted edge-aligned with respect to sclk_out (baud clock). If the programmed value of DDR_DRIVE_EDGE is 1 then the data is transmitted one ssi_clk before the edge of sclk_out.

> 👉 **Note**   If the baud rate is programmed to be 2, then the data is always edge aligned.

Figure 2-58, Figure 2-59, and Figure 2-60 show examples of how data is transmitted using different values of the DDR_DRIVE_EDGE register. The green arrows in these examples represent the points where data is driven. Baud rate used in all these examples is 12. In Figure 2-58, transmit edge and driving edge of the data are the same. This is default behavior in DDR mode.

Figure 2-58   Transmit Data With DDR_DRIVE_EDGE = 0

INST = Instruction Phase
A3,A2,A1,A0 - Address Bytes
D3,D2,D1,D0 - Data Bytes
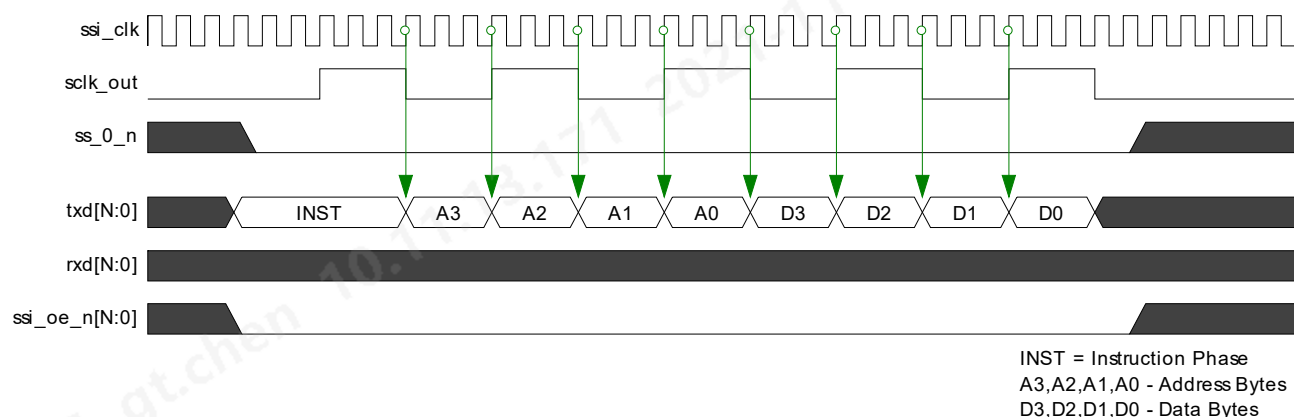
Figure 2-58 shows the default behavior in which the transmit and driving edge of the data is the same.

Figure 2-59   Transmit Data With DDR_DRIVE_EDGE = 1

INST = Instruction Phase
A3,A2,A1,A0 - Address Bytes
D3,D2,D1,D0 - Data Bytes
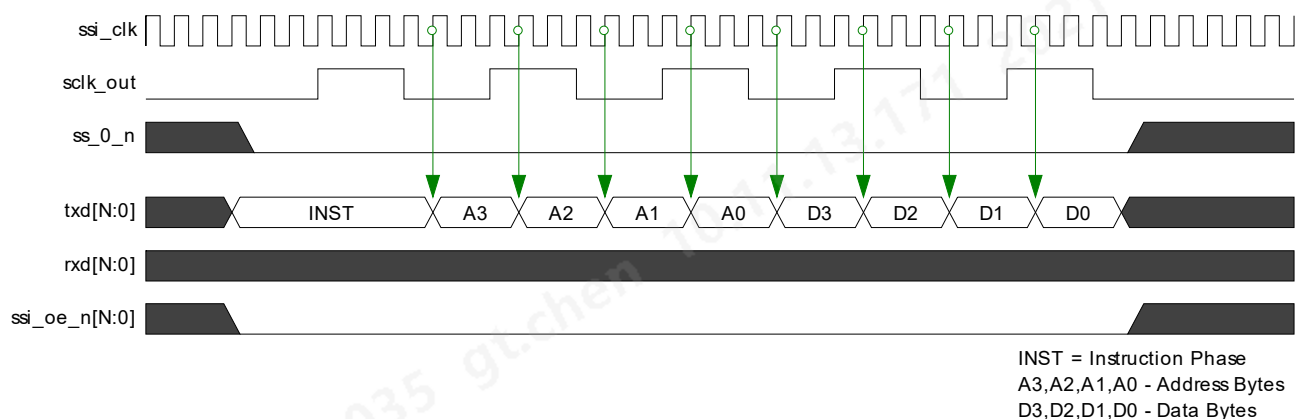
Figure 2-60   Transmit Data With DDR_DRIVE_EDGE = 2

INST = Instruction Phase
A3,A2,A1,A0 - Address Bytes
D3,D2,D1,D0 - Data Bytes
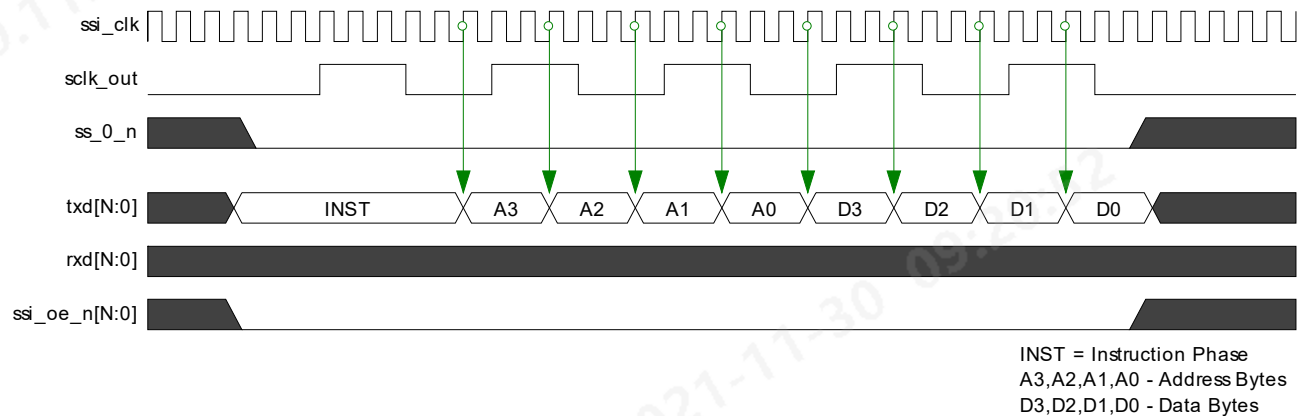
2.8.6   Read Data Strobe Signal Support

To achieve high frequency, data strobe signaling is used in read operations. The device provides a data strobe, known as read data strobe (rxds), signal along with read data, which determines the output data valid window. The data is edge aligned with respect to the rxds signal.

DW_apb_ssi provides the SSI_HAS_RXDS configurable parameter to include the data strobe signal. When SSI_HAS_RXDS is set to 1, you must the SPI_CTRLR0.RXDS_EN register to 1 to sample incoming data with respect to the read data strobe signal. If you do not set the SPI_CTRLR0.RXDS_EN signal to 1, the data is sampled based on the existing logic.

👉 **Note**   The data must be stable around the clock edge of the data strobe signal while sampling on both edges.

2.8.6.1   Design for Test

When SS_HAS_RXDS is set to 1, then rxds is used to capture data on both the edges. One of the internal shifter uses negative edge of the rxds signal to capture the data on the rxd line. During scan testing, these flops may remain uncovered. Therefore, you must connect scan_mode to chip-level scan mode. During scan mode (scan_mode =1), the clock input for these flip-flops is connected to ssi_clk, rather than rxds which leaves the register testable and subsequent downstream points controllable.

2.8.7   XIP Mode Support in SPI Mode

The eXecute In Place (XIP) mode enables transfer of SPI data directly through the APB interface without writing the data register of DW_apb_ssi. XIP mode can be enabled in DW_apb_ssi by selecting the SSI_XIP_EN configuration parameter, which includes an extra sideband signal xip_en, on the APB interface. This signal indicates whether APB transfers are register read-write or XIP reads. If the xip_en signal is driven to 1, DW_apb_ssi expects only read request on the APB interface. This request is translated to SPI read on the serial interface and soon after the data is received, the data is returned to the APB interface in the same transaction.

> **Note**
> ■ XIP mode can be enabled only when APB interface type is set to APB 3.0 and only APB reads are supported during an XIP operation
>
> ■ The xip_en signal must be driven HIGH one cycle before any read is initiated on the APB interface.

The paddr signal is used to derive the address to be sent on the XIP interface. The address length is derived from the SPI_CTRLR0.ADDR_L field, and relevant bits from paddr ([SPI_CTRLR0.ADDR_L-1:0]) are transferred as address to the SPI interface.

2.8.7.1 Read Operation in XIP Mode

The XIP operation is supported only in enhanced SPI modes (Dual, Quad, and Octal) of operation. Therefore, the CTRLR0.SPI_FRF bit should not be programmed to 0. An XIP read operation is divided into two phases:

- ■ Address phase
- ■ Data phase

For an XIP read operation

1. Set the SPI frame format and data frame size value in CTRLR0 register. Note that the value of the maximum data frame size is equal to the APB_DATA_WIDTH parameter.

2. Set the Address length, Wait cycles, and transaction type in the SPI_CTRLR0 register. Note that the maximum address length is 32.

After these settings, you can initiate a read transaction through the APB interface which is transferred to SPI peripheral using programmed values. Figure 2-61 shows the typical XIP transfer. The Value of N = 1, 3 and 7 for SPI mode Dual, Quad and Octal modes, respectively.

Figure 2-61 Typical Read Operation in XIP Mode
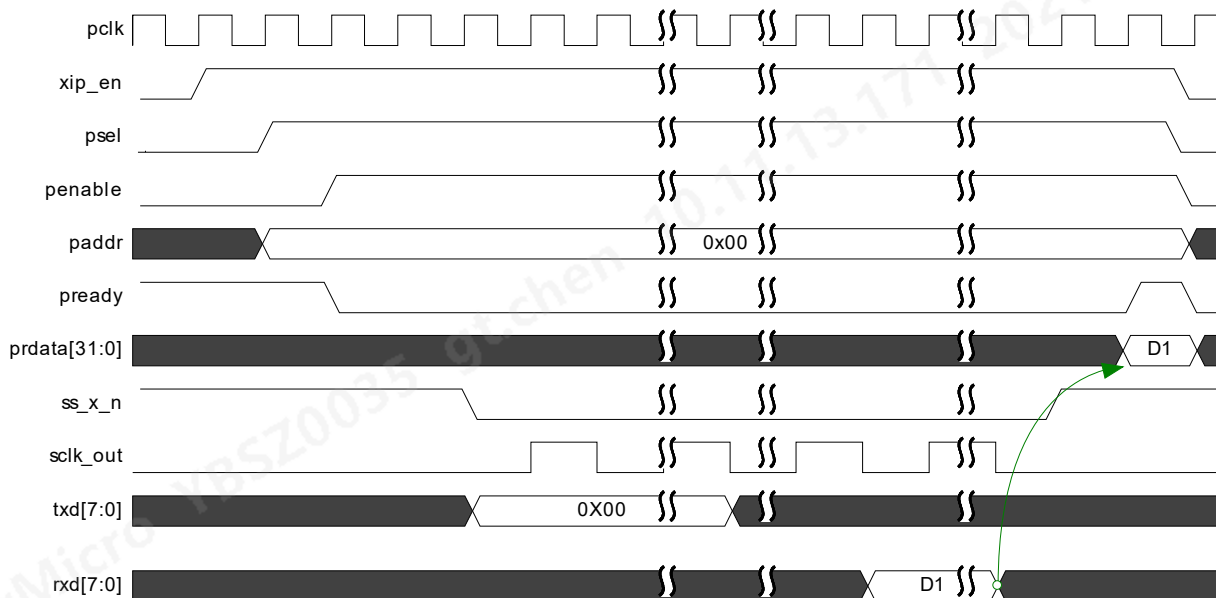
2.8.8   Data Mask Support for SPI

During any data transfer, if you want to selectively update the data bytes in the memory, and mask some bytes, DW_apb_ssi provides the txd_dm signals to mask the data on the txd line. The output enable bit for the data mask pin is txd_dm_oe_n, which is used to enable txd_dm. The data mask pin is only used when DW_apb_ssi is performing WRITE transaction to a memory in enhanced SPI formats (Dual/Quad/Octal). The SSI_SPI_DM_EN parameter is used to include data mask signal on DW_apb_ssi interface.

For read transfers, the txd_dm_oe_n signal remains HIGH for the entire transfer. you can mask the data bytes while writing in the data register (DR) of DW_apb_ssi by using the pstrb signal. A pstrb signal is mapped to txd_dm and transmitted during the data phase.

Figure 2-62 represents an example usage of the data mask signal in extended SPI DDR transfer. The data byte D2 is being masked in this transfer by asserting the txd_dm signal.

Figure 2-62   Data Mask Signal Usage Example

INST-1,INST-2 = Instruction Bytes
A3,A2,A1,A0 - Address Bytes
D3,D2,D1,D0 - Data Bytes
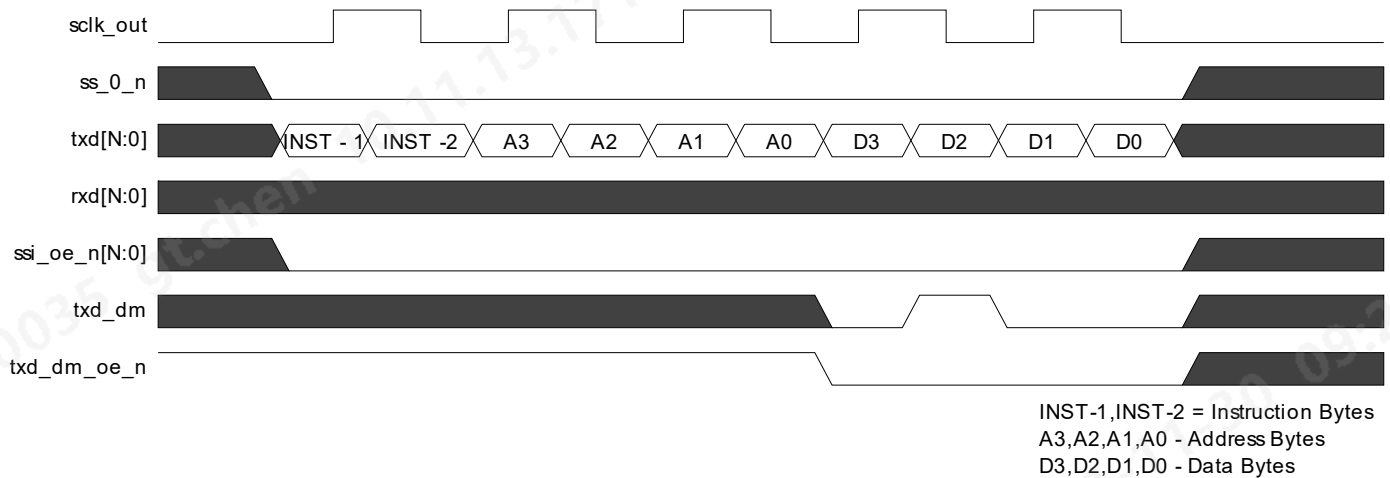
2.9   DMA Controller Interface

The DW_apb_ssi has optional built-in DMA capability which 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_ssi 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_ssi 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_ssi 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 *DesignWare DW_ahb_dmac Databook*. Other DMA controllers act in a similar manner.

The DW_apb_ssi uses two DMA channels, one for the transmit data and one for the receive data. The DW_apb_ssi has these DMA registers:

- DMACR – Control register to enable DMA operation.
- DMATDLR – Register to set the transmit the FIFO level at which a DMA request is made.
- DMARDLR – Register to set the receive FIFO level at which a DMA request is made.

The DW_apb_ssi uses the following handshaking signals to interface with the DMA controller.

- dma_tx_req
- dma_tx_single
- dma_tx_ack
- dma_rx_req
- dma_rx_single
- dma_rx_ack

For more information about these signals, see "Signal Descriptions" on page 105. They are discussed further in the "Handshaking Interface Operation" on page 84.

The DMA output dma_finish is a status signal to indicate that the DMA block transfer is complete; for more information on the dma_finish signal, see the Signals chapter in the *DesignWare DW_ahb_dmac Databook*. The DW_apb_ssi does not use this status signal, and therefore it does not appear in the I/O signal list.

To enable the DMA Controller interface on the DW_apb_ssi, you must write the DMA Control Register (DMACR). Writing a 1 into the TDMAE bit field of DMACR register enables the DW_apb_ssi transmit handshaking interface. Writing a 1 into the RDMAE bit field of the DMACR register enables the DW_apb_ssi receive handshaking interface.

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

Table 2-4   DMA Transmit Data Level (DMATDL) Decode Value

DMATDL Value	Description
0000_0000	dma_tx_req is asserted when 0 data entries are present in the transmit FIFO
0000_0001	dma_tx_req is asserted when 1 or less data entry is present in the transmit FIFO
0000_0010	dma_tx_req is asserted when 2 or less data entries are present in the transmit FIFO
0000_0011	dma_tx_req is asserted when 3 or less data entries are present in the transmit FIFO
...	...
...	...
1111_1100	dma_tx_req is asserted when 252 or less data entries are present in the transmit FIFO
1111_1101	dma_tx_req is asserted when 253 or less data entries are present in the transmit FIFO
1111_1110	dma_tx_req is asserted when 254 or less data entries are present in the transmit FIFO

Table 2-4   DMA Transmit Data Level (DMATDL) Decode Value (Continued)

DMATDL Value	Description
1111_1111	dma_tx_req is asserted when 255 or less data entries are present in the transmit FIFO

Table 2-5 provides description for different DMA Receive Data Level values.

Table 2-5   DMA Receive Data Level (DMARDL) Decode Value

DMARDL Value	Description
0000_0000	dma_rx_req is asserted when 1 or more data entries are present in the receive FIFO
0000_0001	dma_rx_req is asserted when 2 or more data entries are present in the receive FIFO
0000_0010	dma_rx_req is asserted when 3 or more data entries are present in the receive FIFO
0000_0011	dma_rx_req is asserted when 4 or more valid data entries are present in the receive FIFO
...	...
...	...
1111_1100	dma_rx_req is asserted when 253 or more data entries are present in the receive FIFO
1111_1101	dma_rx_req is asserted when 254 or more data entries are present in the receive FIFO
1111_1110	dma_rx_req is asserted when 255 or more data entries are present in the receive FIFO
1111_1111	dma_rx_req is asserted when 256 data entries are present in the receive FIFO

2.9.1   Overview of Operation

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 the DW_apb_ssi; this is programmed into the BLOCK_TS field of the CTL*x* register.

The block is broken into a number of transactions, each initiated by a request from the DW_apb_ssi. The DMA Controller must also be programmed with the number of data items (in this case, DW_apb_ssi 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 2-63 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.

Figure 2-63   Breakdown of DMA Transfer into Burst Transactions

12 Data Items

```
┌─────────────────┐
│       DMA       │
│ Multi-block     │
│ Transfer Level  │
└─────────────────┘
```

12 Data Items

```
┌─────────┐
│   DMA   │
│  Block  │
│  Level  │
└─────────┘
```

```
┌─────────────┐      ┌─────────────┐      ┌─────────────┐
│ DMA Burst   │      │ DMA Burst   │      │ DMA Burst   │
│ Transaction 1│     │ Transaction 2│     │ Transaction 3│
└─────────────┘      └─────────────┘      └─────────────┘
```

4 Data Items   4 Data Items   4 Data Items
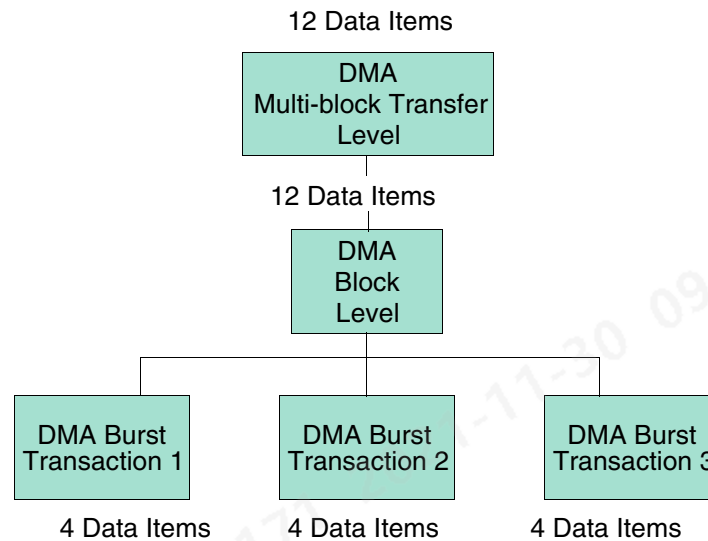
Block Size: DMA.CTLx.BLOCK_TS=12
Number of data items per source burst transaction: DMA.CTLx.SRC_MSIZE = 4
SSI receive FIFO watermark level: SSI.DMARDLR + 1 = DMA.CTLx.SRC_MSIZE = 4
   (for more information, see discussion on)

If the DW_apb_ssi makes a transmit request to this channel, four data items are written to the DW_apb_ssi transmit FIFO. Similarly, if the DW_apb_ssi makes a receive request to this channel, four data items are read from the DW_apb_ssi receive FIFO. Three separate requests must be made to this DMA channel before all 12 data items are written or read.

> ☞ **Note**   The source and destination transfer width settings in the DW_ahb_dmac – DMA.CTLx.SRC_TR_WIDTH and DMA.CTLx.DST_TR_WIDTH – should be set to 3'b001 because the DW_apb_ssi FIFOs are 16 bits wide.

When the block size programmed into the DMA Controller is not a multiple of the burst transaction length, as shown in Figure 2-64, a series of burst transactions followed by single transactions are needed to complete the block transfer.

Figure 2-64   Breakdown of DMA Transfer into Single and Burst Transactions
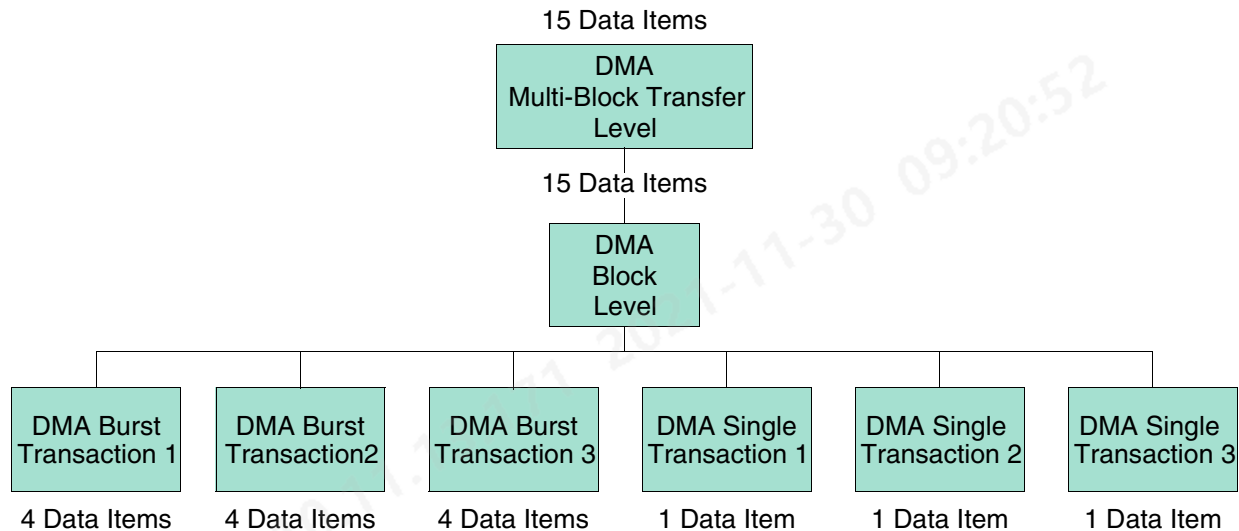
ck Size: DMA.CTLx.BLOCK_TS=15
mber of data items per burst transaction: DMA.CTLx.DEST_MSIZE = 4
SI transmit FIFO watermark level: SSI.DMATDLR = DMA.CTLx.DEST_MSIZE = 4
 (for more information, see discussion on)

2.9.2   Transmit Watermark Level and Transmit FIFO Underflow

During DW_apb_ssi 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 (DMATDLR) 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 CTLx.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 (underflow). To prevent this condition, you must set the watermark level correctly.

2.9.3   Choosing the Transmit Watermark Level

Consider the example where the assumption is made:

DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - SSI.DMATDLR

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.

2.9.3.1 Case 1: DMATDLR = 2

Figure 2-65 Case 1 Watermark Levels
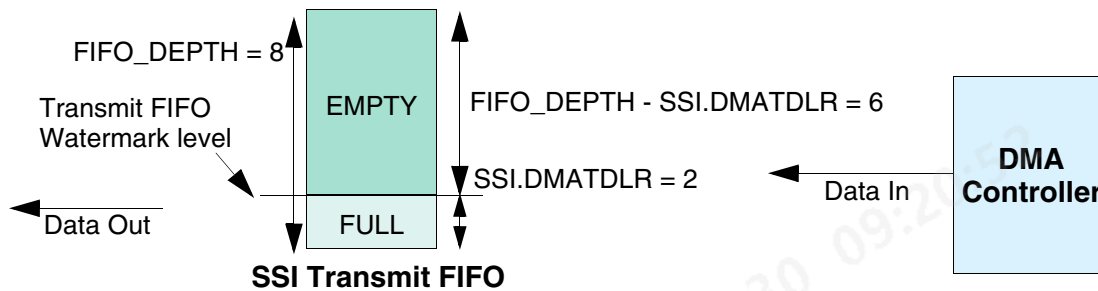
- Transmit FIFO watermark level = SSI.DMATDLR = 2

- DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - SSI.DMATDLR = 6

- SSI transmit FIFO_DEPTH = 8

- DMA.CTLx.BLOCK_TS = 30

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, SSI.DMATDLR, is quite low. Therefore, the probability of an SSI underflow is high where the SSI 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.

2.9.3.2 Case 2: DMATDLR = 6

Figure 2-66 Case 2 Watermark Levels
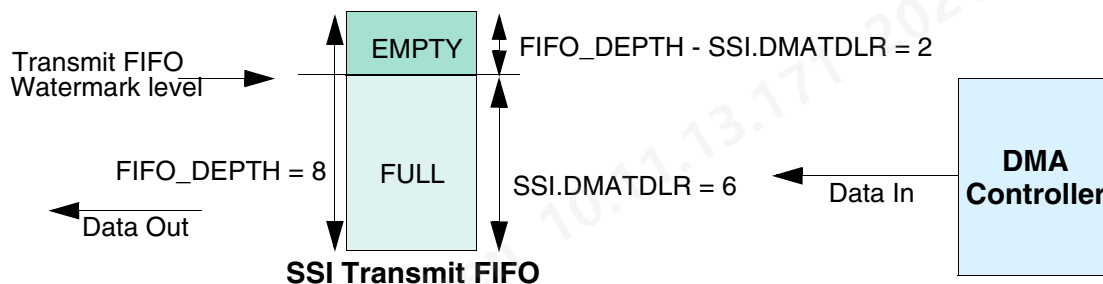
- Transmit FIFO watermark level = SSI.DMATDLR = 6

- DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - SSI.DMATDLR = 2

- SSI transmit FIFO_DEPTH = 8

- DMA.CTLx.BLOCK_TS = 30

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, SSI.DMATDLR, is high. Therefore, the probability of an SSI underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the SSI 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 SSI 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.

2.9.4   Selecting DEST_MSIZE and Transmit FIFO Overflow

As can be seen from Figure 2-66, programming DMA.CTLx.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 SSI 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 <= SSI.FIFO_DEPTH - SSI.DMATDLR   **(1)**

In Case 2: DMATDLR = 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.CTLx.DEST_MSIZE. Thus, the transmit FIFO may be full, but not overflowed, at the completion of the burst transaction.

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

> DMA.CTLx.DEST_MSIZE = SSI.FIFO_DEPTH - SSI.DMATDLR   **(2)**

This is the setting used in Figure 2-64.

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 full at the end of a DMA burst transfer if the SSI has successfully transmitted one data item or more on the SSI serial transmit line during the transfer.

2.9.5   Receive Watermark Level and Receive FIFO Overflow

During DW_apb_ssi 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, DMARDLR+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.

2.9.6 Choosing the Receive Watermark Level

Similar to choosing the transmit watermark level described earlier, the receive watermark level, DMARDLR+1, should be set to minimize the probability of overflow, as shown in Figure 2-67. It is a tradeoff between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

Figure 2-67   SSI Receive FIFO
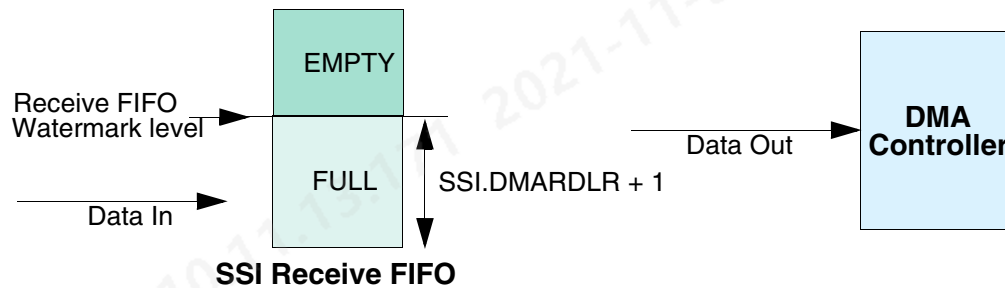
2.9.7 Selecting SRC_MSIZE and Receive FIFO Underflow

As seen in Figure 2-67, 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) below 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.CTLx.SRC_MSIZE – the receive FIFO may be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA.CTLx.SRC_MSIZE should be set at the watermark level; that is:

> DMA.CTLx.SRC_MSIZE = SSI.DMARDLR + 1   **(3)**

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

☞ **Note**   The receive FIFO is not empty at the end of the source burst transaction if the SSI has successfully received one data item or more on the SSI serial receive line during the burst.

2.9.8 Handshaking Interface Operation

The following sections discuss the DW_apb_ssi handshaking interface.

2.9.8.1 dma_tx_req, dma_rx_req

The request signals for source and destination, dma_tx_req and dma_rx_req, 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 signal/dma_rx_req to identify a request on the channel. Upon reception of the dma_tx_ack/dma_rx_ack signal from the DW_ahb_dmac to indicate the burst transaction is complete, the DW_apb_ssi de-asserts the burst request signals, dma_tx_req/dma_rx_req, until dma_tx_ack/dma_rx_ack is de-asserted by the DW_ahb_dmac.

When the DW_apb_ssi samples that dma_tx_ack/dma_rx_ack is de-asserted, it can re-assert the dma_tx_req/dma_rx_req 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 2-68 shows a timing diagram of a burst transaction where pclk = hclk.

Figure 2-68   Burst Transaction – pclk = hclk
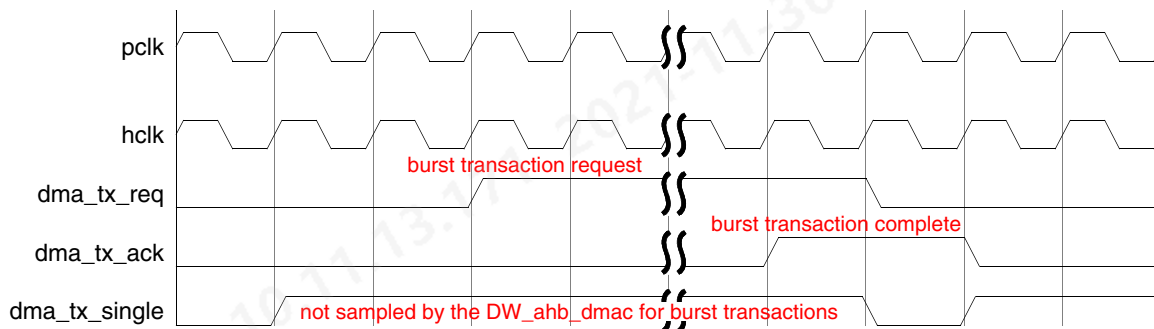
Figure 2-69 shows two back-to-back burst transactions where the hclk frequency is twice the pclk frequency.

Figure 2-69   Back-to-Back Burst Transactions – hclk = 2*pclk
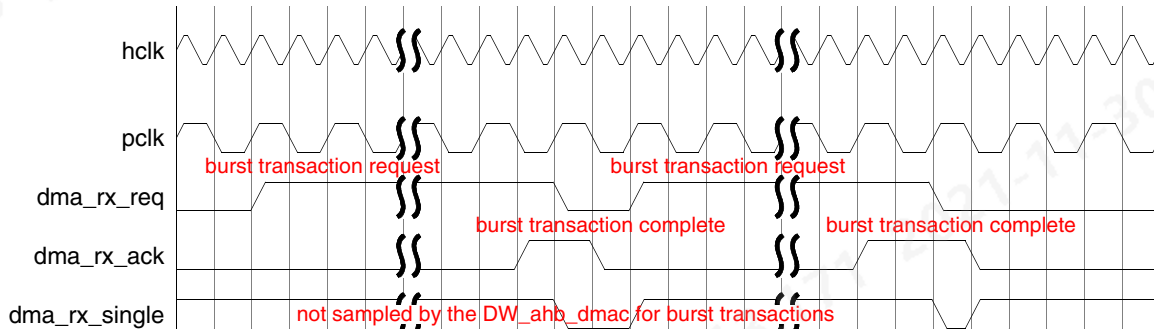
The handshaking loop is as follows:

dma_tx_req/dma_rx_req asserted by DW_apb_ssi

-> dma_tx_ack/dma_rx_ack asserted by DW_ahb_dmac

-> dma_tx_req/dma_rx_req de-asserted by DW_apb_ssi

-> dma_tx_ack/dma_rx_ack de-asserted by DW_ahb_dmac

-> dma_tx_req/dma_rx_req re-asserted by DW_apb_ssi, if back-to-back transaction is required

> ☞ **Note** The burst transaction request signals, dma_tx_req and dma_rx_req, are generated in the DW_apb_ssi off pclk and sampled in the DW_ahb_dmac by hclk. The acknowledge signals, dma_tx_ack and dma_rx_ack, are generated in the DW_ahb_dmac off hclk and sampled in the DW_apb_ssi of pclk. The handshaking mechanism between the DW_ahb_dmac and the DW_apb_ssi 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.

Two things to keep in mind:

■ The burst request lines, dma_tx_req signal/dma_rx_req, once asserted remain asserted until their corresponding dma_tx_ack/dma_rx_ack signal is received even if the respective FIFOs drop below their watermark levels during the burst transaction.

■ The dma_tx_req/dma_rx_req signals are de-asserted when their corresponding dma_tx_ack/dma_rx_ack signals are asserted, even if the respective FIFOs exceed their watermark levels.

2.9.8.2 dma_tx_single, dma_rx_single

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

The dma_rx_single signal is asserted when there is at least one valid data entry in the receive FIFO, and is cleared when the dma_rx_ack signal is active. The dma_rx_single signal is re-asserted when the dma_rx_ack 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, CTLx.BLOCK_TS, that is programmed into the DW_ahb_dmac is not a multiple of the burst transaction length, CTLx.SRC_MSIZE, CTLx.DEST_MSIZE, as shown in Figure 2-64. 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_ssi 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_ssi is as follows:

DMA.CTLx.SRC_MSIZE = SSI.DMARDLR + 1 = 4

DMA.CTLx.BLOCK_TS = 12

For the example in Figure 2-63, 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_ssi 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 = SSI.DMARDLR + 1 = 4

DMA.CTLx.BLOCK_TS = 15

The first 12 data items are transferred as already described using 3 burst transactions. But when the last three data frames enter the receive FIFO, the dma_rx_req signal is not activated because the FIFO level is below the watermark level. The DW_ahb_dmac samples dma_rx_single 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 2-70 shows a single transaction.

Figure 2-70   Single Transaction
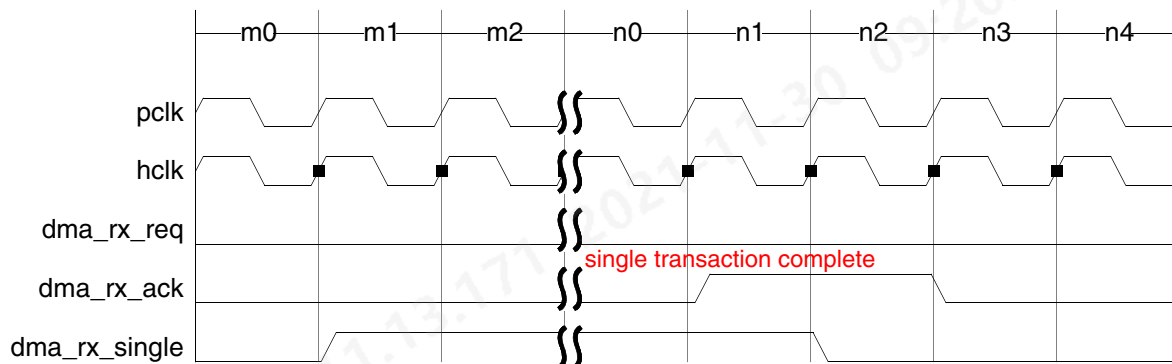
The handshaking loop is as follows:

dma_tx_single/dma_rx_single asserted by DW_apb_ssi

-> dma_tx_ack/dma_rx_ack asserted by DW_ahb_dmac

-> dma_tx_single/dma_rx_single de-asserted by DW_apb_ssi

-> dma_tx_ack/dma_rx_ack de-asserted by DW_ahb_dmac.

Figure 2-71 shows a burst transaction, followed by three back-to-back single transactions, where the hclk frequency is twice the pclk frequency.

Figure 2-71   Burst Transaction + 3 Back-to-Back Singles – hclk = 2*pclk
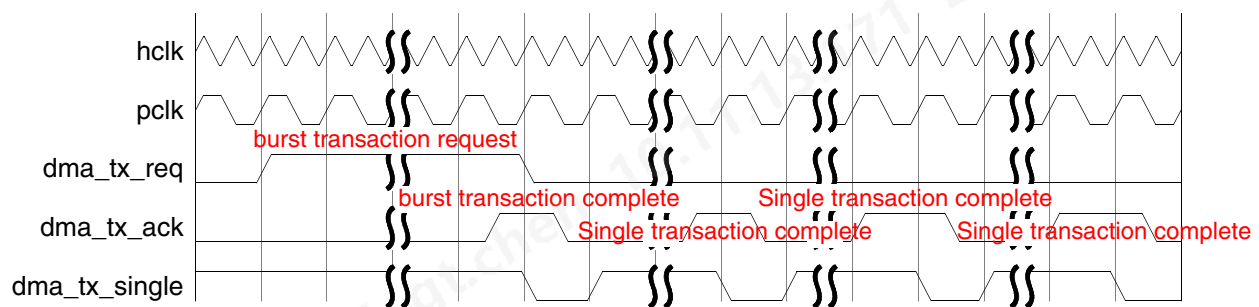
👉 **Note**   The single transaction request signals, dma_tx_single and dma_rx_single, are generated in the DW_apb_ssi on the pclk edge and sampled in DW_ahb_dmac on hclk. The acknowledge signals, dma_tx_ack and dma_rx_ack, are generated in the DW_ahb_dmac on the hclk edge hclk and sampled in the DW_apb_ssi on pclk. The handshaking mechanism between the DW_ahb_dmac and the DW_apb_ssi supports quasi-synchronous clocks; that is, hclk and pclk must be phase aligned and the hclk frequency must be a multiple of pclk frequency.

2.10 APB Interface

The host processor accesses data, control, and status information on the DW_apb_ssi through the APB interface. The DW_apb_ssi supports APB data bus widths of 8, 16, and 32 bits. APB accesses to the DW_apb_ssi peripheral are described in the following subsections.

2.10.1 Control and Status Register APB Access

Control and status registers within the DW_apb_ssi are byte-addressable. The maximum width of the control or status register in the DW_apb_ssi is 16 bits. Therefore, if the APB data bus is 16 or 32 bits wide, all read and write operations to the DW_apb_ssi control and status registers require only one APB access. When the APB data bus width is 8 bits, you may independently write to the lower byte lane [7:0] of a register by accessing the base address of the register. The upper byte lane [15:8] can be accessed by addressing the register base address + 1.

2.10.2 Data Register APB Access

The data register (DR) within the DW_apb_ssi is 16/32-bits wide (depending upon SSI_MAX_XFER_SIZE) in order to remain consistent with the maximum serial transfer size (data frame). An APB write operation to DR moves data from pwdata into the transmit FIFO buffer. An APB read operation from DR moves data from the receive FIFO buffer onto prdata. There are two possible configurations of the DR register.

2.10.2.1 SSI_MAX_XFER_SIZE = 16

When the APB data bus is 8 bits wide, you must perform two APB accesses to write or read to and from this register. When data are written to the DW_apb_ssi DR, the lower byte [7:0] is stored in an intermediate 8-bit register. Only when the second APB write occurs are the 16 bits of data loaded into the DW_apb_ssi transmit FIFO buffer. When data are read from the DW_apb_ssi DR, the upper byte [15:8] is stored in an intermediate 8-bit register. The lower byte [7:0] is returned on the first APB read, and the stored upper byte [15:8] is returned on the second APB read.

When the APB data bus is 16 bits wide, the DW_apb_ssi DR can be written or read in one APB access. This is the optimal configuration because the full bandwidth of the bus is utilized when accessing this peripheral. When the APB data bus is 32 bits wide, the DW_apb_ssi DR can be written or read in one APB access. It is impossible to write or read two 16-bit FIFO entries in a single 32-bit APB access. Therefore, only half of the APB bus bandwidth is utilized when accessing DW_apb_ssi from a 32-bit APB bus.

2.10.2.2 SSI_MAX_XFER_SIZE = 32

When the APB data bus is 8 bits wide, you must perform four APB accesses to write or read to/from this register. When data is written to the DW_apb_ssi DR, the first three bytes (byte [7:0], byte [15:8], byte [23:16]) are stored in three intermediate 8-bit registers. Only when the fourth APB write occurs are the 32 bits of data loaded into the DW_apb_ssi transmit FIFO buffer. When data is read from the DW_apb_ssi DR, the three upper bytes (byte [31:24], byte [23:16],byte [15:8]) are stored in three intermediate 8-bit registers. The lower byte [7:0] is returned on the first APB read, and the rest of the bytes (byte [31:24], byte [23:16], byte [15:8]) are transferred on each APB read.

When the APB data bus is 16 bits wide, you must perform two APB accesses to write or read to/from this register. When data is written to the DW_apb_ssi DR, the lower word [15:0] is stored in an intermediate 16-bit register. Only when the second APB write occurs are the 32 bits of data loaded into the DW_apb_ssi transmit FIFO buffer. When data is read from the DW_apb_ssi DR, the upper word [31:16] is stored in an

intermediate 16 bit register. The lower word [15:0] is returned on the first APB read, and the upper word [31:16] is returned on the second APB read.

When the APB data bus is 32 bits wide, the DW_apb_ssi DR can be written/read in one APB access. This is the optimal configuration as the full bandwidth of the bus is used when accessing this peripheral.

> **⌖ Note**   The DR register in the DW_apb_ssi occupies sixty-four 32-bit locations of the memory map to facilitate AHB burst transfers. There are no burst transactions on the APB bus itself, but DW_apb_ssi supports the AHB bursts that happen on the AHB side of the AHB/APB bridge. Writing to any of these address locations has the same effect as pushing the data from the pwdata bus into the transmit FIFO. Reading from any of these locations has the same effect as popping data from the receive FIFO onto the prdata bus. The FIFO buffers on the DW_apb_ssi are not addressable.

For more information about the APB Interface and data widths, see "Integration Considerations" on page 193.

2.10.3   APB 3.0 Support

The register interface of DW_apb_ssi is complaint with both APB 2.0 and APB 3.0 specifications. The SSI_APBIF_TYPE parameter is used to select the APB interface type of the register interface. The pready and pslverr signals are included to support the APB 3.0 interface. The pready signal is always kept to its default value (HIGH ==1) for all operations except for XIP operations. For information on the XIP mode of operation, see "XIP Mode Support in SPI Mode" on page 75.

The pslverr signal functionality is enabled when the SSI_APB3_ERR_RESP_EN parameter is set to 1, so that DW_apb_ssi provides any slave error response from APB. If the SSI_APB3_ERR_RESP_EN parameter is set to 1 then DW_apb_ssi provides an error response in following conditions:

- If Transmit FIFO is full and a write is attempted on Data register (DR).
- If receive FIFO is empty and a read is attempted in Data register (DR).
- Any read/write operation is attempted in the data register when DW_apb_ssi is not enabled (SSIENR register set to 0).
- Under XIP mode of operation (xip_en == 1) following conditions results in error response:
  - If any register write occurs during XIP operation.
  - If an XIP read is attempted while another master is active on the SPI bus (Master contention).

2.10.4   APB 4.0 Support

DW_apb_ssi complies with the APB 4.0 specification and to adhere to this compliance, the pstrb signal to the APB interface is available in DW_apb_ssi. In a the write transaction to this interface, the pstrb signal indicates validity of pdata bytes. DW_apb_ssi component selectively writes to the bytes of the addressed register whose corresponding bit in the pstrb signal is high. Bytes strobed low by the corresponding pstrb bits are not modified. The incoming strobe bits for a read transaction is always zero as per protocol.

Figure 2-72 shows the byte lane mapping of the pstrb signal.

Figure 2-72 Byte Lane Mapping of the pstrb Signal

31 24	23 16	15 8	7 0
PSTRB[3]	PSTRB[2]	PSTRB[1]	PSTRB[0]
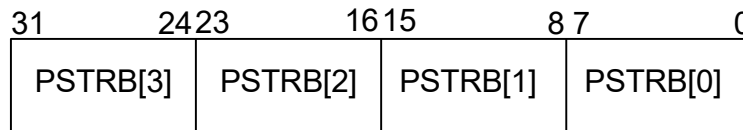
☞ Note
- The pstrb signal is ignored for all the writes to the Data register.
- DW_apb_ssi does not use the pprot signal and it used only for interface consistency.

2.11 Clocks and Resets

2.11.1 Synchronization Depth

The SSI_P2S_SYNC_DEPTH and SSI_S2P_SYNC_DEPTH coreConsultant parameters can be used to specify the number of synchronizer register stages. The user can select the synchronization depth when crossing clock domains between APB and SSI.

- 2 - 2-stage synchronization with positive-edge capturing at both the stages
- 3 - 3-stage synchronization with positive-edge capturing at all stages
- 4 - 4-stage synchronization with positive-edge capturing at all stages

2.11.2 Reset Signals

When the DW_apb_ssi is configured for asynchronous clock operation (SSI_SYNC_CLK = 0), the DW_apb_ssi includes the following reset signals, each dedicated to its own clock domain:

- presetn – resets logic in the pclk clock domain
- ssi_rst_n – resets logic in ssi_clk clock domain

In order to avoid serious operational failures, both clock domains of the DW_apb_ssi must be reset before attempting send or receive on the serial data line. Resetting one clock domain of the DW_apb_ssi without resetting the other clock domain is an illegal operation.

To reset DW_apb_ssi, follow the steps:

1. Assert the ssi_rst_n and presetn signal.

   When asserting the reset signals, the ssi_rst_n signal should be asserted before or at the same time as the presetn signal. This prevents any unexpected activity on the serial line that might result from resetting the programming registers without resetting the serial logic.

2. De-assert the ssi_rst_n signal synchronously with ssi_clk.

3. De-assert the presetn signal synchronously with pclk.

☞ Note
ssi_rst_n and presetn reset signals should be active for at least three cycles of the respective clock signal before de-asserting.

2.12   Endian Conversion Support

DW_apb_ssi supports endian conversion feature for Data register and XIP read transfers. The endianness can be changed using either a register programming bit (CTRLR0.SECONV bit) or an external input signal (endconv_en). These options can be selected using the SSI_INC_ENDCONV parameter.

Figure 2-73 shows the endianness conversion when the CTRLR0.SECONV register bit is set to 1 or the endconv_en signal is driven to 1 for different values of data frame size.

Figure 2-73   Endian Conversion for Data Register and XIP Reads
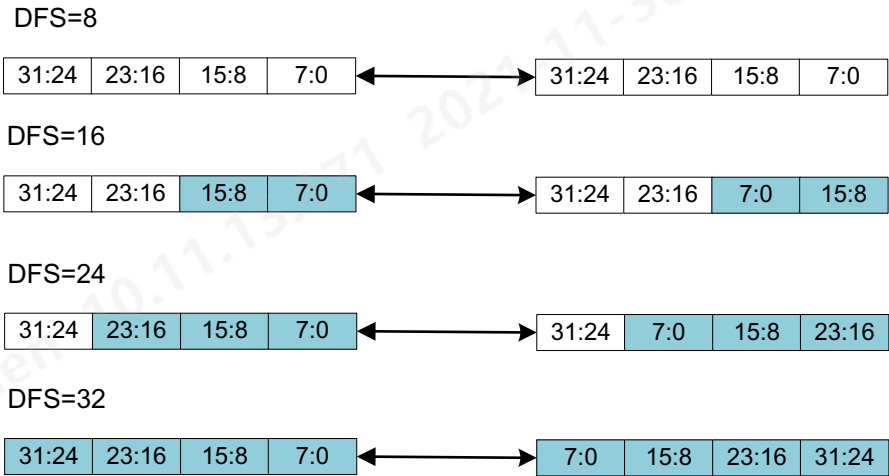
		DW_apb_ssi supports either sideband or register programming mode to choose the endianness for XIP and data register reads.
👉 **Note**	▪	DW_apb_ssi supports either sideband or register programming mode to choose the endianness for XIP and data register reads.
	▪	Endian conversion is supported only for Data Frame Sizes of 16, 24 and 32.
	▪	The Endianness conversion is only applicable for data register or XIP reads, all the other register reads are not affected by this feature.
	▪	endconv_en sideband signal should be stable before enabling DW_apb_ssi.

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.

- Top Level Parameters on
- SPI Parameters on
- Clocking on

3.1 Top Level Parameters

Table 3-1   Top Level Parameters

Label	Description
System Configuration	
APB Interface type	Selects the APB slave interface type. **Values:** ■ APB 2.0 (0) ■ APB 3.0 (1) ■ APB 4.0 (2) **Default Value:** APB 2.0 **Enabled:** Always **Parameter Name:** SSI_APBIF_TYPE
Enable APB error response?	Configures if APB 3.0 interface should provide error response for invalid read and write accesses. If this parameter is set to 0 then pslverr will always be driven to 0. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** SSI_APBIF_TYPE != 0 **Parameter Name:** SSI_APB3_ERR_RESP_EN
APB Data Bus Width	Width of APB data bus **Values:** 8, 16, 32 **Default Value:** 32 **Enabled:** Always **Parameter Name:** APB_DATA_WIDTH
Device Configuration	
Serial master or slave configuration	Configure the device as a master or a slave serial peripheral **Values:** ■ Serial Slave (0) ■ Serial Master (1) **Default Value:** Serial Master **Enabled:** Always **Parameter Name:** SSI_IS_MASTER

Table 3-1   Top Level Parameters (Continued)

Label	Description
Include Enhanced Clock Ratio Architecture?	Configures the device to include new architecture for Transmit and Receive FIFO. This will enable the device to work in clock ratio of 4 and 6 between ssi_clk and sclk_in signals. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** SSI_IS_MASTER==0 and DWC-APB-Advanced-Source source license exists. **Parameter Name:** SSI_ENH_CLK_RATIO
Maximum Transfer Size	Configures the Maximum Transfer Size supported by device. The Receive amd Transmit FIFO widths will be equal to configured value **Values:** ■ 16 Bits (16) ■ 32 Bits (32) **Default Value:** 16 Bits **Enabled:** Always **Parameter Name:** SSI_MAX_XFER_SIZE
Receive FIFO buffer depth	Configures the depth of the receive FIFO buffer. ■ If SSI_SYNC_CLK == 0 or SSI_SPI_MODE != 0, minimum FIFO depth is 4. ■ Else minimum FIFO depth is 2. **Values:** 0x2, ..., 0x100 **Default Value:** 0x8 **Enabled:** Always **Parameter Name:** SSI_RX_FIFO_DEPTH
Transmit FIFO buffer depth	Configures the depth of the transmit FIFO buffer. ■ If SSI_SPI_MODE == 0, minimum FIFO depth is 2. ■ If SSI_SPI_MODE != 0 and SSI_MAX_XFER_SIZE == 16, minimum FIFO depth is 5. ■ If SSI_SPI_MODE != 0 and SSI_MAX_XFER_SIZE == 32, minimum FIFO depth is 3. **Values:** 0x2, ..., 0x100 **Default Value:** 0x8 **Enabled:** Always **Parameter Name:** SSI_TX_FIFO_DEPTH

Table 3-1   Top Level Parameters (Continued)

Label	Description
Number of slave select lines	Configures the number of slave select lines from the DW_apb_ssi master. **Values:** 1, ..., 16 **Default Value:** 1 **Enabled:** SSI_IS_MASTER==1 **Parameter Name:** SSI_NUM_SLAVES
Include - ~~Programmble~~Programmable RXD Sample Logic	Include logic to allow programmable delay on the sample time of the rxd input. When this logic is included, the default sample time of the rxd input can be delayed by a programmable number of ssi_clk cycles. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** SSI_IS_MASTER==1 **Parameter Name:** SSI_HAS_RX_SAMPLE_DELAY
Maximum RXD Sample Delay	Defines the maximum number of ssi_clk cycles that can be used to delay the sampling of the rxd input ■ 2 flip-flops added to design logic for each value **Values:** 4, ..., 255 **Default Value:** 4 **Enabled:** SSI_HAS_RX_SAMPLE_DELAY==1 **Parameter Name:** SSI_RX_DLY_SR_DEPTH
Peripheral ID code	Individual peripheral Identification code. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0xffffffff **Enabled:** Always **Parameter Name:** SSI_ID
Include Endian conversion for XIP and data register Reads?	When enabled user can choose whether a sideband signal or register programming is used to select the endianness for XIP and data register reads. ■ 0 - No ■ 1 - Use sideband signal endconv_en ■ 2 - Programming SECONV field in CTRLR0 register **Values:** ■ No (0) ■ Use sideband signal (1) ■ Use register programming (2) **Default Value:** No **Enabled:** SSI_IS_MASTER==1 **Parameter Name:** SSI_INC_ENDCONV

Table 3-1   Top Level Parameters (Continued)

Label	Description
External Configuration	
Include DMA handshaking interface signals?	Configures the inclusion of DMA handshaking interface signals. Check the box to include the DMA interface signals. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** SSI_HAS_DMA
Configure interrupt pinout	Selects which interrupt related pins appear as outputs of the design. The user has a choice of either a single combined interrupt (the logical OR of all DW_apb_ssi interrupt outputs) or have each individual interrupt appear as a separate output pin on the component. ■ When configured as a master there are 6 individual interrupts. ■ When configured as a slave there are 5 individual interrupts. **Values:** ■ Individual Interrupts (0) ■ Combined Interrupt (1) **Default Value:** Individual Interrupts **Enabled:** Always **Parameter Name:** SSI_INTR_IO
Active interrupt level	Configures the active level of the output interrupt lines. **Values:** ■ Active Low (0) ■ Active High (1) **Default Value:** Active Low **Enabled:** Always **Parameter Name:** SSI_INTR_POL

Table 3-1   Top Level Parameters (Continued)

Label	Description
Internal Configuration	
Hard code the frame format?	When set, the frame format (serial protocol) can be fixed so that the user cannot dynamically program it. This setting restricts the use of DW_apb_ssi to be only a single-frame format peripheral. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** SSI_HC_FRF
Default frame format	Selects the frame format that will be available directly after reset. User can configure any of the frame formats to be the default frame format. If the frame format is hardcoded, the default frame format is the only frame format possible. **Values:** ■ Motorola SPI (0x0) ■ TI SSP (0x1) ■ NatSemi Microwire (0x2) **Default Value:** Motorola SPI **Enabled:** Always **Parameter Name:** SSI_DFLT_FRF
Default serial clock polarity	Controls the default state of the clock polarity. Defines the level of the serial clock when in-active (not toggling). Only used when the frame format is Motorola SPI. **Values:** ■ 0 (0x0) ■ 1 (0x1) **Default Value:** 0 **Enabled:** SSI_DFLT_FRF==0 **Parameter Name:** SSI_DFLT_SCPOL
Default serial clock phase	Controls the default state of the clock phase. Only used when the frame format is Motorola SPI. The serial clock phase selects the relationship of the serial clock with the serial data. When 0, data is captured on the first edge of the serial clock. When 1, data is captured on the second edge of the serial clock. **Values:** ■ 0 (0x0) ■ 1 (0x1) **Default Value:** 0 **Enabled:** SSI_DFLT_FRF==0 **Parameter Name:** SSI_DFLT_SCPH

Table 3-1 Top Level Parameters (Continued)

Label	Description
Toggle slave select signal between frames when in SPI mode and SCPH=0?	When operating in SPI mode with clock phase (SCPH) set to 0, this parameter controls the behavior of the slave select line (ss_*_n) between data frames. ■ If this parameter is set to "Yes" and CTRLR0.SSTE is "1" the ss_*_n line will toggle between consecutive data frames, with the serial clock (sclk) being held to its default value while ss_*_n is high. ■ If this parameter is set to "Yes" and CTRLR0.SSTE is "0" the ss_*_n will stay low and sclk will run continuously for the duration of the transfer; ■ If this parameter is set to 0 the ss_*_n will stay low and sclk will run continuously for the duration of the transfer ~~consecutive data frames, with the serial clock (sclk) being held to its default value while ss_*_n is high.~~ ■ ~~If this parameter is set to "Yes" and CTRLR0.SSTE is "0" the ss_*_n will~~ ~~stay low and sclk will run continuously for the duration of the transfer; if this parameter is set to 0 the ss_*_n will stay low and sclk will run continuously for the duration of the transfer~~ **Values:** ■ No (0) ■ Yes (1) **Default Value:** Yes **Enabled:** (SSI_DFLT_FRF==0) || (SSI_HC_FRF==0) **Parameter Name:** SSI_SCPH0_SSTOGGLE
Default Endianess Value	Controls the default state of the endian conversion feature. Defines the reset value of the CTRLR0.SECONV register field. Only used in Master mode. **Values:** ■ Endian Conversion Disabled (0x0) ■ Endian Conversion Enabled (0x1) **Default Value:** Endian Conversion Disabled **Enabled:** SSI_INC_ENDCONV==2 **Parameter Name:** SSI_DFLT_SECONV

3.2 SPI Parameters

Table 3-2   SPI Parameters

Label	Description
Enhanced SPI mode Parameters	
Select SPI mode	Configures whether the core works in Standard or Dual or Quad or Octal SPI Mode. ■ In Dual Mode - width of txd and rxd signals will be 2-bits. ■ In Quad Mode - width of txd and rxd signals will be 4-bits. ■ In Octal Mode - width of txd and rxd signals will be 8-bits. **Values:** ■ Standard SPI Mode (0) ■ SPI Dual Mode (1) ■ SPI Quad Mode (2) ■ SPI Octal Mode (3) **Default Value:** Standard SPI Mode **Enabled:** ((SSI_DFLT_FRF==0) or (SSI_HC_FRF==0)) and (SSI_IS_MASTER==1) and DWC-APB-Advanced-Source source license exists **Parameter Name:** SSI_SPI_MODE
Enable Enhanced SPI I/O mapping	Configures whether user wants to hardcode the I/O Mapping inside the controller. After selecting this option the RXD[1] signal will be used to sample the incoming data during SPI standard mode of operation. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** SSI_SPI_MODE!=0 **Parameter Name:** SSI_IO_MAP_EN
Include Dual transfer rate transfers in SPI frame format?	Configures the device to support Dual Data transmission on both positive and negative edge of sclk_out **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** SSI_SPI_MODE != 0 and SSI_IS_MASTER==1 and DWC-APB-Advanced-Source source license exists **Parameter Name:** SSI_HAS_DDR

Table 3-2   SPI Parameters (Continued)

Label	Description
Include data strobe signal for rxd line?	Configures the device to include data strobe signaling for rxd signal. This signal can be included only when DW_apb_ssi support dual data rate transfers **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** SSI_HAS_DDR ==1 and SSI_IS_MASTER==1 and DWC-APB-Advanced-Source source license exists **Parameter Name:** SSI_HAS_RXDS
Include Data Mask Signal for data transfers in SPI mode?	Selects if data mask signal should be included on SPI interface. Data mask signal, when active, will mask the write data on SPI interface **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** (SSI_SPI_MODE != 0) && (SSI_APBIF_TYPE == 2) **Parameter Name:** SSI_SPI_DM_EN
Include XIP feature in SPI mode?	If selected DW_apb_ssi will be able to perform ~~eXucute~~eXecute In Place (XIP) commands while working in SPI protocol. A new sideband signal (xip_en) will be included as a part of APB interface which can be used to select if slave interface is used to perform register read/write or in XIP mode. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** (SSI_SPI_MODE != 0) && (SSI_APBIF_TYPE != 0) **Parameter Name:** SSI_XIP_EN

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

101

3.3 Clocking Parameters

Table 3-3   Clocking Parameters

Label	Description
Synchronization parameters	
Are pclk and ssi_clk synchronous?	Defines if the pclk is synchronous to the ssi_clk. If they are synchronous then one does not need to retime signals across the clock domains. **Values:** ■ No (0) ■ Yes (1) **Default Value:** Yes **Enabled:** Always **Parameter Name:** SSI_SYNC_CLK
Generate clock enable input for ssi_clk?	When enabled, the ssi_clk_en signal enables data propagation through ssi_clk flip-flops. When disabled, the ssi_clk flip-flops are always enabled. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** SSI_SYNC_CLK==1 **Parameter Name:** SSI_CLK_EN_MODE
pclk to ssi_clk Synchronization Depth	Defines the number of synchronization register stages for signals passing from the ~~DWC_ssi~~DW_apb_ssi Slave clock domain (pclk) to ~~DWC_ssi~~DW_apb_ssi Core Clock domain (ssi_clk). ■ 2: Two-stage synchronization, both stages positive edge. ■ 3: Three-stage synchronization, all stages positive edge. ■ 4: Four-stage synchronization, all stages positive edge. across the clock domains. **Values:** 2, 3, 4 **Default Value:** 2 **Enabled:** Always **Parameter Name:** SSI_P2S_SYNC_DEPTH

Table 3-3   Clocking Parameters (Continued)

Label	Description
ssi_clk to pclk Synchronization Depth	Defines the number of synchronization register stages for signals passing from the ~~DWC_ssi DWC_ssi~~DW_apb_ssi Core Clock domain (ssi_clk) to Slave clock domain (pclk). ■ 2: Two-stage synchronization, both stages positive edge. ■ 3: Three-stage synchronization, all stages positive edge. ■ 4: Four-stage synchronization, all stages positive edge. across the clock domains. **Values:** 2, 3, 4 **Default Value:** 2 **Enabled:** Always **Parameter Name:** SSI_S2P_SYNC_DEPTH

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

103

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 clocks 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: Names of configuration parameters that populate this signal in your configuration.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

105

Validated by: Assertion or de-assertion of signals 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 107
- Serial Interface on page 111
- DMA Interface on page 115
- Slave Interface on page 117
- Master Interface on page 118
- Interrupt Signals on page 119

4.1 APB Slave Interface Signals

```
          pclk -        - pready
       presetn -        - pslverr
          psel -        - prdata
       penable -
        pwrite -
         paddr -
        pwdata -
        xip_en -
         pstrb -
         pprot -
    endconv_en -
```

Table 4-1   APB Slave Interface Signals

Port Name	I/O	Description
pclk	I	APB clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
presetn	I	APB reset signal. Asynchronous APB interface domain reset. This signal resets only the bus interface. The signal is asserted asynchronously, but is deasserted synchronously after the rising edge of pclk. The synchronization must be provided external to this component. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
psel	I	APB peripheral select that lasts for two pclk cycles. When asserted, indicates that the peripheral has been selected for read or write operation. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

107

Table 4-1   APB Slave Interface Signals (Continued)

Port Name	I/O	Description
penable	I	APB enable control. Asserted for a single pclk cycle and used for timing read or write operations. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pwrite	I	APB write control. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** When high, indicates a write access to the peripheral; when low, indicates a read access.
paddr[(APB_ADDR_WIDTH-1):0]	I	APB address bus. Uses lower 8 bits of the address bus for register decode. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
pwdata[(APB_DATA_WIDTH-1):0]	I	APB write data bus. Driven by the bus master (bridge unit) during write cycles. Can be 8, 16, or 32 bits wide. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
xip_en	I	APB side band signal which will be used to distinguish between normal register read-write and XIP transfers. When xip_en is driven to 1 all the transfers will consider to be XIP transfers, in this scenario APB writes are not allowed and READs can be performed. **Exists:** (SSI_XIP_EN==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-1   APB Slave Interface Signals (Continued)

Port Name	I/O	Description
pready	O	Indicates whether a request cycle was accepted. **Exists:** (SSI_APBIF_TYPE!=0) **Synchronous To:** pclk **Registered:** SSI_XIP_EN==1 ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pslverr	O	Flag for the slave error response from APB. **Exists:** (SSI_APBIF_TYPE!=0) **Synchronous To:** pclk **Registered:** SSI_APB3_ERR_RESP_EN==1 ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pstrb[((APB_DATA_WIDTH/8)-1):0]	I	APB4 Write strobe bus. A high on individual bits in the pstrb bus indicate that the corresponding incoming write data byte on APB bus is to be updated in the addressed register. **Exists:** (SSI_APBIF_TYPE==2) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pprot[2:0]	I	APB4 Protection type. This signal is only added for interface consistency. **Exists:** (SSI_APBIF_TYPE==2) **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. Can be 8, 16, or 32 bits wide **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-1   APB Slave Interface Signals (Continued)

Port Name	I/O	Description
endconv_en	I	APB side band signal which will be used to select endian conversion for data register and XIP reads. **Exists:** (SSI_INC_ENDCONV==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.2 Serial Interface Signals

Table 4-2 Serial Interface Signals

Port Name	I/O	Description
rxd[(SSI_SPI_MULTIIO-1):0]	I	Receive data signal. Input data from a serial-master or serial-slave device is received on this line. **Exists:** Always **Synchronous To:** sclk_in **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rxds	I	Read Data strobe signal. In SPI DDR mode, once this SPI_CTRLR0.RXDS_EN bit is set to 1, DW_apb_ssi will use Read data strobe (rxds) to capture read data. **Exists:** (SSI_HAS_RXDS==1) **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ss_in_n	I	Slave select input. When configured as a serial slave, this signal selects the device. When configured as a serial master, this signal can be used to inform the system of master contention on the bus. **Exists:** Always **Synchronous To:** ssi_clk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
ssi_clk	I	Peripheral serial clock signal. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

111

Table 4-2   Serial Interface Signals (Continued)

Port Name	I/O	Description
ssi_clk_en	I	Optional. Enable signal for ssi_clk domain flip flops. When the ssi_clk input is connected to pclk, all flip-flops clocked by the ssi_clk propagates data only when this signal is active, which gives you control of the frequency ratio between pclk and ssi_clk. This signal is used only when SSI_CLK_EN_MODE = 1. **Exists:** (SSI_CLK_EN_MODE==1) **Synchronous To:** ssi_clk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ssi_rst_n	I	Peripheral serial reset signal. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
scan_mode	I	Scan mode. This signal should be asserted - that is, driven to logic 1 during scan testing and should be deasserted - tied to logic 0 - at all other times. **Exists:** (SSI_ENH_CLK_RATIO==1 || SSI_HAS_RXDS==1) **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ssi_sleep	O	SSI enable flag. This signal is asserted when the DW_apb_ssi is enabled. It can be used by the system clock generator/controller module in order to disable the ssi_clk input of the DW_apb_ssi; this reduces power consumption in the system. 0- Not safe to remove ssi_clk 1- Safe to remove ssi_clk **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-2   Serial Interface Signals (Continued)

Port Name	I/O	Description
ssi_busy	O	SSI busy flag. This signal is asserted when the DW_apb_ssi is busy with serial transfer. This signal can be used to know the status of DW_apb_ssi instead of polling for busy bit in SR register. **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
txd[(SSI_SPI_MULTIIO-1):0]	O	Transmit data signal. Output data from the serial master or serial slave is transmitted on this line. **Exists:** Always **Synchronous To:** SSI_ENH_CLK_RATIO==1 ? "sclk_in" : "ssi_clk" **Registered:** SSI_HAS_EXTD_SPI==1 ? Yes : (SSI_ENH_CLK_RATIO==1 ? No : Yes) **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
txd_dm	O	Data mask signal for transmit line (txd). When active data on txd line is masked. This signal can be used to partially update memory locations in SPI device. **Exists:** (SSI_SPI_DM_EN==1) **Synchronous To:** ssi_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
txd_dm_oe_n	O	Output enable signal for data mask. **Exists:** (SSI_SPI_DM_EN==1) **Synchronous To:** ssi_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
ssi_oe_n[(SSI_SPI_MULTIIO-1):0]	O	Output enable signal. Used to control external buffers on serial output lines. **Exists:** Always **Synchronous To:** ssi_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low

Table 4-2   Serial Interface Signals (Continued)

Port Name	I/O	Description
spi_mode[1:0]	O	SPI Frame format indicator signal. Indicates in which mode does the SPI transfer is happening. 00 - Standard SPI mode 01 - SPI Dual Mode 10 - SPI Quad mode 11 - SPI Octal mode This signal can be used for multiplexing the I/O lines for different mode of operations. **Exists:** (SSI_SPI_MODE!=0) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.3   DMA Interface Signals

dma_tx_ack -
dma_rx_ack -
                          - dma_tx_req
                          - dma_rx_req
                          - dma_tx_single
                          - dma_rx_single

Table 4-3   DMA Interface Signals

Port Name	I/O	Description
dma_tx_ack	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:** (SSI_HAS_DMA==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
dma_rx_ack	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:** (SSI_HAS_DMA==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
dma_tx_req	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 SRC_MSIZE field of the CTLx register. **Exists:** (SSI_HAS_DMA==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-3   DMA Interface Signals (Continued)

Port Name	I/O	Description
dma_rx_req	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 DEST_MSIZE field of the CTLx register. **Exists:** (SSI_HAS_DMA==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
dma_tx_single	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:** (SSI_HAS_DMA==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
dma_rx_single	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:** (SSI_HAS_DMA==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4　　　Slave Interface Signals

sclk_in -

Table 4-4　　　Slave Interface Signals

Port Name	I/O	Description
sclk_in	I	Serial bit-rate clock. Generated by the serial bus master and used by the DW_apb_ssi slave to regulate data transfer. Never used to clock any registers; instead, an edge-detector is used in order for everything to run in sclk domain. This signal is asynchronous to the ssi_clk. **Exists:** (SSI_IS_MASTER==0) **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.5 Master Interface Signals

- sclk_out
- ss_x_n (for x = 0; x <= SSI_NUM_SLAVES)

Table 4-5 Master Interface Signals

Port Name	I/O	Description
sclk_out	O	Serial bit-rate clock. Generated by the DW_apb_ssi from ssi_clk. **Exists:** (SSI_IS_MASTER==1) **Synchronous To:** ssi_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
ss_x_n (for x = 0; x <= SSI_NUM_SLAVES)	O	Slave select output. For MWire and SPI, the active polarity is logic 0; for SSP, the active polarity is logic 1.The number of slave select signals depends on the number of configured slaves(SSI_NUM_SLAVES). **Exists:** SSI_NUM_SLAVES-1 >= x **Synchronous To:** ssi_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.6   Interrupt Signals

- ssi_intr (_n)
- ssi_txe_intr (_n)
- ssi_txo_intr (_n)
- ssi_rxf_intr (_n)
- ssi_rxo_intr (_n)
- ssi_rxu_intr (_n)
- ssi_mst_intr (_n)
- ssi_intr_n
- ssi_txe_intr_n
- ssi_txo_intr_n
- ssi_rxf_intr_n
- ssi_rxo_intr_n
- ssi_rxu_intr_n
- ssi_mst_intr_n

Table 4-6   Interrupt Signals

Port Name	I/O	Description
ssi_intr	O	Optional. Combined SSI active high interrupt flag. Logical OR of all individual interrupts. **Exists:** (SSI_INTR_IO==SSI_COMBINED) && (SSI_INTR_POL==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ssi_txe_intr	O	Optional. Transmit FIFO empty active high interrupt. Active when the transmit FIFO is equal to or below the threshold value (TFT). **Exists:** (SSI_INTR_IO==SSI_INDIVIDUAL) && (SSI_INTR_POL==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ssi_txo_intr	O	Optional. Transmit FIFO overflow active high interrupt. Active when the APB attempts to write to a full transmit FIFO. **Exists:** (SSI_INTR_IO==SSI_INDIVIDUAL) && (SSI_INTR_POL==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6   Interrupt Signals (Continued)

Port Name	I/O	Description
ssi_rxf_intr	O	Optional. Receive FIFO full active high interrupt. Active when the receive FIFO is equal to or above the threshold value (RFT) plus 1. **Exists:** (SSI_INTR_IO==SSI_INDIVIDUAL) && (SSI_INTR_POL==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ssi_rxo_intr	O	Optional. Receive FIFO overflow active high interrupt. Active when the receive logic attempts to write into a full receive FIFO. **Exists:** (SSI_INTR_IO==SSI_INDIVIDUAL) && (SSI_INTR_POL==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ssi_rxu_intr	O	Optional. Receive FIFO underflow active high interrupt. Active when the APB attempts to read from an empty receive FIFO. **Exists:** (SSI_INTR_IO==SSI_INDIVIDUAL) && (SSI_INTR_POL==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ssi_mst_intr	O	Optional. Multi-master contention active high interrupt. Informs of possible bus contention. **Exists:** (SSI_INTR_IO==SSI_INDIVIDUAL) && (SSI_IS_MASTER==1) && (SSI_INTR_POL==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
~~ssi_intr(_n)~~ssi_intr_n	O	Optional. Combined SSI interrupt flag. Logical OR of all individual interrupts. **Exists:** (SSI_INTR_IO==~~1~~SSI_COMBINED) && (SSI_INTR_POL==0) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** ~~High when SSI_INTR_POL=1 otherwise~~ Low

Table 4-6   Interrupt Signals (Continued)

Port Name	I/O	Description
~~ssi_txe_intr(_n)~~ssi_txe_intr_n	O	Optional. Transmit FIFO empty interrupt. Active when the transmit FIFO is equal to or below the threshold value (TFT). **Exists:** (SSI_INTR_IO==~~0~~SSI_INDIVIDUAL) && (SSI_INTR_POL==0) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** ~~High when SSI_INTR_POL=1 otherwise~~ Low
~~ssi_txo_intr(_n)~~ssi_txo_intr_n	O	Optional. Transmit FIFO overflow interrupt. Active when the APB attempts to write to a full transmit FIFO. **Exists:** (SSI_INTR_IO==~~0~~SSI_INDIVIDUAL) && (SSI_INTR_POL==0) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** ~~High when SSI_INTR_POL=1 otherwise~~ Low
~~ssi_rxf_intr(_n)~~ssi_rxf_intr_n	O	Optional. Receive FIFO full interrupt. Active when the receive FIFO is equal to or above the threshold value (RFT) plus 1. **Exists:** (SSI_INTR_IO==~~0~~SSI_INDIVIDUAL) && (SSI_INTR_POL==0) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** ~~High when SSI_INTR_POL=1 otherwise~~ Low
~~ssi_rxo_intr(_n)~~ssi_rxo_intr_n	O	Optional. Receive FIFO overflow interrupt. Active when the receive logic attempts to write into a full receive FIFO. **Exists:** (SSI_INTR_IO==~~0~~SSI_INDIVIDUAL) && (SSI_INTR_POL==0) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** ~~High when SSI_INTR_POL=1 otherwise~~ Low
~~ssi_rxu_intr(_n)~~ssi_rxu_intr_n	O	Optional. Receive FIFO underflow interrupt. Active when the APB attempts to read from an empty receive FIFO. **Exists:** (SSI_INTR_IO==~~0~~SSI_INDIVIDUAL) && (SSI_INTR_POL==0) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** ~~High when SSI_INTR_POL=1 otherwise~~ Low

Table 4-6   Interrupt Signals (Continued)

Port Name	I/O	Description
~~ssi_mst_intr(_n)~~ssi_mst_intr_n	O	Optional. Multi-master contention interrupt. Informs of possible bus contention. **Exists:** (SSI_INTR_IO==~~0~~SSI_INDIVIDUAL) && (SSI_IS_MASTER==1) && (SSI_INTR_POL==0) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** ~~High when SSI_INTR_POL=1 otherwise~~ Low

5

Register Descriptions

This chapter details all possible registers in the IP. They are arranged hierarchically into maps and blocks (banks).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.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

123

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 once to this register 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: ssi_memory_map

Address Block	Description
ssi_address_block on page 126	DW_apb_ssi register address block **Exists:** Always

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

125

5.1 ssi_memory_map/ssi_address_block Registers

~~ssi_address_block~~DW_apb_ssi register address block. Follow the link for the register to see a detailed description of the register.

Table 5-5   Registers for Address Block: ssi_memory_map/ssi_address_block

Register	Offset	Description
CTRLR0 on page 129	0x0	~~This register controls the serial data transfer. It is impossible to write to this register when...~~Control Register 0
CTRLR1 on page 139	0x4	~~This register exists only when the DW_apb_ssi is configured as a master device. When the DW_apb_ssi...~~Control Register 1
SSIENR on page 140	0x8	~~This register enables and disables the DW_apb_ssi. Reset Value: 0x0~~SSI Enable Register
MWCR on page 141	0xc	~~This register controls the direction of the data word for the half-duplex Microwire serial protocol....~~Microwire Control Register
SER on page 143	0x10	~~This register is valid only when the DW_apb_ssi is configured as a master device. When the DW_apb_ssi...~~Slave Enable Register
BAUDR on page 145	0x14	~~This register is valid only when the DW_apb_ssi is configured as a master device. When the DW_apb_ssi...~~Baud Rate Select
TXFTLR on page 147	0x18	~~This register controls the threshold value for the transmit FIFO memory. The DW_apb_ssi is enabled...~~Transmit FIFO Threshold Level
RXFTLR on page 149	0x1c	~~This register controls the threshold value for the receive FIFO memory. The DW_apb_ssi is enabled...~~Receive FIFO Threshold Level
TXFLR on page 151	0x20	~~This register contains the number of valid data entries in the transmit FIFO memory. Reset Value:...~~Transmit FIFO Level Register
RXFLR on page 152	0x24	~~This register contains the number of valid data entries in the receive FIFO memory. This register...~~Receive FIFO Level Register
SR on page 153	0x28	~~This is a read-only register used to indicate the current transfer status, FIFO status, and any...~~Status Register
IMR on page 156	0x2c	~~This read/write reigster masks or enables all interrupts generated by the DW_apb_ssi. When the DW_apb_ssi...~~Interrupt Mask Register

Table 5-5   Registers for Address Block: ssi_memory_map/ssi_address_block (Continued)

Register	Offset	Description
ISR on page 158	0x30	~~This register reports the status of the DW_apb_ssi interrupts after they have been masked. Reset...~~Interrupt Status Register
RISR on page 161	0x34	~~This read-only register reports the status of the DW_apb_ssi interrupts prior to masking. Reset...~~Raw Interrupt Status Register
TXOICR on page 164	0x38	Transmit FIFO Overflow Interrupt Clear Register~~. Reset Value: 0x0~~
RXOICR on page 165	0x3c	Receive FIFO Overflow Interrupt Clear Register~~. Reset Value: 0x0~~
RXUICR on page 166	0x40	Receive FIFO Underflow Interrupt Clear Register~~. Reset Value: 0x0~~
MSTICR on page 167	0x44	Multi-Master Interrupt Clear Register~~. Reset Value: 0x0~~
ICR on page 168	0x48	Interrupt Clear Register~~. Reset Value: 0x0~~
DMACR on page 169	0x4c	~~This register is only valid when DW_apb_ssi is configured with a set of DMA Controller interface...~~DMA Control Register
DMATDLR on page 171	0x50	~~This register is only valid when the DW_apb_ssi is configured with a set of DMA interface signals...~~DMA Transmit Data Level
DMARDLR on page 173	0x54	~~This register is only valid when DW_apb_ssi is configured with a set of DMA interface signals (SSI_HAS_DMA...~~DMA Receive Data Level
IDR on page 174	0x58	~~This register contains the peripherals identification code, which is written into the register at...~~Identification Register
SSI_VERSION_ID on page 175	0x5c	~~This read-only register stores the specific DW_apb_ssi component version. Reset Value:...~~coreKit version ID Register
DRx (for x = 0; x <= 35) on page 176	0x60	~~The DW_apb_ssi data register is a 16/32-bit (depending on SSI_MAX_XFER_SIZE) read/write buffer for the...~~Data Register x
RX_SAMPLE_DLY on page 178	0xf0	~~This register is only valid when the DW_apb_ssi is configured with rxd sample delay logic (SSI_HAS_RX_SAMPLE_DELAY==1)....~~RX Sample Delay Register
SPI_CTRLR0 on page 180	0xf4	~~This register is valid only when SSI_SPI_MODE is either set to "Dual" or "Quad" or "Octal" mode. This...~~SPI Control Register

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

127

Table 5-5   Registers for Address Block: ssi_memory_map/ssi_address_block (Continued)

Register	Offset	Description
TXD_DRIVE_EDGE on page 183	0xf8	~~This Register is valid only when SSI_HAS_DDR is equal to 1. This register is used to control the...~~Transmit Drive Edge Register
RSVD on page 184	0xfc	RSVD - Reserved address location~~.~~

5.1.1   CTRLR0

- **Name:** Control Register 0

- **Description:** This register controls the serial data transfer. It is impossible to write to this register when the DW_apb_ssi is enabled. The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

  Reset Value: SSI_CTRLR0_RST

- **Size:** 32 bits

- **Offset:** 0x0

- **Exists:** Always

31:2526	25	24	23	22:21	20:16	15:12	11	10	9:8	7	6	5:4	3:0
RSVD_CTRLR0	SECONV	SSTE	RSVD_CTRLR0_23	SPI_FRF	DFS_32	CFS	SRL	SLV_OE	TMOD	SCPOL	SCPH	FRF	DFS

Table 5-6   Fields for Register: CTRLR0

Bits	Name	Memory Access	Description
~~31:25~~ 31:26	RSVD_CTRLR0	R	SSTE Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
25	SECONV	R/W	Set the Endianness for XIP and data register reads. **Values:** ■ 0x0 (DISABLED): Endian Conversion disabled. ■ 0x1 (ENABLED): Endian Conversion enabled. **Value After Reset:** SSI_DFLT_SECONV **Exists:** SSI_INC_ENDCONV == 2

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
24	SSTE	* Varies	**Slave Select Toggle Enable.** When operating in SPI mode with clock phase (SCPH) set to 0, this register controls the behavior of the slave select line (ss_*_n) between data frames. If this register field is set to 1 the ss_*_n line will toggle between consecutive data frames, with the serial clock (sclk) being held to its default value while ss_*_n is high; if this register field is set to 0 the ss_*_n will stay low and sclk will run continuously for the duration of the transfer. **Note:** This register is only valid when SSI_SCPH0_SSTOGGLE is set to 1. **Value After Reset:** "(SSI_SCPH0_SSTOGGLE==0) ? \"0\": \"1\"" **Exists:** Always **Memory Access:** "(SSI_SCPH0_SSTOGGLE==0) ? \"read-only\": \"read-write\""
23	RSVD_CTRLR0_23	R	CTRLR0_23 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
22:21	SPI_FRF	* Varies	**SPI Frame Format:** Selects data frame format for Transmitting/Receiving the data Bits only valid when SSI_SPI_MODE is either set to "Dual" or "Quad" or "Octal" mode. When SSI_SPI_MODE is configured for "Dual Mode", 10/11 combination is reserved. When SSI_SPI_MODE is configured for "Quad Mode", 11 combination is reserved. **Values:** ■ 0x0 (STD_SPI_FRF): Standard SPI Frame Format ■ 0x1 (DUAL_SPI_FRF): Dual SPI Frame Format ■ 0x2 (QUAD_SPI_FRF): Quad SPI Frame Format ■ 0x3 (OCTAL_SPI_FRF): Octal SPI Frame Format **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** "(SSI_SPI_MODE==0) ? \"read-only\": \"read-write\""

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
20:16	DFS_32	* Varies	**Data Frame Size** in 32-bit transfer size mode. Used to select the data frame size in 32-bit transfer mode. These bits are only valid when SSI_MAX_XFER_SIZE is configured to 32. When the data frame size is programmed to be less than 32 bits, the receive data are automatically right-justified by the receive logic, with the upper bits of the receive FIFO zero-padded. You are responsible for making sure that transmit data is right-justified before writing into the transmit FIFO. The transmit logic ignores the upper unused bits when transmitting the data. **Note:** When SSI_SPI_MODE is either set to "Dual" or "Quad" or "Octal" mode and SPI_FRF is not set to 2'b00. ■ DFS value should be multiple of 2 if SPI_FRF = 0x01, ■ DFS value should be multiple of 4 if SPI_FRF = 0x10, ■ DFS value should be multiple of 8 if SPI_FRF = 0x11. ~~DFS value should be multiple of 2 if SPI_FRF = 0x01,~~ ~~DFS value should be multiple of 4 if SPI_FRF = 0x10,~~ ~~DFS value should be multiple of 8 if SPI_FRF = 0x11.~~ **Values:** ■ 0x3 (FRAME_04BITS): 4-bit serial data transfer ■ 0x4 (FRAME_05BITS): 5-bit serial data transfer ■ 0x5 (FRAME_06BITS): 6-bit serial data transfer ■ 0x6 (FRAME_07BITS): 7-bit serial data transfer ■ 0x7 (FRAME_08BITS): 8-bit serial data transfer ■ 0x8 (FRAME_09BITS): 9-bit serial data transfer ■ 0x9 (FRAME_10BITS): 10-bit serial data transfer ■ 0xa (FRAME_11BITS): 11-bit serial data transfer ■ 0xb (FRAME_12BITS): 12-bit serial data transfer ■ 0xc (FRAME_13BITS): 13-bit serial data transfer

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

131

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
			■ 0xd (FRAME_14BITS): 14-bit serial data transfer ■ 0xe (FRAME_15BITS): 15-bit serial data transfer ■ 0xf (FRAME_16BITS): 16-bit serial data transfer ■ 0x10 (FRAME_17BITS): 17-bit serial data transfer ■ 0x11 (FRAME_18BITS): 18-bit serial data transfer ■ 0x12 (FRAME_19BITS): 19-bit serial data transfer ■ 0x13 (FRAME_20BITS): 20-bit serial data transfer ■ 0x14 (FRAME_21BITS): 21-bit serial data transfer ■ 0x15 (FRAME_22BITS): 22-bit serial data transfer ■ 0x16 (FRAME_23BITS): 23-bit serial data transfer ■ 0x17 (FRAME_24BITS): 24-bit serial data transfer ■ 0x18 (FRAME_25BITS): 25-bit serial data transfer ■ 0x19 (FRAME_26BITS): 26-bit serial data transfer ■ 0x1a (FRAME_27BITS): 27-bit serial data transfer ■ 0x1b (FRAME_28BITS): 28-bit serial data transfer ■ 0x1c (FRAME_29BITS): 29-bit serial data transfer ■ 0x1d (FRAME_30BITS): 30-bit serial data transfer ■ 0x1e (FRAME_31BITS): 31-bit serial data transfer ■ 0x1f (FRAME_32BITS): 32-bit serial data transfer **Value After Reset:** "(SSI_MAX_XFER_SIZE==32) ? \"0x7\" : \"0x0\"" **Exists:** Always **Memory Access:** "(SSI_MAX_XFER_SIZE==32) ? \"read-write\" : \"read-only\""

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
15:12	CFS	R/W	Control Frame Size. Selects the length of the control word for the Microwire frame format. **Values:** ■ 0x0 (SIZE_01_BIT): 1-bit Control Word ■ 0x1 (SIZE_02_BIT): 2-bit Control Word ■ 0x2 (SIZE_03_BIT): 3-bit Control Word ■ 0x3 (SIZE_04_BIT): 4-bit Control Word ■ 0x4 (SIZE_05_BIT): 5-bit Control Word ■ 0x5 (SIZE_06_BIT): 6-bit Control Word ■ 0x6 (SIZE_07_BIT): 7-bit Control Word ■ 0x7 (SIZE_08_BIT): 8-bit Control Word ■ 0x8 (SIZE_09_BIT): 9-bit Control Word ■ 0x9 (SIZE_10_BIT): 10-bit Control Word ■ 0xa (SIZE_11_BIT): 11-bit Control Word ■ 0xb (SIZE_12_BIT): 12-bit Control Word ■ 0xc (SIZE_13_BIT): 13-bit Control Word ■ 0xd (SIZE_14_BIT): 14-bit Control Word ■ 0xe (SIZE_15_BIT): 15-bit Control Word ■ 0xf (SIZE_16_BIT): 16-bit Control Word **Value After Reset:** 0x0 **Exists:** Always
11	SRL	R/W	**Shift Register Loop.** Used for testing purposes only. When internally active, connects the transmit shift register output to the receive shift register input. Can be used in both serial-slave and serial-master modes. When the DW_apb_ssi is configured as a slave in loopback mode, the ss_in_n and ssi_clk signals must be provided by an external source. In this mode, the slave cannot generate these signals because there is nothing to which to loop back **Values:** ■ 0x1 (TESTING_MODE): Test mode: Tx & Rx shift reg connected ■ 0x0 (NORMAL_MODE): Normal mode operation **Value After Reset:** 0x0 **Exists:** Always

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
10	SLV_OE	~~R/W~~* Varies	Slave Output Enable. Relevant only when the DW_apb_ssi is configured as a serial-slave device. When configured as a serial master, this bit field has no functionality. This bit enables or disables the setting of the ssi_oe_n output from the DW_apb_ssi serial slave. When SLV_OE = 1, the ssi_oe_n output can never be active. When the ssi_oe_n output controls the tri-state buffer on the txd output from the slave, a high impedance state is always present on the slave txd output when SLV_OE = 1. This is useful when the master transmits in broadcast mode (master transmits data to all slave devices). Only one slave may respond with data on the master rxd line. This bit is enabled after reset and must be disabled by software (when broadcast mode is used), if you do not want this device to respond with data. **Values:** ■ 0x1 (DISABLED): Slave Output is disabled ■ 0x0 (ENABLED): Slave Output is enabled **Value After Reset:** 0x0 **Exists:** Always ~~**Exists**~~**Memory Access:** "(SSI_IS_MASTER==~~0~~0) ? \"read-write\" : \"read-only\""

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
9:8	TMOD	R/W	**Transfer Mode.** Selects the mode of transfer for serial communication. This field does not affect the transfer duplicity. Only indicates whether the receive or transmit data are valid. In transmit-only mode, data received from the external device is not valid and is not stored in the receive FIFO memory; it is overwritten on the next transfer. In receive-only mode, transmitted data are not valid. After the first write to the transmit FIFO, the same word is retransmitted for the duration of the transfer. In transmit-and-receive mode, both transmit and receive data are valid. The transfer continues until the transmit FIFO is empty. Data received from the external device are stored into the receive FIFO memory, where it can be accessed by the host processor. In eeprom-read mode, receive data is not valid while control data is being transmitted. When all control data is sent to the EEPROM, receive data becomes valid and transmit data becomes invalid. All data in the transmit FIFO is considered control data in this mode. This transfer mode is only valid when the DW_apb_ssi is configured as master device. 00 - Transmit & Receive 01 - Transmit Only 10 - Receive Only 11 - EEPROM Read When SSI_SPI_MODE is either set to "Dual" or "Quad" or "Octal" mode and SPI_FRF is not set to 2'b00. There are only two valid combinations: 10 - Read 01 - Write **Values:** ■ 0x0 (TX_AND_RX): Transmit & receive ■ 0x1 (TX_ONLY): Transmit only mode or Write (SPI_FRF != 2'b00) ■ 0x2 (RX_ONLY): Receive only mode or Read (SPI_FRF != 2'b00) ■ 0x3 (EEPROM_READ): EEPROM Read mode **Value After Reset:** 0x0 **Exists:** Always

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

135

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
7	SCPOL	* Varies	**Serial Clock Polarity.** Valid when the frame format (FRF) is set to Motorola SPI. Used to select the polarity of the inactive serial clock, which is held inactive when the DW_apb_ssi master is not actively transferring data on the serial bus. **Values:** ■ 0x0 (SCLK_LOW): Inactive state of serial clock is low ■ 0x1 (SCLK_HIGH): Inactive state of serial clock is high **Value After Reset:** SSI_DFLT_SCPOL **Exists:** Always **Memory Access:** "(SSI_HC_FRF==0) ? \"read-write\" : \"read-only\""
6	SCPH	* Varies	**Serial Clock Phase.** Valid when the frame format (FRF) is set to Motorola SPI. The serial clock phase selects the relationship of the serial clock with the slave select signal. When SCPH = 0, data are captured on the first edge of the serial clock. When SCPH = 1, the serial clock starts toggling one cycle after the slave select line is activated, and data are captured on the second edge of the serial clock. **Values:** ■ 0x0 (SCPH_MIDDLE): Serial clock toggles in middle of first data bit ■ 0x1 (SCPH_START): Serial clock toggles at start of first data bit **Value After Reset:** SSI_DFLT_SCPH **Exists:** Always **Memory Access:** "(SSI_HC_FRF==0) ? \"read-write\" : \"read-only\""

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
5:4	FRF	* Varies	**Frame Format.**Selects which serial protocol transfers the data. **Values:** ■ 0x0 (MOTOROLA_SPI): ~~Motorolla~~Motorola SPI Frame Format ■ 0x1 (TEXAS_SSP): Texas Instruments SSP Frame Format ■ 0x2 (NS_MICROWIRE): National Microwire Frame Format ■ 0x3 (RESERVED): Reserved value **Value After Reset:** SSI_DFLT_FRF **Exists:** Always **Memory Access:** "(SSI_HC_FRF==0) ? \"read-write\" : \"read-only\""

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

137

Table 5-6   Fields for Register: CTRLR0 (Continued)

Bits	Name	Memory Access	Description
3:0	DFS	* Varies	**Data Frame Size.** This register field is only valid when SSI_MAX_XFER_SIZE is configured to 16. If SSI_MAX_XFER_SIZE is configured to 32, then writing to this field will not have any effect. Selects the data frame length. When the data frame size is programmed to be less than 16 bits, the receive data are automatically right-justified by the receive logic, with the upper bits of the receive FIFO zero-padded. You must right-justify transmit data before writing into the transmit FIFO. The transmit logic ignores the upper unused bits when transmitting the data. **Note:** When SSI_SPI_MODE is either set to "Dual" or "Quad" or "Octal" mode and SPI_FRF is not set to 2'b00. - DFS value should be multiple of 2 if SPI_FRF = 01, - DFS value should be multiple of 4 if SPI_FRF = 10, - DFS value should be multiple of 8 if SPI_FRF = 11. **Values:** ■ 0x3 (FRAME_04BITS): 4-bit serial data transfer ■ 0x4 (FRAME_05BITS): 5-bit serial data transfer ■ 0x5 (FRAME_06BITS): 6-bit serial data transfer ■ 0x6 (FRAME_07BITS): 7-bit serial data transfer ■ 0x7 (FRAME_08BITS): 8-bit serial data transfer ■ 0x8 (FRAME_09BITS): 9-bit serial data transfer ■ 0x9 (FRAME_10BITS): 10-bit serial data transfer ■ 0xa (FRAME_11BITS): 11-bit serial data transfer ■ 0xb (FRAME_12BITS): 12-bit serial data transfer ■ 0xc (FRAME_13BITS): 13-bit serial data transfer ■ 0xd (FRAME_14BITS): 14-bit serial data transfer ■ 0xe (FRAME_15BITS): 15-bit serial data transfer ■ 0xf (FRAME_16BITS): 16-bit serial data transfer **Value After Reset:** "(SSI_MAX_XFER_SIZE==16) ? \"0x7\" : \"0x0\"" **Exists:** Always **Memory Access:** "(SSI_MAX_XFER_SIZE==16) ? \"read-write\" : \"read-only\""

5.1.2   CTRLR1

- **Name:** Control Register 1

- **Description:** This register exists only when the DW_apb_ssi is configured as a master device. When the DW_apb_ssi is configured as a serial slave, writing to this location has no effect; reading from this location returns 0. Control register 1 controls the end of serial transfers when in receive-only mode. It is impossible to write to this register when the DW_apb_ssi is enabled. The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

  Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x4

- **Exists:** SSI_IS_MASTER == 1

Table 5-7   Fields for Register: CTRLR1

Bits	Name	Memory Access	Description
31:16	RSVD_CTRLR1	R	CTRLR1 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
15:0	NDF	R/W	**Number of Data Frames.** When TMOD = 10 or TMOD = 11 , this register field sets the number of data frames to be continuously received by the DW_apb_ssi. The DW_apb_ssi continues to receive serial data until the number of data frames received is equal to this register value plus 1, which enables you to receive up to 64 KB of data in a continuous transfer. When the DW_apb_ssi is configured as a serial slave, the transfer continues for as long as the slave is selected. Therefore, this register serves no purpose and is not present when the DW_apb_ssi is configured as a serial slave. **Value After Reset:** 0x0 **Exists:** Always

5.1.3 SSIENR

- **Name:** SSI Enable Register
- **Description:** This register enables and disables the DW_apb_ssi.

  Reset Value: 0x0
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always

Table 5-8   Fields for Register: SSIENR

Bits	Name	Memory Access	Description
31:1	RSVD_SSIENR	R	SSIENR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	SSI_EN	R/W	**SSI Enable.** Enables and disables all DW_apb_ssi operations. When disabled, all serial transfers are halted immediately. Transmit and receive FIFO buffers are cleared when the device is disabled. It is impossible to program some of the DW_apb_ssi control registers when enabled. When disabled, the ssi_sleep output is set (after delay) to inform the system that it is safe to remove the ssi_clk, thus saving power consumption in the system. **Values:** ■ 0x0 (DISABLE): Disables Serial Transfer ■ 0x1 (ENABLED): Enables Serial Transfer **Value After Reset:** 0x0 **Exists:** Always

5.1.4 MWCR

- **Name:** Microwire Control Register

- **Description:** This register controls the direction of the data word for the half-duplex Microwire serial protocol. It is impossible to write to this register when the DW_apb_ssi is enabled. The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

  Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0xc

- **Exists:** Always

Table 5-9   Fields for Register: MWCR

Bits	Name	Memory Access	Description
31:3	RSVD_MWCR	R	MWCR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
2	MHS	R/W	**Microwire Handshaking.** Relevant only when the DW_apb_ssi is configured as a serial-master device. When configured as a serial slave, this bit field has no functionality. Used to enable and disable the busy/ready handshaking interface for the Microwire protocol. When enabled, the DW_apb_ssi checks for a ready status from the target slave, after the transfer of the last data/control bit, before clearing the BUSY status in the SR register. **Values:** ■ 0x0 (DISABLE): Handshaking interface is disabled ■ 0x1 (ENABLED): Handshaking interface is enabled **Value After Reset:** 0x0 **Exists:** SSI_IS_MASTER == 1

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

141

Table 5-9   Fields for Register: MWCR (Continued)

Bits	Name	Memory Access	Description
1	MDD	R/W	**Microwire Control.** Defines the direction of the data word when the Microwire serial protocol is used. When this bit is set to 0, the data word is received by the DW_apb_ssi MacroCell from the external serial device. When this bit is set to 1, the data word is transmitted from the DW_apb_ssi MacroCell to the external serial device. **Values:** ■ 0x0 (RECEIVE): SSI receives data ■ 0x1 (TRANSMIT): SSI transmits data **Value After Reset:** 0x0 **Exists:** Always
0	MWMOD	R/W	**Microwire Transfer Mode.** Defines whether the Microwire transfer is sequential or non-sequential. When sequential mode is used, only one control word is needed to transmit or receive a block of data words. When non-sequential mode is used, there must be a control word for each data word that is transmitted or received. **Values:** ■ 0x0 (NON_SEQUENTIAL): Non-Sequential Microwire Transfer ■ 0x1 (SEQUENTIAL): Sequential Microwire Transfer **Value After Reset:** 0x0 **Exists:** Always

5.1.5   SER

- **Name:** Slave Enable Register

- **Description:** This register is valid only when the DW_apb_ssi is configured as a master device. When the DW_apb_ssi is configured as a serial slave, writing to this location has no effect; reading from this location returns 0. The register enables the individual slave select output lines from the DW_apb_ssi master. Up to 16 slave-select output pins are available on the DW_apb_ssi master. Register bits can be set or cleared when SSI_EN=0.

    If SSI_EN=1, then register bits can be set (to delay the slave select assertion while TX FIFO is getting filled) but cannot be cleared.

    Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x10

- **Exists:** SSI_IS_MASTER == 1

Table 5-10   Fields for Register: SER

Bits	Name	Memory Access	Description
31:y	RSVD_SER	R	SER Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** SSI_NUM_SLAVES

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

143

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

Bits	Name	Memory Access	Description
x:0	SER	* Varies	**Slave Select Enable Flag.** Each bit in this register corresponds to a slave select line (ss_x_n) from the DW_apb_ssi master. When a bit in this register is set (1), the corresponding slave select line from the master is activated when a serial transfer begins. It should be noted that setting or clearing bits in this register have no effect on the corresponding slave select outputs until a transfer is started. Before beginning a transfer, you should enable the bit in this register that corresponds to the slave device with which the master wants to communicate. When not operating in broadcast mode, only one bit in this field should be set. **Values:** ■ 0x0 (NOT_SELECTED): No slave selected ■ 0x1 (SELECTED): Slave is selected **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** SSI_NUM_SLAVES - 1 **Memory Access:** "(SSI_IS_MASTER==1) ? \"read-write\" : \"read-only\""

5.1.6   BAUDR

- **Name:** Baud Rate Select
- **Description:** This register is valid only when the DW_apb_ssi is configured as a master device. When the DW_apb_ssi is configured as a serial slave, writing to this location has no effect; reading from this location returns 0. The register derives the frequency of the serial clock that regulates the data transfer. The 16-bit field in this register defines the ssi_clk divider value. It is impossible to write to this register when the DW_apb_ssi is enabled. The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

    Reset Value: 0x0

- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** SSI_IS_MASTER == 1

| RSVD_BAUDR 31:16 | SCKDV 15:0 |

Table 5-11   Fields for Register: BAUDR

Bits	Name	Memory Access	Description
31:16	RSVD_BAUDR	R	BAUDR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

145

Table 5-11   Fields for Register: BAUDR (Continued)

Bits	Name	Memory Access	Description
15:0	SCKDV	R/W	**SSI Clock Divider.** The LSB for this field is always set to 0 and is unaffected by a write operation, which ensures an even value is held in this register. If the value is 0, the serial output clock (sclk_out) is disabled. The frequency of the sclk_out is derived from the following equation: Fsclk_out = Fssi_clk/SCKDV where SCKDV is any even value between 2 and 65534. For example: ~~for~~For Fssi_clk = 3.6864MHz and SCKDV =2 Fsclk_out = 3.6864/2 = 1.8432MHz **Value After Reset:** 0x0 **Exists:** Always

146

SolvNetPlus
DesignWare

Synopsys, Inc.

4.03a
December 2020

5.1.7 TXFTLR

- **Name:** Transmit FIFO Threshold Level

- **Description:** This register controls the threshold value for the transmit FIFO memory. The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

    Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x18

- **Exists:** Always

Table 5-12   Fields for Register: TXFTLR

Bits	Name	Memory Access	Description
31:y	RSVD_TXFTLR	R	TXFTLR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** TX_ABW

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

147

Table 5-12   Fields for Register: TXFTLR (Continued)

Bits	Name	Memory Access	Description
x:0	TFT	R/W	**Transmit FIFO Threshold.** Controls the level of entries (or below) at which the transmit FIFO controller triggers an interrupt. The FIFO depth is configurable in the range 2-256; this register is sized to the number of address bits needed to access the FIFO. If you attempt to set this value greater than or equal to the depth of the FIFO, this field is not written and retains its current value. When the number of transmit FIFO entries is less than or equal to this value, the transmit FIFO empty interrupt is triggered. For information on the Transmit FIFO Threshold values, see the "Master SPI and SSP Serial Transfers" in the DW_apb_ssi Databook. ssi_txe_intr is asserted when TFT or less data entries are present in transmit FIFO **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** TX_ABW - 1

5.1.8   RXFTLR

- **Name:** Receive FIFO Threshold Level

- **Description:** This register controls the threshold value for the receive FIFO memory. The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

    Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x1c

- **Exists:** Always

Table 5-13   Fields for Register: RXFTLR

Bits	Name	Memory Access	Description
31:y	RSVD_RXFTLR	R	RXFTLR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** RX_ABW

Table 5-13   Fields for Register: RXFTLR (Continued)

Bits	Name	Memory Access	Description
x:0	RFT	R/W	**Receive FIFO Threshold.** Controls the level of entries (or above) at which the receive FIFO controller triggers an interrupt. The FIFO depth is configurable in the range 2-256. This register is sized to the number of address bits needed to access the FIFO. If you attempt to set this value greater than the depth of the FIFO, this field is not written and retains its current value. When the number of receive FIFO entries is greater than or equal to this value + 1, the receive FIFO full interrupt is triggered. For information on the Receive FIFO Threshold values, see the "Master SPI and SSP Serial Transfers" in the DW_apb_ssi Databook. ssi_rxf_intr is asserted when RFT or more data entries are present in receive FIFO. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** RX_ABW - 1

5.1.9   TXFLR

- ■ **Name:** Transmit FIFO Level Register
- ■ **Description:** This register contains the number of valid data entries in the transmit FIFO memory.

  Reset Value: 0x0
- ■ **Size:** 32 bits
- ■ **Offset:** 0x20
- ■ **Exists:** Always

RSVD_TXFLR	31:y
TXTFL	x:0

Table 5-14   Fields for Register: TXFLR

Bits	Name	Memory Access	Description
31:y	RSVD_TXFLR	R	TXFLR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** TX_ABW + 1
x:0	TXTFL	R	**Transmit FIFO Level.** Contains the number of valid data entries in the transmit FIFO. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** TX_ABW

5.1.10 RXFLR

- **Name:** Receive FIFO Level Register

- **Description:** This register contains the number of valid data entries in the receive FIFO memory. This register can be ready at any time.

  Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x24

- **Exists:** Always

Table 5-15 Fields for Register: RXFLR

Bits	Name	Memory Access	Description
31:y	RSVD_RXFLR	R	RXFLR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** RX_ABW + 1
x:0	RXTFL	R	**Receive FIFO Level.** Contains the number of valid data entries in the receive FIFO. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** RX_ABW

5.1.11 SR

- **Name:** Status Register

- **Description:** This is a read-only register used to indicate the current transfer status, FIFO status, and any transmission/reception errors that may have occurred. The status register may be read at any time. None of the bits in this register request an interrupt.

  Reset Value: 0x6

- **Size:** 32 bits

- **Offset:** 0x28

- **Exists:** Always

Table 5-16   Fields for Register: SR

Bits	Name	Memory Access	Description
31:7	RSVD_SR	R	SR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
6	DCOL	R	**Data Collision Error.** Relevant only when the DW_apb_ssi is configured as a master device. This bit will be set if ss_in_n input is asserted by other master, when the DW_apb_ssi master is in the middle of the transfer. This informs the processor that the last data transfer was halted before completion. This bit is cleared when read. **Values:** ■ 0x0 (NO_ERROR_CONDITION): No Error ■ 0x1 (TX_COLLISION_ERROR): Transmit Data Collision Error **Value After Reset:** 0x0 **Exists:** SSI_IS_MASTER == 1 **Volatile:** true

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

153

Table 5-16   Fields for Register: SR (Continued)

Bits	Name	Memory Access	Description
5	TXE	R	**Transmission Error.** Set if the transmit FIFO is empty when a transfer is started. This bit can be set only when the DW_apb_ssi is configured as a slave device. Data from the previous transmission is resent on the txd line. This bit is cleared when read. **Values:** ■ 0x0 (NO_ERROR): No Error ■ 0x1 (TX_ERROR): Transmission Error **Value After Reset:** 0x0 **Exists:** SSI_IS_MASTER == 0 **Volatile:** true
4	RFF	R	**Receive FIFO Full.** When the receive FIFO is completely full, this bit is set. When the receive FIFO contains one or more empty location, this bit is cleared. **Values:** ■ 0x0 (NOT_FULL): Receive FIFO is not full ■ 0x1 (FULL): Receive FIFO is full **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
3	RFNE	R	**Receive FIFO Not Empty.** Set when the receive FIFO contains one or more entries and is cleared when the receive FIFO is empty. This bit can be polled by software to completely empty the receive FIFO. **Values:** ■ 0x0 (EMPTY): Receive FIFO is empty ■ 0x1 (NOT_EMPTY): Receive FIFO is not empty **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-16   Fields for Register: SR (Continued)

Bits	Name	Memory Access	Description
2	TFE	R	**Transmit FIFO Empty.** When the transmit FIFO is completely empty, this bit is set. When the transmit FIFO contains one or more valid entries, this bit is cleared. This bit field does not request an interrupt. **Values:** ■ 0x0 (NOT_EMPTY): Transmit FIFO is not empty ■ 0x1 (EMPTY): Transmit FIFO is empty **Value After Reset:** 0x1 **Exists:** Always **Volatile:** true
1	TFNF	R	**Transmit FIFO Not Full.** Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full. **Values:** ■ 0x0 (FULL): Transmit FIFO is full ■ 0x1 (NOT_FULL): Transmit FIFO is not Full **Value After Reset:** 0x1 **Exists:** Always **Volatile:** true
0	BUSY	R	**SSI Busy Flag.** When set, indicates that a serial transfer is in progress; when cleared indicates that the DW_apb_ssi is idle or disabled. **Values:** ■ 0x0 (INACTIVE): DW_apb_ssi is idle or disabled ■ 0x1 (ACTIVE): DW_apb_ssi is actively transferring data **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.12 IMR

- **Name:** Interrupt Mask Register
- **Description:** This read/write ~~reigster~~register masks or enables all interrupts generated by the DW_apb_ssi. When the DW_apb_ssi is configured as a slave device, the MSTIM bit field is not present. This changes the reset value from 0x3F for serial-master configurations to 0x1F for serial-slave configurations.

  Reset Value: (SSI_IS_MASTER == 1) ? 0x3F : 0x1F

- **Size:** 32 bits
- **Offset:** 0x2c
- **Exists:** Always

RSVD_IMR	MSTIM	RXFIM	RXOIM	RXUIM	TXOIM	TXEIM
31:6	5	4	3	2	1	0

Table 5-17   Fields for Register: IMR

Bits	Name	Memory Access	Description
31:6	RSVD_IMR	R	IMR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
5	MSTIM	* Varies	Multi-Master Contention Interrupt Mask. This bit field is not present if the DW_apb_ssi is configured as a serial-slave device. **Values:** ■ 0x0 (MASKED): ssi_mst_intr interrupt is masked ■ 0x1 (UNMASKED): ssi_mst_intr interrupt is not masked **Value After Reset:** "(SSI_IS_MASTER==1) ? \"1\" : \"0\"" **Exists:** SSI_IS_MASTER == 1 **Memory Access:** "(SSI_IS_MASTER==1) ? \"read-write\" : \"read-only\""

Table 5-17   Fields for Register: IMR (Continued)

Bits	Name	Memory Access	Description
4	RXFIM	R/W	Receive FIFO Full Interrupt Mask **Values:** ■ 0x0 (MASKED): ssi_rxf_intr interrupt is masked ■ 0x1 (UNMASKED): ssi_rxf_intr interrupt is not masked **Value After Reset:** 0x1 **Exists:** Always
3	RXOIM	R/W	Receive FIFO Overflow Interrupt Mask **Values:** ■ 0x0 (MASKED): ssi_rxo_intr interrupt is masked ■ 0x1 (UNMASKED): ssi_rxo_intr interrupt is not masked **Value After Reset:** 0x1 **Exists:** Always
2	RXUIM	R/W	Receive FIFO Underflow Interrupt Mask **Values:** ■ 0x0 (MASKED): ssi_rxu_intr interrupt is masked ■ 0x1 (UNMASKED): ssi_rxu_intr interrupt is not masked **Value After Reset:** 0x1 **Exists:** Always
1	TXOIM	R/W	Transmit FIFO Overflow Interrupt Mask **Values:** ■ 0x0 (MASKED): ssi_txo_intr interrupt is masked ■ 0x1 (UNMASKED): ssi_txo_intr interrupt is not masked **Value After Reset:** 0x1 **Exists:** Always
0	TXEIM	R/W	Transmit FIFO Empty Interrupt Mask **Values:** ■ 0x0 (MASKED): ssi_txe_intr interrupt is masked ■ 0x1 (UNMASKED): ssi_txe_intr interrupt is not masked **Value After Reset:** 0x1 **Exists:** Always

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

157

5.1.13   ISR

- **Name:** Interrupt Status Register

- **Description:** This register reports the status of the DW_apb_ssi interrupts after they have been masked.

    Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x30

- **Exists:** Always

31:6	5	4	3	2	1	0
RSVD_ISR	MSTIS	RXFIS	RXOIS	RXUIS	TXOIS	TXEIS

Table 5-18   Fields for Register: ISR

Bits	Name	Memory Access	Description
31:6	RSVD_ISR	R	ISR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
5	MSTIS	R	Multi-Master Contention Interrupt Status. This bit field is not present if the DW_apb_ssi is configured as a serial-slave device. **Values:** ■ 0x0 (INACTIVE): ssi_mst_intr interrupt not active after masking ■ 0x1 (ACTIVE): ssi_mst_intr interrupt is active after masking **Value After Reset:** 0x0 **Exists:** SSI_IS_MASTER == 1 **Volatile:** true

Table 5-18   Fields for Register: ISR (Continued)

Bits	Name	Memory Access	Description
4	RXFIS	R	Receive FIFO Full Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_rxf_intr interrupt is not active after masking ■ 0x1 (ACTIVE): ssi_rxf_intr interrupt is full after masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
3	RXOIS	R	Receive FIFO Overflow Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_rxo_intr interrupt is not active after masking ■ 0x1 (ACTIVE): ssi_rxo_intr interrupt is active after masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
2	RXUIS	R	Receive FIFO Underflow Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_rxu_intr interrupt is not active after masking ■ 0x1 (ACTIVE): ssi_rxu_intr interrupt is active after masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
1	TXOIS	R	Transmit FIFO Overflow Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_txo_intr interrupt is not active after masking ■ 0x1 (ACTIVE): ssi_txo_intr interrupt is active after masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

159

Table 5-18   Fields for Register: ISR (Continued)

Bits	Name	Memory Access	Description
0	TXEIS	R	Transmit FIFO Empty Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_txe_intr interrupt is not active after masking ■ 0x1 (ACTIVE): ssi_txe_intr interrupt is active after masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.14   RISR

- **Name:** Raw Interrupt Status Register

- **Description:** This read-only register reports the status of the DW_apb_ssi interrupts prior to masking.

    Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x34

- **Exists:** Always

Table 5-19   Fields for Register: RISR

Bits	Name	Memory Access	Description
31:6	RSVD_RISR	R	RISR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
5	MSTIR	R	Multi-Master Contention Raw Interrupt Status. This bit field is not present if the DW_apb_ssi is configured as a serial-slave device. **Values:** ■ 0x0 (INACTIVE): ssi_mst_intr interrupt is not active prior to masking ■ 0x1 (ACTIVE): ssi_mst_intr interrupt is active prior masking **Value After Reset:** 0x0 **Exists:** SSI_IS_MASTER == 1 **Volatile:** true

Table 5-19   Fields for Register: RISR (Continued)

Bits	Name	Memory Access	Description
4	RXFIR	R	Receive FIFO Full Raw Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_rxf_intr interrupt is not active prior to masking ■ 0x1 (ACTIVE): ssi_rxf_intr interrupt is active prior to masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
3	RXOIR	R	Receive FIFO Overflow Raw Interrupt Status **Values:** ■ 0x1 (ACTIVE): ssi_rxo_intr interrupt is not active prior to masking ■ 0x0 (INACTIVE): ssi_rxo_intr interrupt is active prior masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
2	RXUIR	R	Receive FIFO Underflow Raw Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_rxu_intr interrupt is not active prior to masking ■ 0x1 (ACTIVE): ssi_rxu_intr interrupt is active prior to masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
1	TXOIR	R	Transmit FIFO Overflow Raw Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_txo_intr interrupt is not active prior to masking ■ 0x1 (ACTIVE): ssi_txo_intr interrupt is active prior masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-19   Fields for Register: RISR (Continued)

Bits	Name	Memory Access	Description
0	TXEIR	R	Transmit FIFO Empty Raw Interrupt Status **Values:** ■ 0x0 (INACTIVE): ssi_txe_intr interrupt is not active prior to masking ■ 0x1 (ACTIVE): ssi_txe_intr interrupt is active prior masking **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

163

5.1.15 TXOICR

- **Name:** Transmit FIFO Overflow Interrupt Clear RegistersRegister
- **Description:** Transmit FIFO Overflow Interrupt Clear Register.

  Reset Value: 0x0
- **Size:** 32 bits
- **Offset:** 0x38
- **Exists:** Always

Table 5-20   Fields for Register: TXOICR

Bits	Name	Memory Access	Description
31:1	RSVD_TXOICR	R	TXOICR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	TXOICR	R	**Clear Transmit FIFO Overflow Interrupt.** This register reflects the status of the interrupt. A read from this register clears the ssi_txo_intr interrupt; writing has no effect. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.16   RXOICR

- **Name:** Receive FIFO Overflow Interrupt Clear Register
- **Description:** Receive FIFO Overflow Interrupt Clear Register.

  Reset Value: 0x0

- **Size:** 32 bits
- **Offset:** 0x3c
- **Exists:** Always

Table 5-21   Fields for Register: RXOICR

Bits	Name	Memory Access	Description
31:1	RSVD_RXOICR	R	RXOICR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RXOICR	R	**Clear Receive FIFO Overflow Interrupt.** This register reflects the status of the interrupt. A read from this register clears the ssi_rxo_intr interrupt; writing has no effect. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.17   RXUICR

- **Name:** Receive FIFO Underflow Interrupt Clear Register
- **Description:** Receive FIFO Underflow Interrupt Clear Register.
  Reset Value: 0x0
- **Size:** 32 bits
- **Offset:** 0x40
- **Exists:** Always

RSVD_RXUICR 31:1	0
RXUICR	

Table 5-22   Fields for Register: RXUICR

Bits	Name	Memory Access	Description
31:1	RSVD_RXUICR	R	RXUICR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RXUICR	R	**Clear Receive FIFO Underflow Interrupt.** This register reflects the status of the interrupt. A read from this register clears the ssi_rxu_intr interrupt; writing has no effect. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.18   MSTICR

■ **Name:** Multi-Master Interrupt Clear Register

■ **Description:** Multi-Master Interrupt Clear Register.

   Reset Value: 0x0

■ **Size:** 32 bits

■ **Offset:** 0x44

■ **Exists:** Always

RSVD_MSTICR 31:1	0
	MSTICR

Table 5-23   Fields for Register: MSTICR

Bits	Name	Memory Access	Description
31:1	RSVD_MSTICR	R	MSTICR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	MSTICR	R	**Clear Multi-Master Contention Interrupt.** This register reflects the status of the interrupt. A read from this register clears the ssi_mst_intr interrupt; writing has no effect. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.19   ICR

- **Name:** Interrupt Clear Register
- **Description:** Interrupt Clear Register.

    Reset Value: 0x0
- **Size:** 32 bits
- **Offset:** 0x48
- **Exists:** Always

Table 5-24   Fields for Register: ICR

Bits	Name	Memory Access	Description
31:1	RSVD_ICR	R	ICR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	ICR	R	**Clear Interrupts.** This register is set if any of the interrupts below are active. A read clears the ssi_txo_intr, ssi_rxu_intr, ssi_rxo_intr, and the ssi_mst_intr interrupts. Writing to this register has no effect. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.20   DMACR

- **Name:** DMA Control Register
- **Description:** This register is only valid when DW_apb_ssi is configured with a set of DMA Controller interface signals (SSI_HAS_DMA = 1). When DW_apb_ssi is not configured for DMA operation, this register will not exist and writing to the register's address will have no effect; reading from this register address will return zero. The register is used to enable the DMA Controller interface operation.

    Reset Value: 0x0

- **Size:** 32 bits
- **Offset:** 0x4c
- **Exists:** SSI_HAS_DMA == 1

Table 5-25   Fields for Register: DMACR

Bits	Name	Memory Access	Description
31:2	RSVD_DMACR	R	DMACR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
1	TDMAE	R/W	**Transmit DMA Enable.** This bit enables/disables the transmit FIFO DMA channel. **Values:** ■ 0x0 (DISABLE): Transmit DMA disabled ■ 0x1 (ENABLED): Transmit DMA enabled **Value After Reset:** 0x0 **Exists:** Always

Table 5-25   Fields for Register: DMACR (Continued)

Bits	Name	Memory Access	Description
0	RDMAE	R/W	**Receive DMA Enable.** This bit enables/disables the receive FIFO DMA channel **Values:** ■ 0x0 (DISABLE): Receive DMA disabled ■ 0x1 (ENABLED): Receive DMA enabled **Value After Reset:** 0x0 **Exists:** Always

5.1.21   DMATDLR

- **Name:** DMA Transmit Data Level

- **Description:** This register is only valid when the DW_apb_ssi is configured with a set of DMA interface signals (SSI_HAS_DMA = 1). When DW_apb_ssi is not configured for DMA operation, this register will not exist and writing to its address will have no effect; reading from its address will return zero.

   Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x50

- **Exists:** SSI_HAS_DMA == 1

Table 5-26   Fields for Register: DMATDLR

Bits	Name	Memory Access	Description
31:y	RSVD_DMATDLR	R	DMATDLR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** TX_ABW

Table 5-26   Fields for Register: DMATDLR (Continued)

Bits	Name	Memory Access	Description
x:0	DMATDL	R/W	**Transmit Data Level.** This bit field controls the level at which a DMA request is made by the transmit logic. It is equal to the watermark level; that is, the dma_tx_req signal is generated when the number of valid data entries in the transmit FIFO is equal to or below this field value, and TDMAE = 1. For information on the DMATDL decode values, see the "Slave SPI and SSP Serial Transfers" section in the DW_apb_ssi Databook. dma_tx_req is asserted when DMATDL or less data entries are present in the transmit FIFO **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** TX_ABW - 1

5.1.22   DMARDLR

- **Name:** DMA Receive Data Level

- **Description:** This register is only valid when DW_apb_ssi is configured with a set of DMA interface signals (SSI_HAS_DMA = 1). When DW_apb_ssi is not configured for DMA operation, this register will not exist and writing to its address will have no effect; reading from its address will return zero.

  Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x54

- **Exists:** SSI_HAS_DMA == 1

RSVD_DMARDLR	31:y
DMARDL	x:0

Table 5-27   Fields for Register: DMARDLR

Bits	Name	Memory Access	Description
31:y	RSVD_DMARDLR	R	DMARDLR Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** RX_ABW
x:0	DMARDL	R/W	**Receive Data Level.** This bit field controls the level at which a DMA request is made by the receive logic. The watermark level = DMARDL+1; that is, dma_rx_req is generated when the number of valid data entries in the receive FIFO is equal to or above this field value + 1, and RDMAE=1. For information on the DMARDL decode values, see the "Slave SPI and SSP Serial Transfers" section in the DW_apb_ssi Databook. dma_rx_req is asserted when DMARDL or more valid data entries are present in the receive FIFO. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** RX_ABW - 1

5.1.23 IDR

- **Name:** Identification Register

- **Description:** This register contains the peripherals identification code, which is written into the register at configuration time using coreConsultant.

  Reset Value: SSI_ID

- **Size:** 32 bits

- **Offset:** 0x58

- **Exists:** Always

IDCODE 31:0

Table 5-28   Fields for Register: IDR

Bits	Name	Memory Access	Description
31:0	IDCODE	R	**Identification code.** The register contains the peripheral's identification code, which is written into the register at configuration time using CoreConsultant. **Value After Reset:** SSI_ID **Exists:** Always

5.1.24 SSI_VERSION_ID

- **Name:** coreKit version ID Register
- **Description:** This read-only register stores the specific DW_apb_ssi component version.

  Reset Value: SSI_VERSION_ID
- **Size:** 32 bits
- **Offset:** 0x5c
- **Exists:** Always

SSI_COMP_VERSION   31:0

Table 5-29   Fields for Register: SSI_VERSION_ID

Bits	Name	Memory Access	Description
31:0	SSI_COMP_VERSION	R	Contains the hex representation of the Synopsys component version. Consists of ASCII value for each number in the version, followed by *. For example 32_30_31_2A represents the version 2.01*. **Value After Reset:** SSI_VERSION_ID **Exists:** Always

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

175

5.1.25 DRx (for x = 0; x <= 35)

- **Name:** Data Register x

- **Description:** The DW_apb_ssi data register is a 16/32-bit (depending on SSI_MAX_XFER_SIZE) read/write buffer for the transmit/receive FIFOs. If the configuration parameter SSI_MAX_XFER_SIZE is set to 32, then all 32 bits are valid, otherwise, only 16 bits ([15:0]) of the register are valid. When the register is read, data in the receive FIFO buffer is accessed. When it is written to, data are moved into the transmit FIFO buffer; a write can occur only when SSI_EN = 1. FIFOs are reset when SSI_EN = 0.**NOTE:** The DR register in the DW_apb_ssi occupies thirty-six 32-bit address locations of the memory map to facilitate AHB burst transfers. Writing to any of these address locations has the same effect as pushing the data from the pwdata bus into the transmit FIFO. Reading from any of these locations has the same effect as popping data from the receive FIFO onto the prdata bus. The FIFO buffers on the DW_apb_ssi are not addressable.

  Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0x60

- **Exists:** Always

Table 5-30   Fields for Register: DRx (for x = 0; x <= 35)

Bits	Name	Memory Access	Description
31:16	RSVD_DR	R	DR{i} Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** SSI_MAX_XFER_SIZE == 16 **Volatile:** true

Table 5-30   Fields for Register: DRx (for x = 0; x <= 35) (Continued)

Bits	Name	Memory Access	Description
x:0	DR	R/W	Data Register. When writing to this register, you must right-justify the data. Read data are automatically right-justified. If SSI_MAX_XFER_SIZE configuration parameter is set to 32, all 32 bits are valid. Otherwise, only 16 bits ([15:0]) of the register are valid. Read = Receive FIFO buffer Write = Transmit FIFO buffer. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** SSI_MAX_XFER_SIZE - 1

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

177

5.1.26 RX_SAMPLE_DLY

- **Name:** RX Sample Delay Register

- **Description:** This register is only valid when the DW_apb_ssi is configured with rxd sample delay logic (SSI_HAS_RX_SAMPLE_DELAY==1). When the DW_apb_ssi is not configured with rxd sample delay logic, this register will not exist and writing to its address location will have no effect; reading from its address will return zero.

  This register control the number of ssi_clk cycles that are delayed (from the default sample time) before the actual sample of the rxd input occurs. It is impossible to write to this register when the DW_apb_ssi is enabled. The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

  Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0xf0

- **Exists:** SSI_HAS_RX_SAMPLE_DELAY == 1

Table 5-31   Fields for Register: RX_SAMPLE_DLY

Bits	Name	Memory Access	Description
31:8	RSVD_RX_SAMPLE_DLY	R	SAMPLE_DLY Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-31   Fields for Register: RX_SAMPLE_DLY (Continued)

Bits	Name	Memory Access	Description
7:0	RSD	R/W	**Rxd Sample Delay.** This register is used to delay the sample of the rxd input port. Each value represents a single ssi_clk delay on the sample of rxd. **Note:** If this register is programmed with a value that exceeds the depth of the internal shift registers (SSI_RX_DLY_SR_DEPTH) zero delay will be applied to the rxd sample. **Value After Reset:** 0x0 **Exists:** Always

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

179

5.1.27 SPI_CTRLR0

- **Name:** SPI Control Register
- **Description:** This register is valid only when SSI_SPI_MODE is either set to "Dual" or "Quad" or "Octal" mode. This register is used to control the serial data transfer in SPI mode of operation. The register is only relevant when SPI_FRF is set to either 01 or 10 or 11. It is not possible to write to this register when the DW_apb_ssi is enabled (SSI_EN=1). The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

   Reset Value: 0x00000200

- **Size:** 32 bits
- **Offset:** 0xf4
- **Exists:** SSI_SPI_MODE != 0

Table 5-32   Fields for Register: SPI_CTRLR0

Bits	Name	Memory Access	Description
31:19	RSVD_SPI_CTRLR0	R	SPI_CTRLR0 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
18	SPI_RXDS_EN	* Varies	**Read data strobe enable bit.** Once this bit is set to 1 DW_apb_ssi will use Read data strobe (rxds) to capture read data in DDR mode. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** "(SSI_HAS_RXDS==0) ? \"read-only\": \"read-write\""

Table 5-32   Fields for Register: SPI_CTRLR0 (Continued)

Bits	Name	Memory Access	Description
17	INST_DDR_EN	* Varies	**Instruction DDR Enable bit.** This will enable Dual-data rate transfer for Instruction phase. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** "(SSI_HAS_DDR==0) ? \"read-only\": \"read-write\""
16	SPI_DDR_EN	* Varies	**SPI DDR Enable bit.** This will enable Dual-data rate transfers in Dual/Quad/Octal frame formats of SPI. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** "(SSI_HAS_DDR==0) ? \"read-only\": \"read-write\""
15:11	WAIT_CYCLES	R/W	**Wait cycles** Number of wait cycles in Dual/Quad/Octal mode between control frames transmit and data reception. This value is specified as number of SPI clock cycles. For information on the WAIT_CYCLES decode value, see "Read Operation in Enhanced SPI Modes" section in the DW_apb_ssi Databook. **Value After Reset:** 0x0 **Exists:** Always
10	RSVD_SPI_CTRLR0_10	R	CTRLR0_10 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
9:8	INST_L	R/W	**Instruction Length** Dual/Quad/Octal mode instruction length in bits. **Values:** ■ 0x0 (INST_L_0): 0-bit (No Instruction) ■ 0x1 (INST_L_1): 4-bit Instruction ■ 0x2 (INST_L_2): 8-bit Instruction ■ 0x3 (INST_L_3): 16-bit Instruction **Value After Reset:** 0x2 **Exists:** Always
7:6	RSVD_SPI_CTRLR0_6_7	R	CTRLR0_6_7 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-32   Fields for Register: SPI_CTRLR0 (Continued)

Bits	Name	Memory Access	Description
5:2	ADDR_L	R/W	**Address Length**. This bit defines Length of Address to be transmitted. Only after this much bits are programmed in to the FIFO the transfer can begin. For information on the ADDR_Ldecode value, see "Read Operation in Enhanced SPI Modes" section in the DW_apb_ssi Databook. **Values:** ■ 0x0 (ADDR_L_0): 0-bit Address Width ■ 0x1 (ADDR_L_1): 4-bit Address Width ■ 0x2 (ADDR_L_2): 8-bit Address Width ■ 0x3 (ADDR_L_3): 12-bit Address Width ■ 0x4 (ADDR_L_4): 16-bit Address Width ■ 0x5 (ADDR_L_5): 20-bit Address Width ■ 0x6 (ADDR_L_6): 24-bit Address Width ■ 0x7 (ADDR_L_7): 28-bit Address Width ■ 0x8 (ADDR_L_8): 32-bit Address Width ■ 0x9 (ADDR_L_9): 36-bit Address Width ■ 0xa (ADDR_L_10): 40-bit Address Width ■ 0xb (ADDR_L_11): 44-bit Address Width ■ 0xc (ADDR_L_12): 48-bit Address Width ■ 0xd (ADDR_L_13): 52-bit Address Width ■ 0xe (ADDR_L_14): 56-bit Address Width ■ 0xf (ADDR_L_15): 60-bit Address Width **Value After Reset:** 0x0 **Exists:** Always
1:0	TRANS_TYPE	R/W	**Address and instruction transfer format.** Selects whether DW_apb_ssi will transmit instruction/address either in Standard SPI mode or the SPI mode selected in CTRLR0.SPI_FRF field. 00 - Instruction and Address will be sent in Standard SPI Mode. 01 - Instruction will be sent in Standard SPI Mode and Address will be sent in the mode specified by CTRLR0.SPI_FRF. 10 - Both Instruction and Address will be sent in the mode specified by SPI_FRF. 11 - Reserved. **Value After Reset:** 0x0 **Exists:** Always

5.1.28 TXD_DRIVE_EDGE

- **Name:** Transmit Drive Edge Register

- **Description:** This Register is valid only when SSI_HAS_DDR is equal to 1. This register is used to control the driving edge of TXD register in DDR mode. It is not possible to write to this register when the DW_apb_ssi is enabled (SSI_EN=1). The DW_apb_ssi is enabled and disabled by writing to the SSIENR register.

  Reset Value: 0x0

- **Size:** 32 bits

- **Offset:** 0xf8

- **Exists:** SSI_HAS_DDR != 0

Table 5-33   Fields for Register: TXD_DRIVE_EDGE

Bits	Name	Memory Access	Description
31:8	RSVD_TXD_DRIVE_EDGE	R	DRIVE_EDGE Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
7:0	TDE	R/W	TXD Drive edge - value of which decides the driving edge of tramit data. The maximum value of this regster is = (BAUDR/2) -1. **Value After Reset:** 0x0 **Exists:** Always

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

183

5.1.29   RSVD

- **Name:** RSVD - Reserved address location
- **Description:** RSVD - Reserved address location.
- **Size:** 32 bits
- **Offset:** 0xfc
- **Exists:** Always

Table 5-34   Fields for Register: RSVD

Bits	Name	Memory Access	Description
31:0	RSVD	R	RSVD 31to0 Reserved address location **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

6

Programming the DW_apb_ssi

This chapter describes the programmable features of the DW_apb_ssi.

6.1 Programming Considerations

You should program the following features during the configuration setup:

- APB data bus width
- Type of device configuration; that is, serial master or serial slave
- Depth of receive and transmit FIFO buffers
- Peripheral ID code
- Whether to include DMA handshaking interface signals
- Whether the interrupt level is active high or active low
- Whether interrupts are individual or combined
- Whether to generate a clock enable input for the ssi_clk
- Whether or not pclk and ssi_clk are synchronous
- Whether to hardcode the frame format, and what type of frame format:
  - Motorola SPI – requires setting the serial clock polarity and phase
  - Texas Instruments Synchronous Serial Protocol
  - National Semiconductor Microwire

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

185

Figure 6-1 shows a typical software flow for starting a DW_apb_ssi master SPI/SSP serial transfer. The diagram also shows the hardware flow inside the serial-master component.

Figure 6-1   DW_apb_ssi Master SPI/SSP Transfer Flow
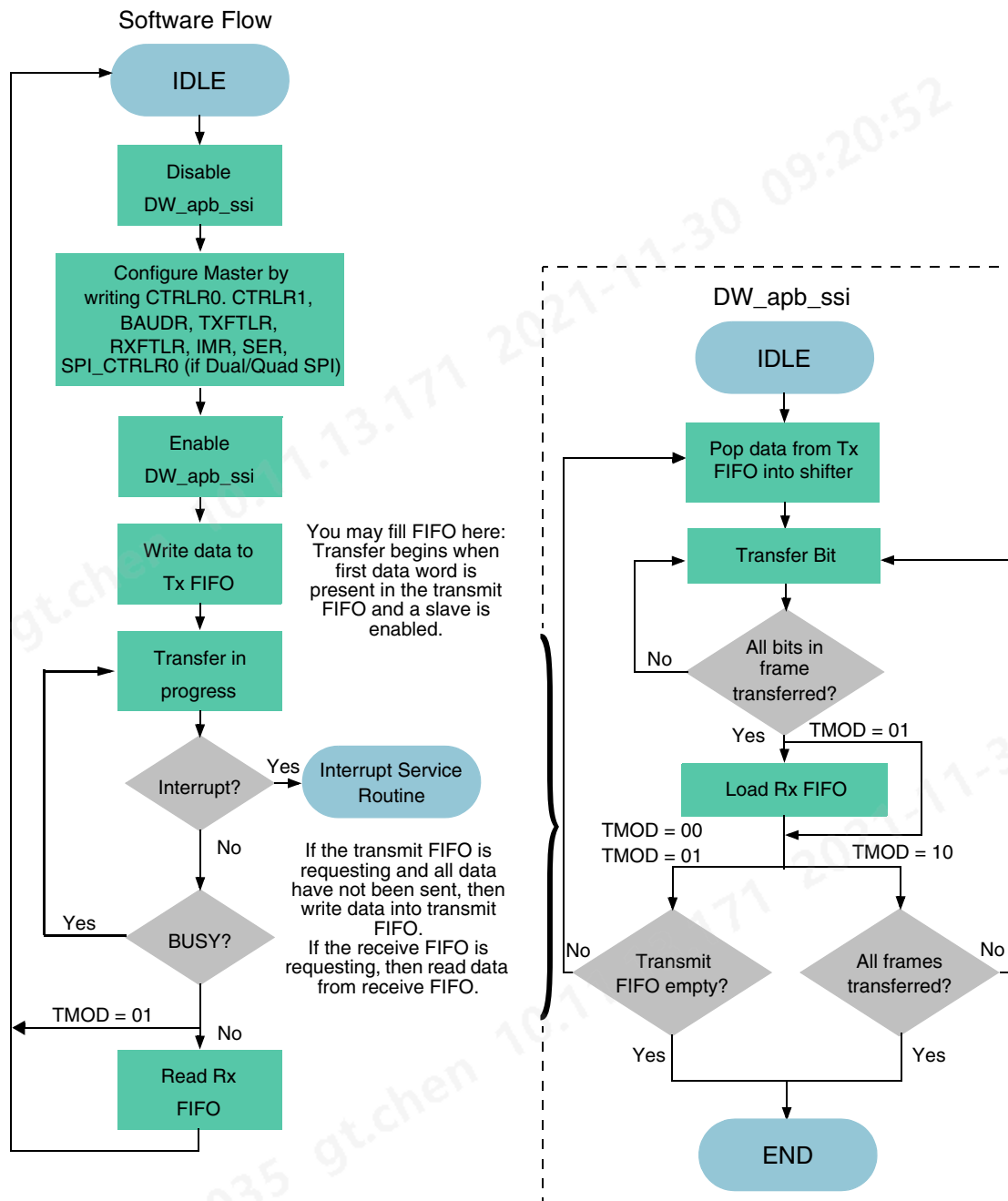
For more detailed information on the DW_apb_ssi master SPI/SSP serial transfer flow, see "Master SPI and SSP Serial Transfers" on page 38.

Figure 6-2 shows a typical software flow for starting a DW_apb_ssi master Microwire serial transfer. The diagram also shows the hardware flow inside the serial-master component.

Figure 6-2   DW_apb_ssi Master Microwire Transfer Flow
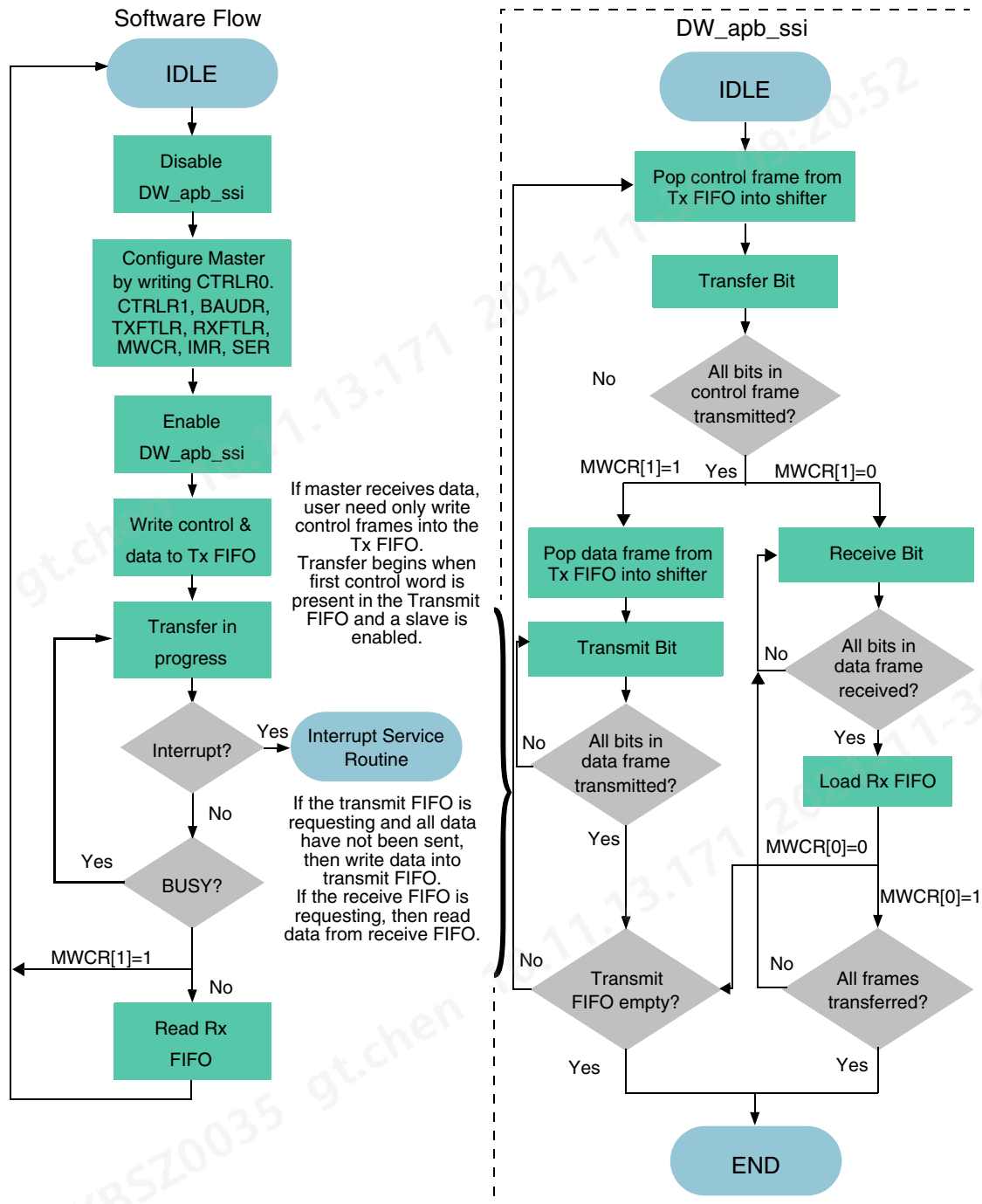
For more detailed information on the DW_apb_ssi master Microwire serial transfer flow, see "Master Microwire Serial Transfers" on page 41.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

187

Figure 6-3 shows a typical software flow for a DW_apb_ssi slave SPI or SSP serial transfer. The diagram also shows the hardware flow inside the serial-slave component.

Figure 6-3   DW_apb_ssi Slave SPI/SSP Transfer Flow
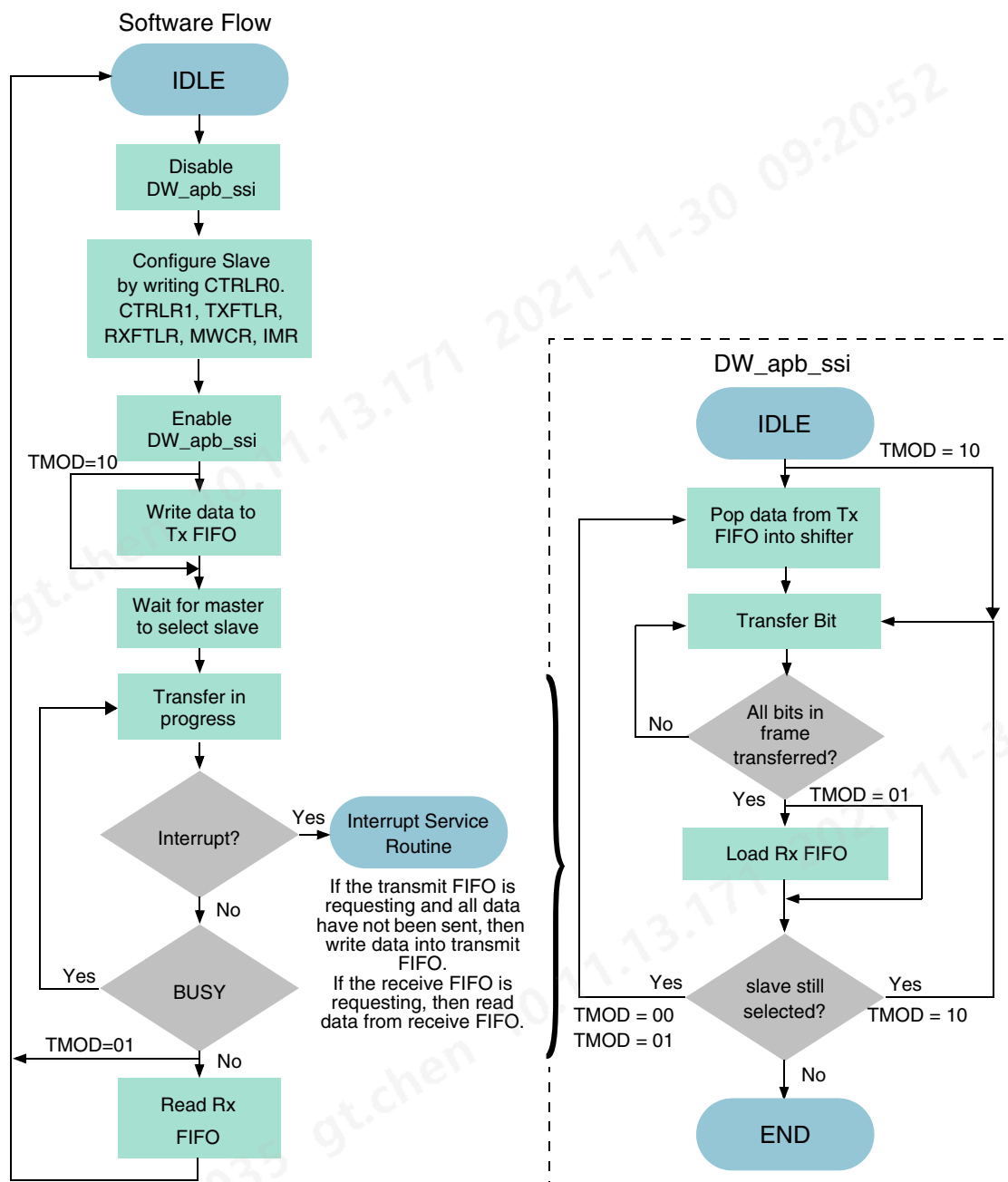
For more detailed information on the DW_apb_ssi slave SPI or SSP serial transfer flow, see "Slave SPI and SSP Serial Transfers" on page 44.

7

Verification

This chapter provides an overview of the testbench available for DW_apb_ssi verification. Once you have configured DW_apb_ssi in either coreAssembler or coreConsultant and set up the verification environment, you can run simulations automatically.

👉 **Note** Synopsys does not currently have a VMT-based VIP model for the SSI interface; thus Synopsys does not support full-system verification of the SSI interface at this time.

The DW_apb_ssi DesignWare IP does ship with a simple testbench, where the SSI interface is driven by Vera-based Bus Functional Models (BFMs) created solely for use in this testbench. The models are included in the testbench in compiled Vera format (.vro); thus the BFM implementation is not visible to the customer, and the customer cannot use these BFMs in their own verification environment.

👉 **Note** The packaged test benches are only for validating the IP configuration in coreConsultant GUI. It is not for system level validation.

IPs that have the Vera test bench packaged, these test benches are encrypted.

7.1   Overview of Vera Tests

The DW_apb_ssi verification testbench performs the set of tests in this section, which have been written to exhaustively verify the functionality and have also achieved maximum RTL code coverage. All tests use the APB Interface to dynamically program memory-mapped registers during tests.

⚠️ **Attention** You should ensure that you have the supported version of the VIP components for this release; otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, see the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide.*

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

189

7.1.1   APB Interface

This suite of tests is run to verify that the APB interface functions correctly by checking the following:

- All address locations are written to with valid data

- Configured MacroCell is AMBA-compliant

- Read/write coherent

- Reset value of all registers

- Functionality of all registers

> 👉 **Note**   DW_apb_ssi does not start any operation when SSI_EN is held low. This control bit is verified separately. Regardless of programmed values of the registers, there is no activity on the component interface or within DW_apb_ssi once bit 0 of SSI Enable Register (SSIENR) is 0. Within the HDL code, there are a number of checkers that insert error information into the log file when the simulation is performed on an invalid configuration.

7.1.2   DW_apb_ssi as Master

This suite of tests is run only when the DW_apb_ssi is configured as a master, during which time all transfers are initiated by the DW_apb_ssi. When a master, the DW_apb_ssi must generate the clock sclk_out.

7.1.3   DW_apb_ssi as Slave

This suite of tests is run only when the DW_apb_ssi is configured as a slave. Similar to the tests developed for the master, the driving force is the serial-master Bus Functional Model (BFM). The DW_apb_ssi should be tested for all frame formats, for all transfer modes, for all length of data frames, for all combinations of clock polarity, and clock phase. The serial-master BFM initiates all the serial transfers.

7.1.4   DW_apb_ssi with DMA Interface

This suite of tests is only run when the DW_apb_ssi is configured with a handshaking interface. DMA is implemented for both Transmit and Receive. These test verify the following:

- SSI_HAS_DMA

- After reset that all DMA control registers read zero.

- It is possible to set the Transmit/Receive DMA enable bit through an APB write. Confirm it is possible to clear these bits through an APB write.[17]

- Once the transmit enable bit is set, a DMA transmit transfer request is generated provided the number of entries in the FIFO is less than or equal to the DMA Data Level. The DMAing BFM is configured initially not to respond to requests. APB transfers can fill the FIFO and confirm that the request line is removed as the level in the FIFO fills. Confirm that when the DMAing BFM is configured to respond that it fills the FIFO with sufficient data to remove the request.

- The dma_tx_req signal is active when the number of entries in the transmit FIFO is equal to or below the DMA Data Level. And remains active.

- The dma_tx_single signal is asserted when there is at least one empty location in the transmit FIFO. Confirm that it is cleared when the transmit FIFO is full.

- The dma_tx_ack signal causes a pulse on dma_tx_req so that the DMAing BFM can respond to requests for subsequent DMA transfers. The BFM only responds to rising edges on the request line.

- The dma_tx_finish signal clears the transmit enable bit. Confirm that the dma_tx_req and dma_tx_single lines are cleared. Confirm that dma_tx_finish stays active until dma_tx_req is sampled low.18

- All data words are transmitted through DMA and that there is never any data left within the DMAing BFM, one should be able to configure the BFM to read in bursts or in singles. This should be confirmed for a range of threshold values.

- The same operation exists for receive FIFOs.

Once a dma_[r|t]x_finish is received, the corresponding enable bit is cleared. Confirm that there are no subsequent requests for DMA once the enable is removed.

7.1.5   Interrupts

These tests verify the following:

- The Transmit FIFO Empty interrupt is not active when the DW_apb_ssi is returned from reset.

- The Receive FIFO Full interrupt is not active when the DW_apb_ssi is returned from reset.

- The Transmit FIFO Overflow interrupt is not active when the DW_apb_ssi is returned from reset.

- The Receive FIFO Overflow interrupt is not active when the DW_apb_ssi is returned from reset.

- The Receive FIFO Underflow interrupt is not active when the DW_apb_ssi is returned from reset.

- The MultiMaster Contention interrupt is not active when the DW_apb_ssi is returned from reset.

7.2 Overview of DW_apb_ssi Testbench

As illustrated in Figure 7-1, the DW_apb_ssi testbench is a Verilog testbench that includes an instantiation of the design under test (DUT), AHB and APB Bridge bus models, and a Vera shell.

Figure 7-1   DW_apb_ssi Testbench
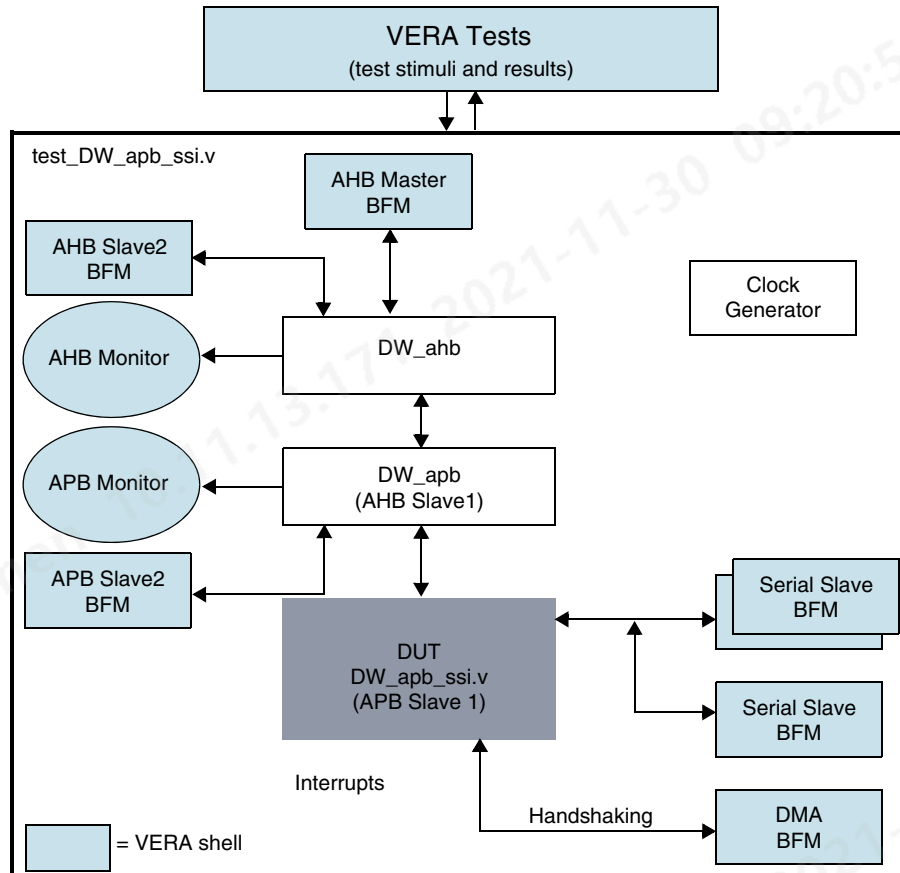
The Vera shell consists of a number of serial-slave BFMs, a master slave BFM, and a DMA BFM to simulate and stimulate traffic to and from the DW_apb_ssi.

The test_DW_apb_ssi.v file shows the instantiation of the top-level MacroCell in a testbench and resides in the *workspace*/src directory. The testbench checks your configuration selected in the Specify Configuration task of coreConsultant. The testbench also determines if 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_ssi form the actual testbench for DW_apb_ssi.

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.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

193

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
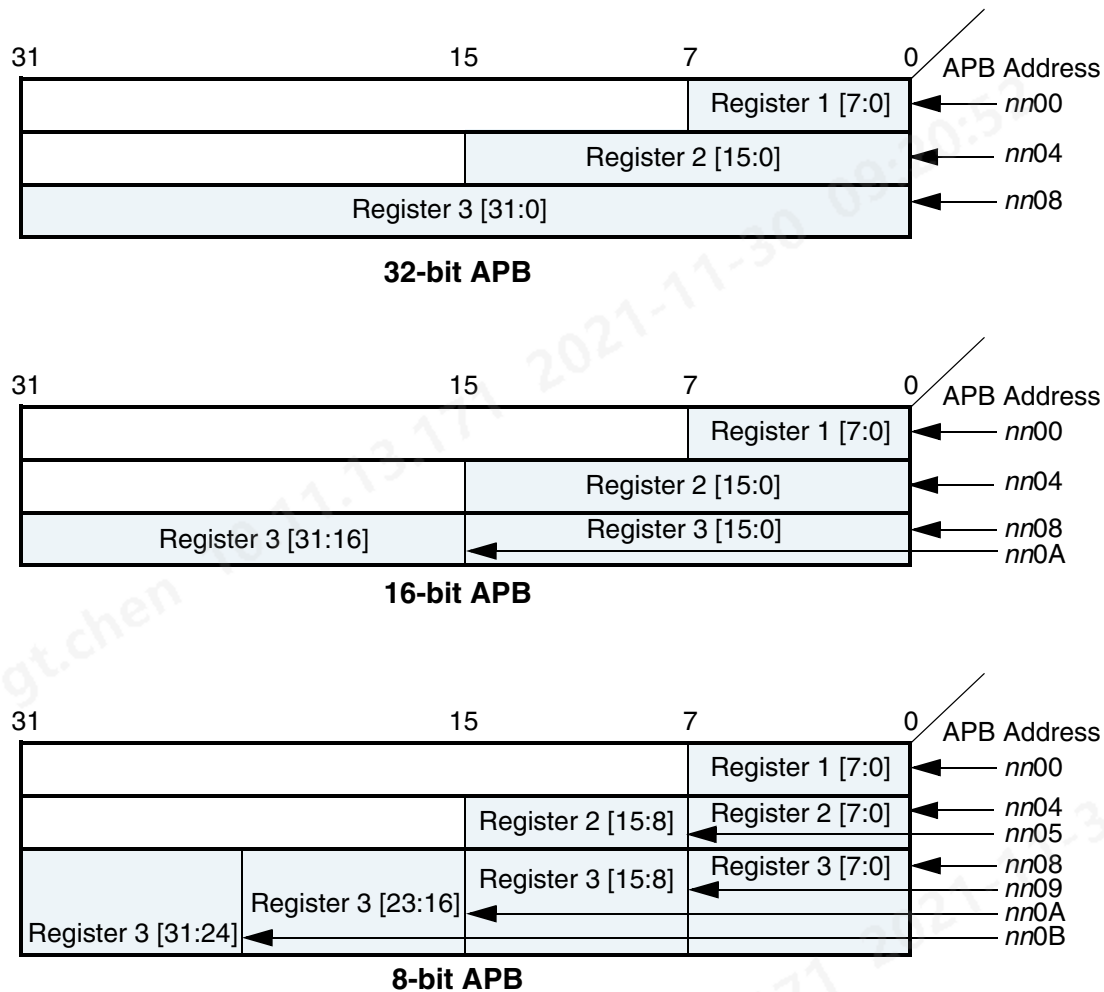
8.1.2   32-bit Bus System

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.

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
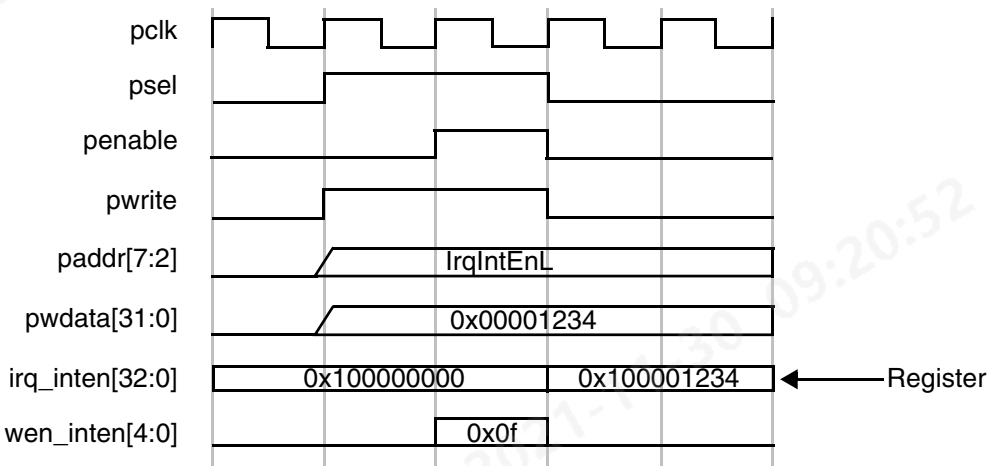
A write can occur after the first phase with penable low, or after the second phase when penable is high. The second phase is preferred and is used in all APB slave components. The timing diagram is shown with the write occurring after the second phase. Whenever the address on paddr matches a corresponding 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
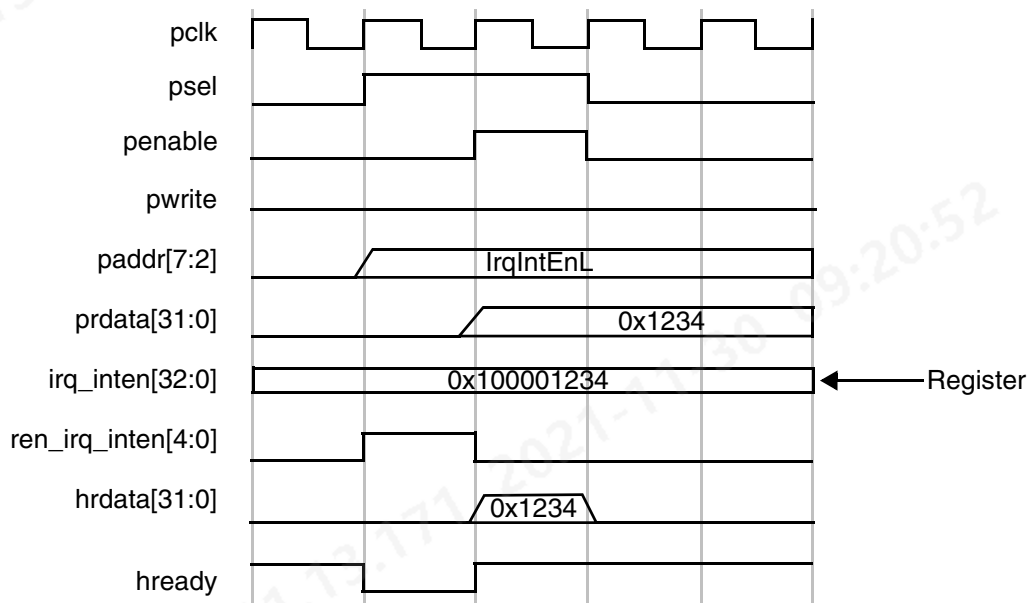
Whenever the address on paddr matches the corresponding address from the memory map—psel is high, pwrite and penable are low—then the corresponding read enable is generated. The read data is registered within the peripheral before passing back to the master through the DW_apb and DW_ahb.

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.

198

SolvNetPlus
DesignWare

Synopsys, Inc.

4.03a
December 2020

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.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

199

Figure 8-4   Coherent Loading – Identical Synchronous Clocks
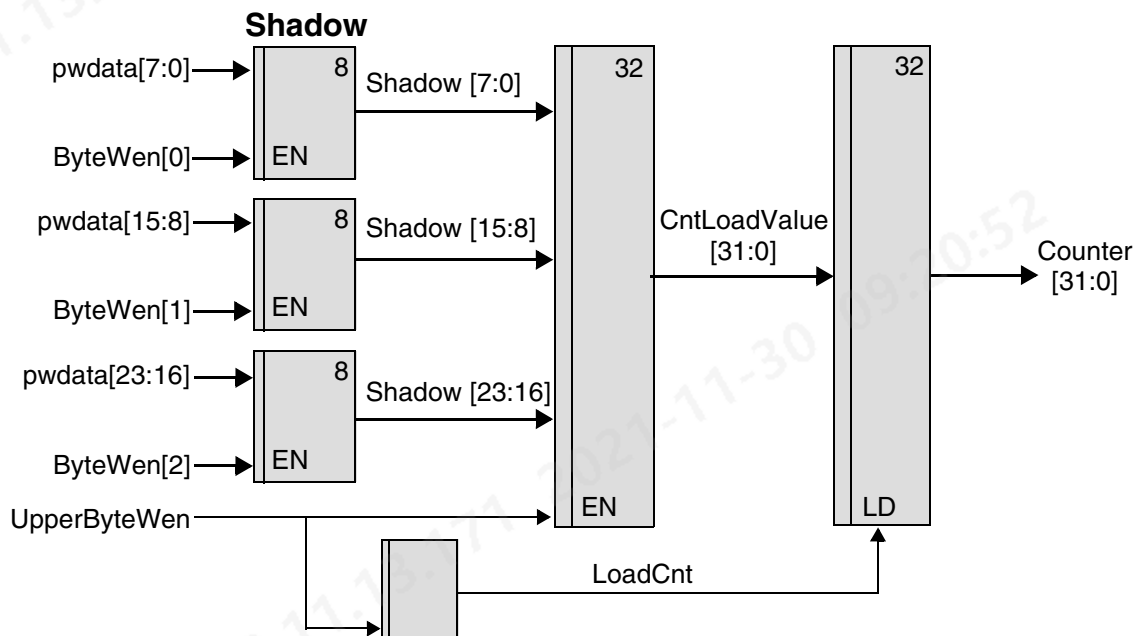
The following figure shows a 32-bit register that is written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal lasts for one cycle and is used to load the counter with CntLoadValue.

Figure 8-5   Coherent Loading – Identical Synchronous Clocks
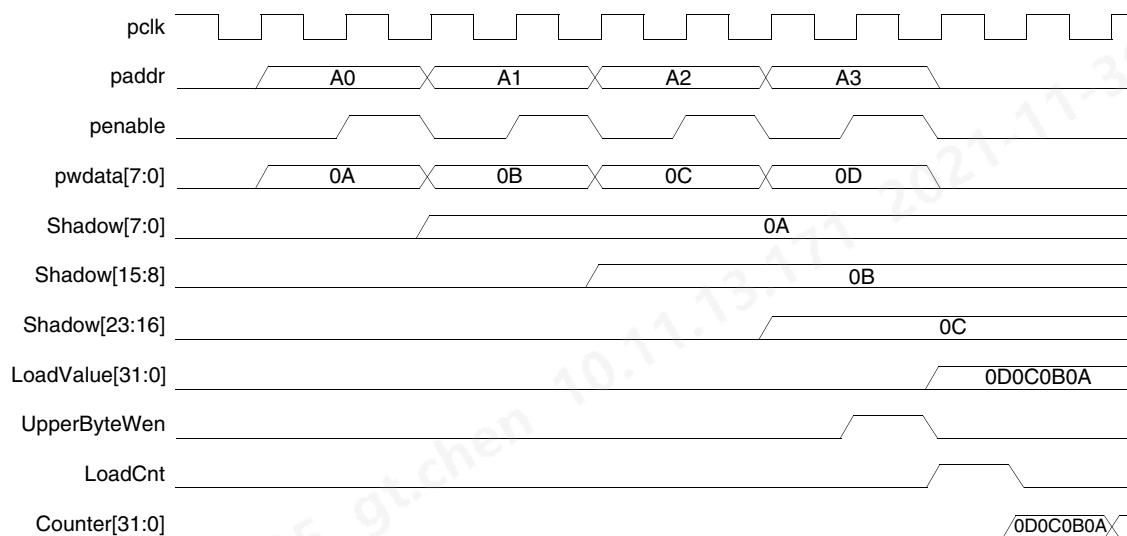
Each of the bytes that make up the load register are stored into shadow registers until the final byte is written. The shadow register is up to three bytes wide. The contents of the shadow registers and the final byte are transferred into the CntLoadValue register when the final byte is written. The 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
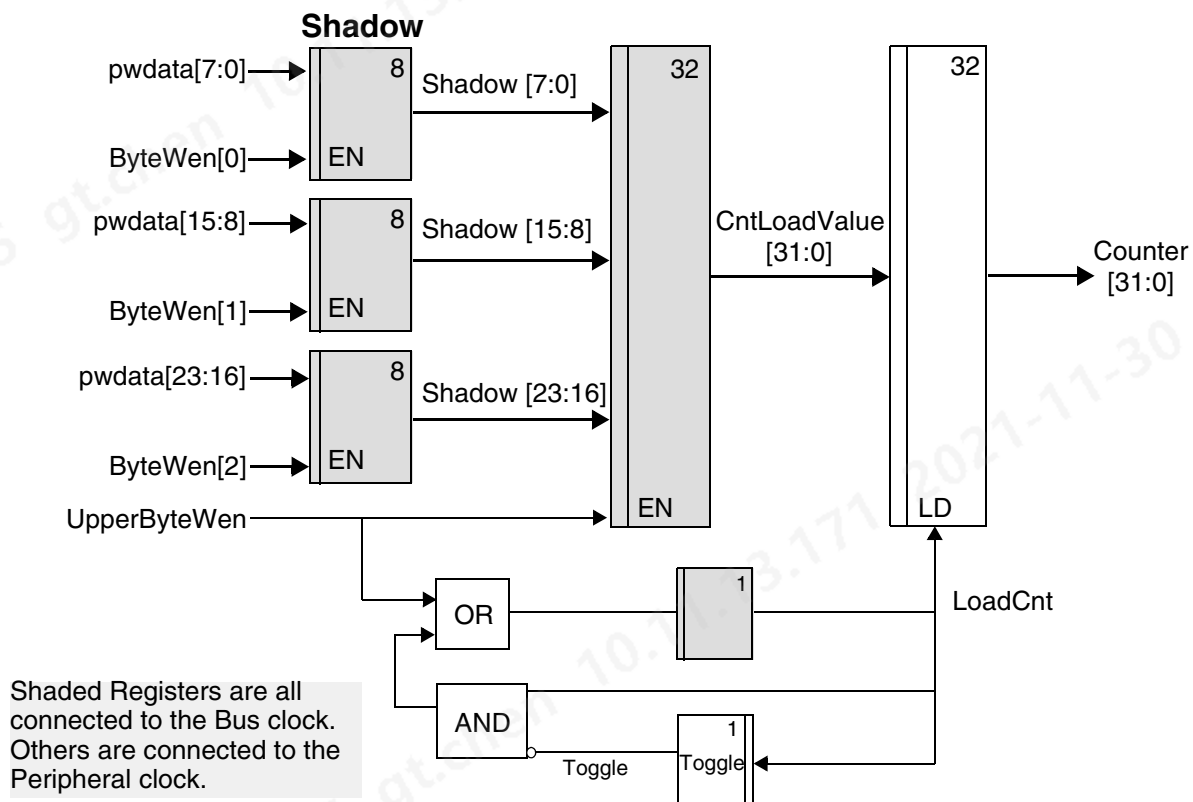
The following figure shows a 32-bit register being written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal is extended until a change in the toggle is detected and is used to load the counter.

Figure 8-7   Coherent Loading – Synchronous Clocks
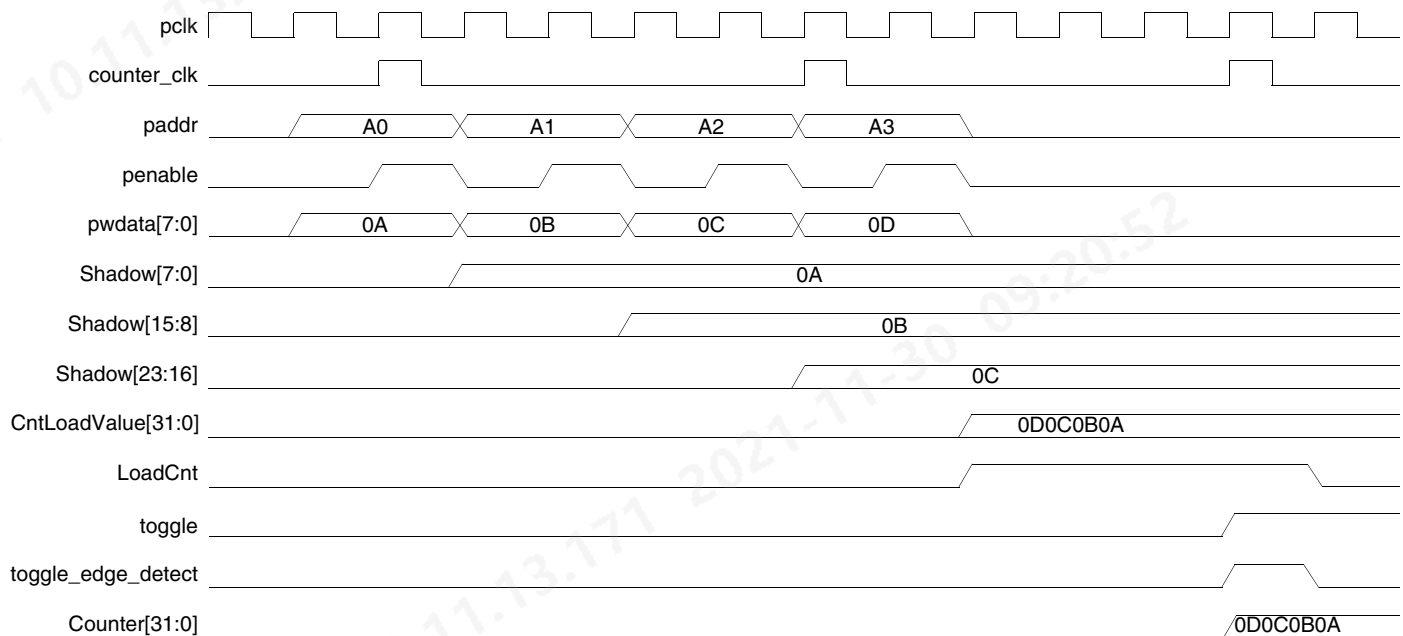
8.5.1.3   Asynchronous Clocks

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

Shaded and edge detect registers are all connected to the Bus clock. Others are connected to the Peripheral clock.

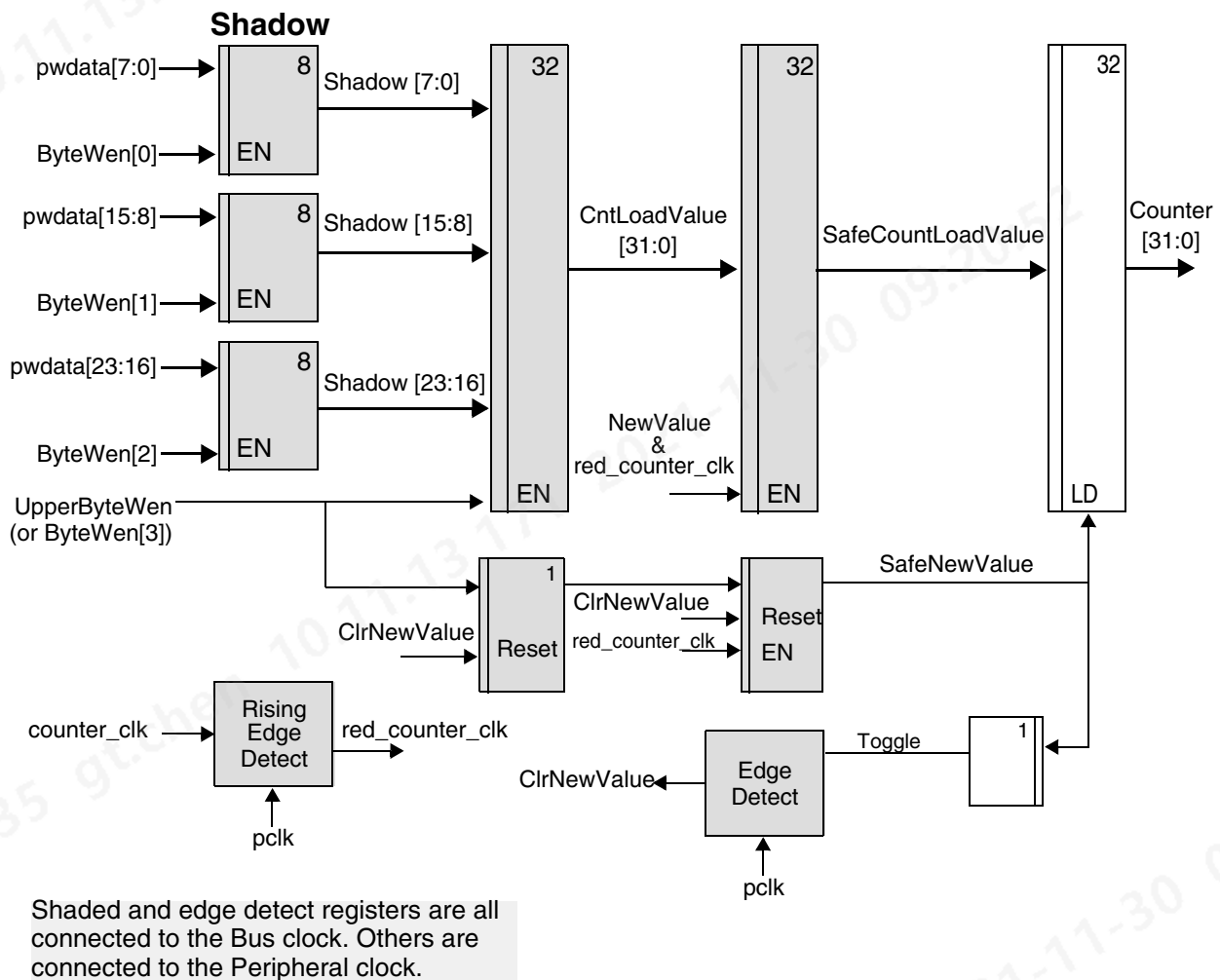When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock 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.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

203

Figure 8-9   Coherent Loading – Asynchronous Clocks
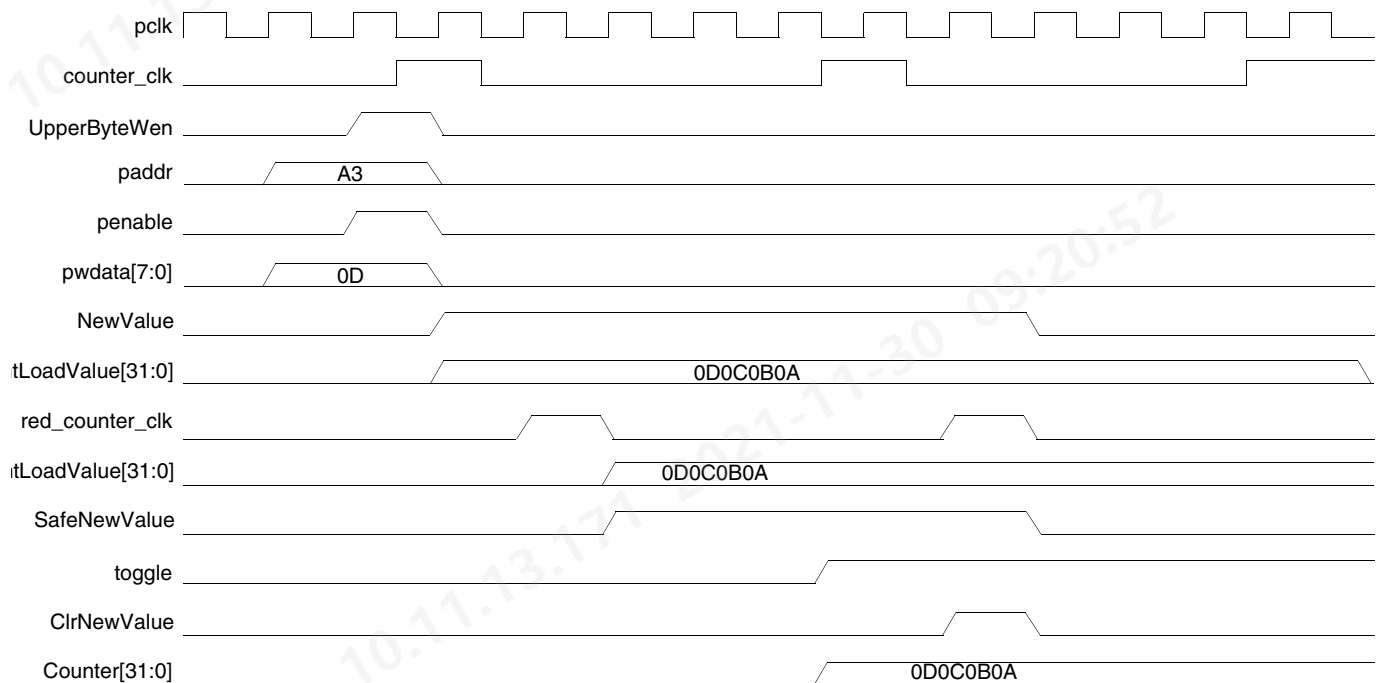
The NewValue signal is extended until a change in the toggle is detected and is used to update the safe value. The SafeNewValue is used to load the counter at the rising edge of the peripheral clock. Each time a new value is written the toggle bit is flipped and the edge detection of the toggle is used 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
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
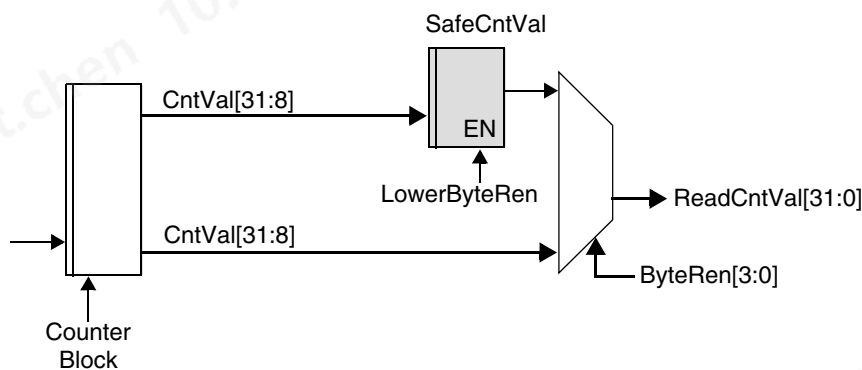
Figure 8-11   Coherent Registering – Synchronous Clocks
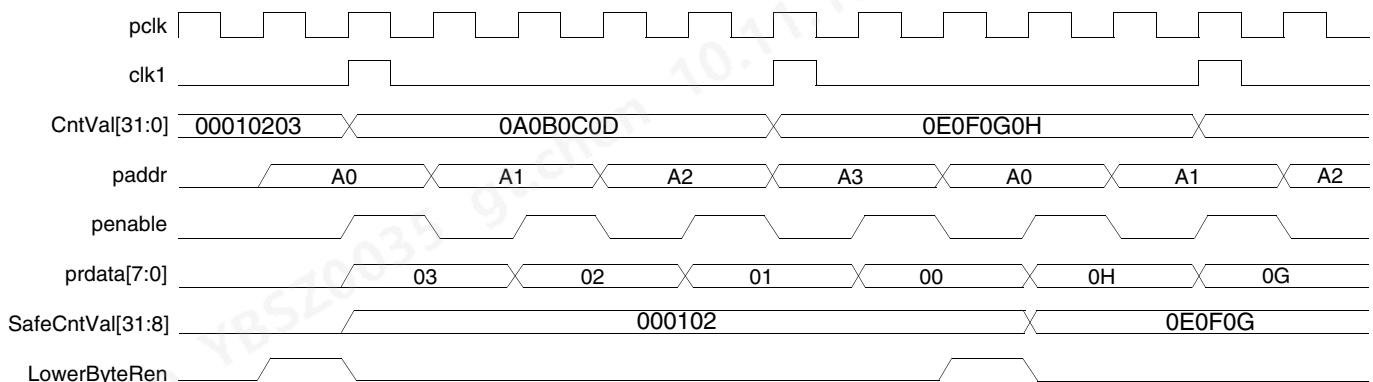
8.5.2.2 Asynchronous Clocks

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
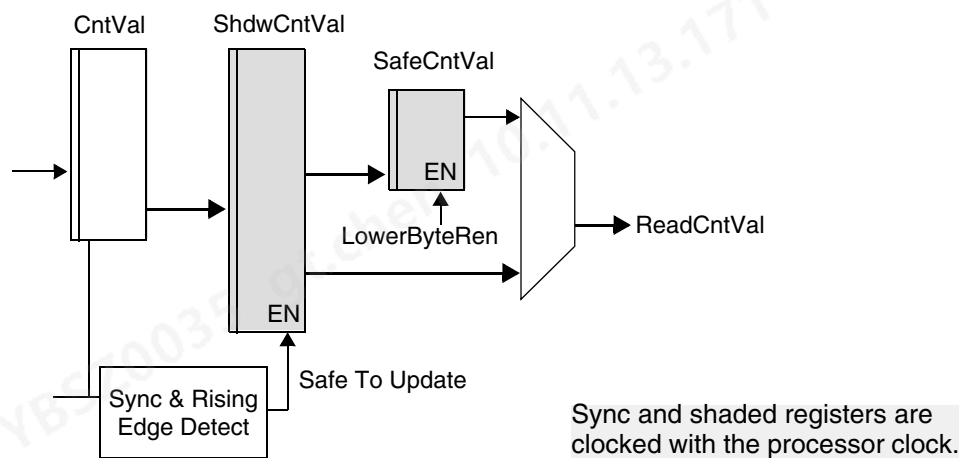
8.6 Timing Exceptions

- For details on multi cycle paths, see the DW_apb_ssi.sdc file generated by the Design Compiler.

- For details on quasi-static signals on the design, refer to manual.sgdc report generated by the SpyGlass tool.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

207

8.7 Performance

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

8.7.1 Power Consumption, Frequency, Area, and DFT Coverage

Table 8-3 provides information about the synthesis results (power consumption, frequency and area) and DFT coverage of the DW_apb_ssi using the industry standard 7nm technology library.

Table 8-3   Synthesis Results for DW_apb_ssi

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Default Configuration	pclk=100 MHz ssi_clk=100 MHz	6328	20 nW	0.032 mW	100	99.45	99.9

Table 8-3   Synthesis Results for DW_apb_ssi (Continued)

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Typical Configuration - 1 SSI_IS_MASTER = 1 SSI_APBIF_TYPE = 2 SSI_SPI_MODE = 3 SSI_IO_MAP_EN = 1 SSI_HAS_DDR = 1 SSI_HAS_RXDS = 1 SSI_XIP_EN = 1 SSI_SPI_DM_EN = 1 SSI_HC_FRF = 0 SSI_DFLT_FRF = 0 SSI_TX_FIFO_DEPTH = 32 SSI_RX_FIFO_DEPTH = 32 SSI_NUM_SLAVES = 16 SSI_HAS_DMA = 1 SSI_INTR_POL = 1 SSI_INTR_IO = 1 SSI_SYNC_CLK = 0 SSI_DFLT_SCPOL = 0 SSI_DFLT_SCPH = 0 SSI_SCPH0_SSTOGGLE = 1 SSI_APB3_ERR_RESP_EN = 1 SSI_HAS_RX_SAMPLE_DELAY =1 SSI_RX_DLY_SR_DEPTH = 8 SSI_INC_ENDCONV = 1	pclk=100 MHz ssi_clk=100 MHz	20193	50 nW	0.154 mW	99.83	99.48	99.7

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

209

Table 8-3   Synthesis Results for DW_apb_ssi (Continued)

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Typical Configuration - 2 SSI_IS_MASTER = 0 SSI_APBIF_TYPE = 1 SSI_HC_FRF = 0 SSI_DFLT_FRF = 0 SSI_TX_FIFO_DEPTH = 32 SSI_RX_FIFO_DEPTH = 32 SSI_HAS_DMA = 1 SSI_INTR_POL = 1 SSI_INTR_IO = 1 SSI_SYNC_CLK = 0 SSI_ENH_CLK_RATIO = 1 SSI_DFLT_SCPOL = 0 SSI_DFLT_SCPH = 0 SSI_SCPH0_SSTOGGLE = 1 SSI_APB3_ERR_RESP_EN = 1	pclk=100 MHz ssi_clk=100 MHz	14768	40 nW	0.119 mW	99.38	99.5	99.6

A

Basic Core Module (BCM) Library

The Basic Core Module (BCM) Library is a library of commonly used blocks for the Synopsys DesignWare IP development. These BCMs are configurable on an instance-by-instance basis and, for the majority of BCM designs, there is an equivalent (or nearly equivalent) DesignWare Building Block (DWBB) component.

This appendix contains the following sections:

- "BCM Library Components" on page 211
- "Synchronizer Methods" on page 211

A.1 BCM Library Components

Table A-1 describes the list of BCM library components used in DW_apb_ssi.

Table A-1 BCM Library Components

BCM Module Name	BCM Description	DWBB Equivalent
DW_apb_ssi_bcm21	Single clock data bus synchronizer	DW_sync
DW_apb_ssi_bcm07	Synchronous (Dual-Clock) FIFO Controller with Static Flags	DW_fifoctl_s2_sf
DW_apb_ssi_bcm06	Synchronous (Single Clock) FIFO Controller with Dynamic Flags	DW_fifoctl_s1_df
DW_apb_ssi_bcm57	Synchronous Write-Port, Asynchronous Read-Port RAM (Flip-Flop Based)	DW_ram_r_w_s_dff
DW_apb_ssi_bcm36_nhs	Bus Delay Component	--

A.2 Synchronizer Methods

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

This section contains the following sections:

- "Synchronizers Used in DW_apb_ssi" on page 212
- "Synchronizer 1: Simple Double Register Synchronizer (DW_apb_ssi)" on page 212

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

211

■ "Synchronizer 2: Synchronous (Dual-clock) FIFO Controller With Static Flags" on page 213

> **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.2.1 Synchronizers Used in DW_apb_ssi

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_ssi are listed and cross referenced to the synchronizer type in Table A-2. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

Table A-2   Synchronizers used in DW_apb_ssi

Synchronizer module file	Sub module file	Synchronizer Type and Number
DW_apb_ssi_bcm21.v		Synchronizer 1: Simple Multiple Register Synchronizer
DW_apb_ssi_bcm07.v	DW_apb_ssi_bcm05.v DW_apb_ssi_bcm21.v	Synchronizer 2: Synchronous dual clock FIFO Controller with Static Flags

> **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.

A.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_apb_ssi)

This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. The synchronization scheme depends on core configuration. If pclk and ssi_clk are asynchronous (SSI_SYNC_CLK =0) or if DW_apb_ssi is configured to be slave (SSI_IS_MASTER=0) then DW_apb_ssi_bcm21 is instantiated inside the core for synchronization. This 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)

Configured as: *f_sync_type* = 2, *tst_mode* = 1
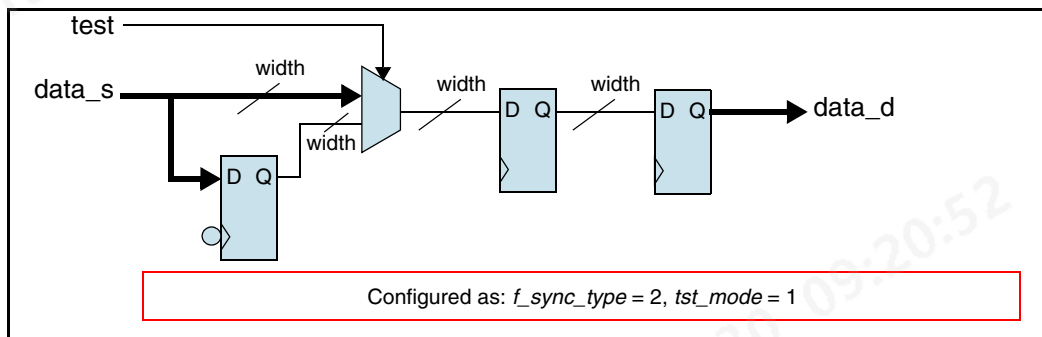
When DW_apb_ssi is working in slave mode and advanced slave architecture is selected (SSI_ENH_CLK_RATIO=1), then DW_apb_ssi uses faster synchronization scheme which uses one negative edge and one positive edge synchronization.

A.2.3   Synchronizer 2: Synchronous (Dual-clock) FIFO Controller With Static Flags

DW_apb_ssi_bcm07 is a dual independent clock FIFO RAM controller. It is designed to interface with a dual-port synchronous RAM. The FIFO controller provides address generation, write-enable logic, flag logic, and operational error detection logic. The DW_apb_ssi_bcm07 component is used as the FIFO controller when the SSI_SYNC_CLK parameter is set to 0; otherwise, DW_apb_ssi uses DW_apb_ssi_bcm06 component which is synchronous with the single clock FIFO controller.

Figure A-2 shows the block diagram of Synchronizer 2.

Figure A-2   Synchronizer 2 Block Diagram

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

213

B
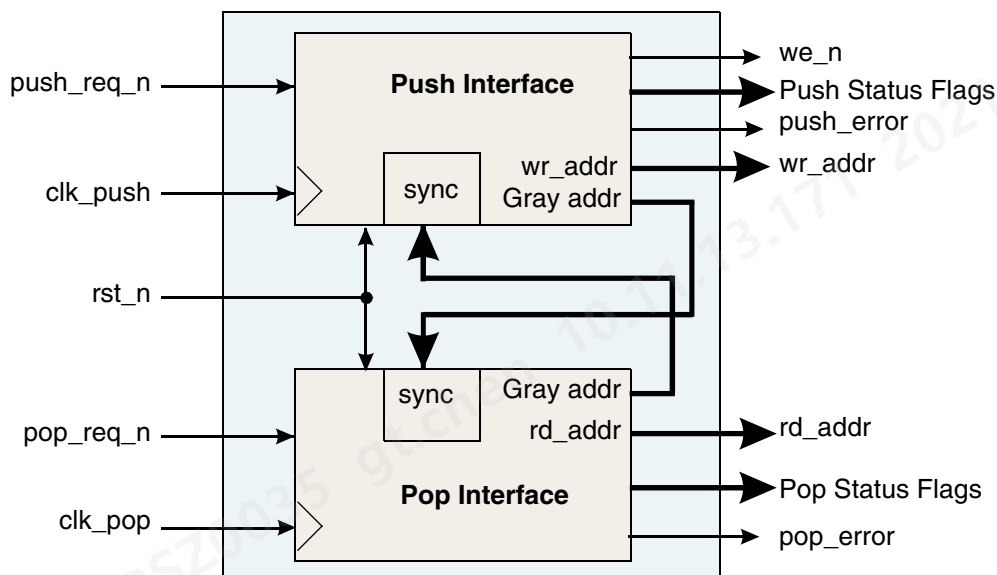
Application Notes

This appendix contains useful "Application Note" information that may be helpful to you in using the DW_apb_ssi component.

B.1   Interfacing DW_apb_ssi and Atmel SPI Devices

Synopsys and Atmel have taken different interpretations of the Motorola SPI serial protocol from the specification. The DW_apb_ssi component is able to communicate with Atmel SPI peripherals but must be programmed slightly differently than with other vendor devices.

The differences between the two devices concern the serial clock phase (SCPH) and serial clock polarity (SCPOL) configuration parameters for the SPI protocol.

B.1.1   Synopsys SPI Operation

When the default serial clock phase is set to logic 0 (SCPH = 0) and the default serial clock polarity is set to logic 0 (SCPOL = 0), the DW_apb_ssi master device toggles the slave select output (ss_0_n) before beginning each new data frame of a continuous serial transfer. This occurs because data transmission starts on the falling edge of the slave select signal. Data is propagated on the negative edge of the serial clock and captured on the positive edge of the serial clock. The inactive state of the serial clock in this mode is logic '0'.

Figure B-1 shows a continuous transfer from a DW_apb_ssi master with the SCPH and SCPOL configuration parameters both set to logic 0.

Figure B-1   DW_apb_ssi SPI: Continuous Transfer where SCPH = 0 and SCPOL = 0
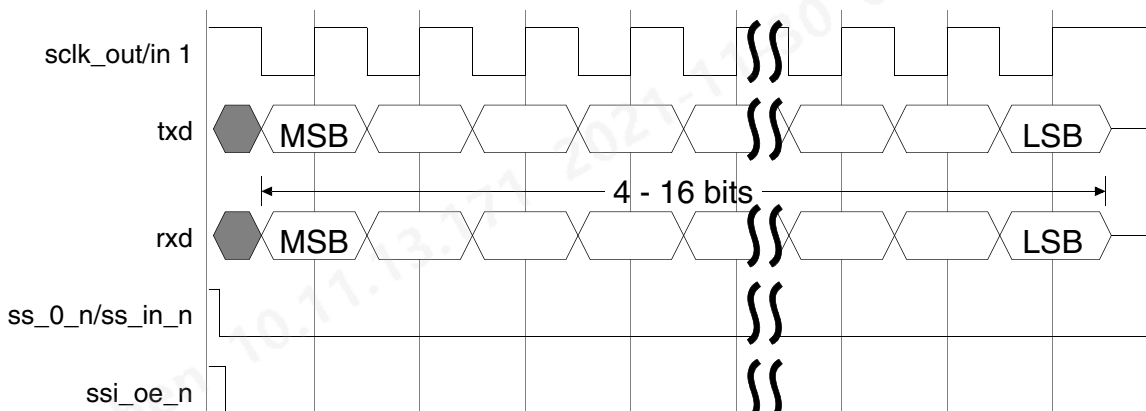

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

215
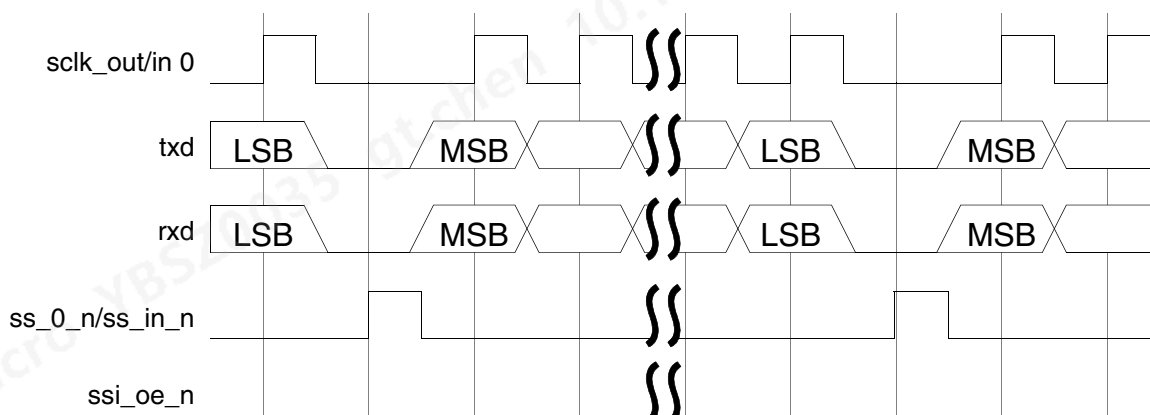
When both the serial clock phase (SCPH) and the serial clock polarity (SCPOL) configuration parameters are set to logic 1, the DW_apb_ssi master device transmits/captures the most significant bit (MSB) of the new data frame directly after the least significant bit (LSB) from the previous data frame. The slave select signal remains active for the duration of the serial transfer. This occurs because data transmission does not begin until the first serial clock edge after the slave select signal is active. Data is propagated on the falling edge of the serial clock and captured on the rising edge of the serial clock. The inactive state of the serial clock in this mode is logic 1.

Figure B-2 shows a continuous transfer from a DW_apb_ssi master with SCPH = 1 and SCPOL = 1.

Figure B-2   DW_apb_ssi SPI: Continuous Transfer where SCPH=1 and SCPOL=1

B.1.2   Atmel SPI Operation

When the Atmel peripheral advertises both the serial clock phase (SCPH) and the serial clock polarity (SCPOL) configuration parameters set to logic 0, it behaves almost exactly like the DW_apb_ssi component when programmed with SCPH = 1 and SCPOL = 1.

The Atmel device transmits/captures the MSB of the new data frame directly after the LSB from the previous data frame. The slave select signal remains active for the duration of the serial transfer. Data is propagated on the falling edge of the serial clock and captured on the rising edge of the serial clock. This behavior matches exactly the behavior of the DW_apb_ssi device when programmed with SCPH and SCPOL set to logic 1.

A timing diagram from an Atmel data sheet (AT25F4096), shown in Figure B-3, illustrates a continuous SPI transfer to one of their SPI memory devices with both SCPH and SCPOL set to logic 0.

Figure B-3   Atmel SPI: Continuous Transfer with SCPH=0 and SCPOL=0
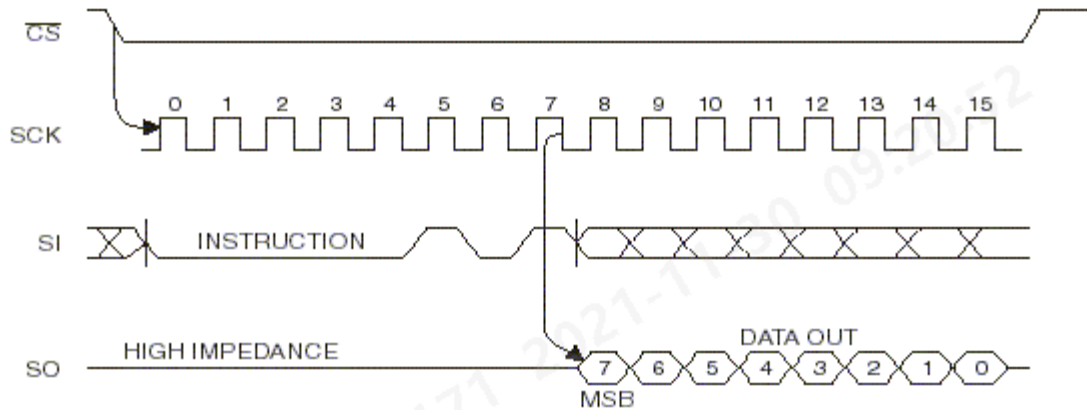
The only visible difference between the Atmel timing diagram and the DW_apb_ssi timing diagram (SCPH = 1, SCPOL = 1) is the inactive level of the serial clock.

B.1.3   Interoperability between DW_apb_ssi and Atmel Devices

In order for the DW_apb_ssi component to communicate with an Atmel peripheral, you must invert the logic on the advertised SCPH and SCPOL parameters when programming the DW_apb_ssi component.

B.2   Interfacing DW_apb_ssi with Dual/Quad Capable Devices

DW_apb_ssi supports Dual/Quad mode of operation in SPI mode. The connection in Dual/Quad mode is different for various devices. In order to comply with all the possible pin-out structure of devices, DW_apb_ssi provides an spi_mode output signal from the device that informs the working mode of the device.

Following are decoding of the SPI modes:

- spi_mode - 00: Normal SPI mode

- spi_mode - 01: Dual SPI mode

- spi_mode - 10: Quad SPI mode

- spi_mode - 11: Reserved for future use

This decoding helps you to perform I/O mapping for any available device.

B.2.1   I/O Connection for A Device That Supports Dual/Quad SPI

This section illustrates how any standard memory device can be connected to DW_apb_ssi using the spi_mode signal. The device taken as an example has an I/O structure similar to the industry standard memories, which support Dual/Quad SPI transfers.

Figure B-4 shows a pin diagram of a typical memory device.

Figure B-4   Pin Diagram for a Memory Device
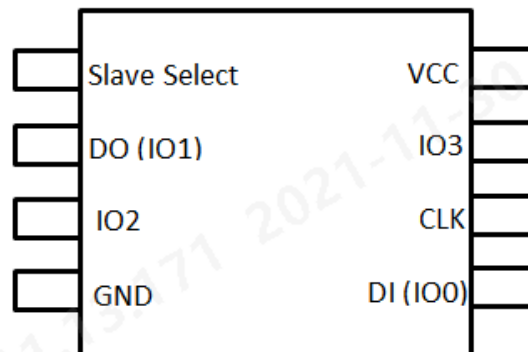
In the Standard mode of operation:

- DI - Represents Master Output Slave Input (MOSI)

- DO - Represents Master Input Slave Output (MISO)

In the Dual/Quad mode of operation, DI becomes IO0 and DO becomes IO1. As illustrated in Figure B-4, these are bi-direction signals and they share the same pins as used in the Standard mode. Additionally, for the Quad mode of operation IO2 and IO3 pins are also available.

Therefore, the DI (IO0) pin acts as data IN for Standard SPI mode and IO0 in Dual/Quad SPI mode. The DO (IO1) pin acts as data OUT in Standard SPI mode and IO1 in Dual/Quad SPI operation.

As signals are shared across different modes of operation, a multiplex logic is required outside the design to route the signal correctly in Standard and Dual/Quad modes.

Figure B-5 shows an example of how the multiplex logic can be implemented using the spi_mode signal for routing the signals.

Figure B-5   Implementation of Multiplex Logic Using the spi_mode Signal

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

219

Synopsys, Inc.

C
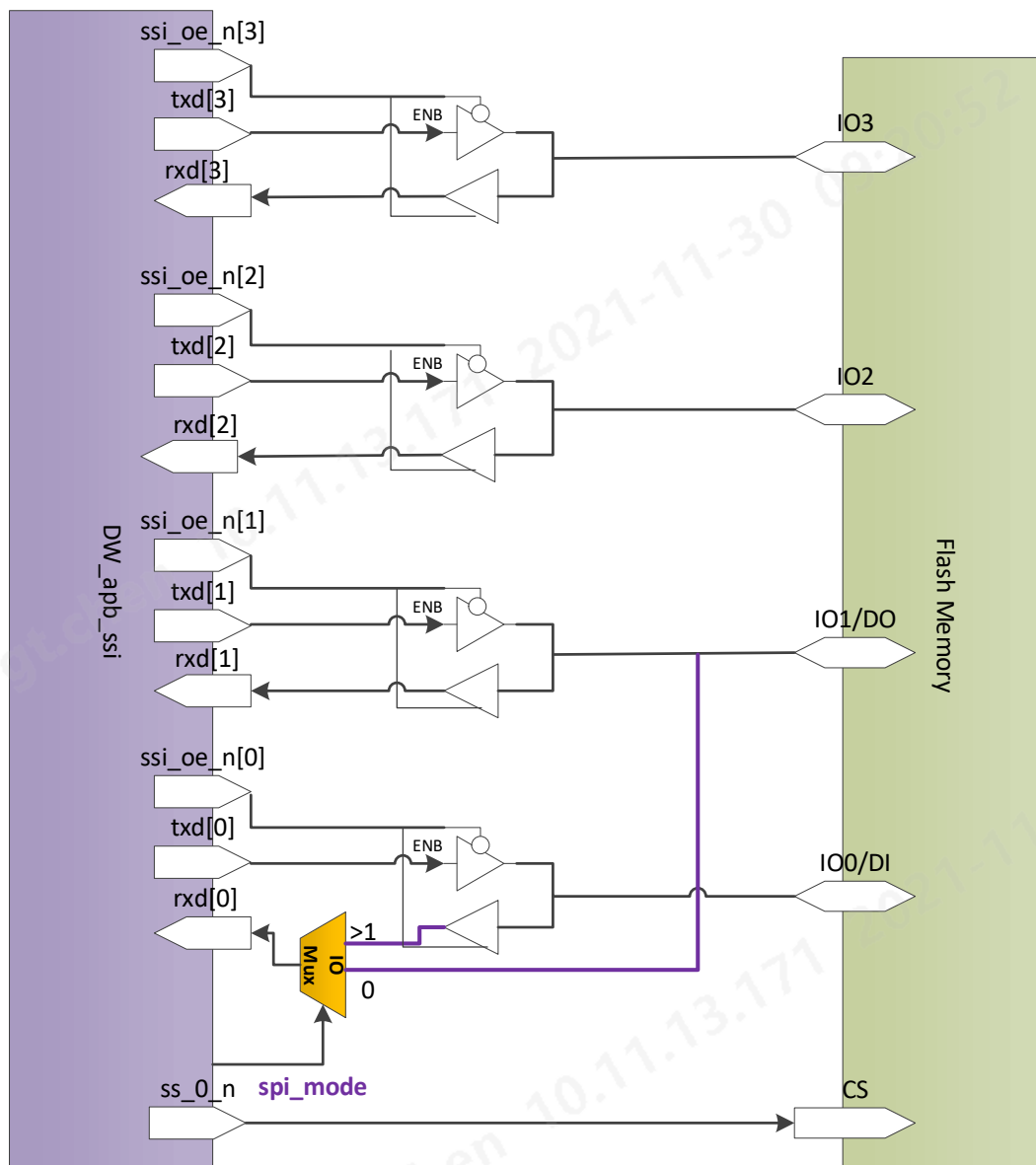
Internal Parameter Descriptions

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 C-1   Internal Parameters

Parameter Name	Equals To
APB_ADDR_WIDTH	= (SSI_XIP_EN == 1 ? 32: 8)
RX_ABW	{[-function_of::::DW_apb_ssi::ssi_calculate_fifo_data_width SSI_RX_FIFO_DEPTH]}
SSI_COMBINED	1
SSI_CTRLR0_RST	= (SSI_SPI_MODE ! = 0 ? (SSI_MAX_XFER_SIZE == 16 ? SSI_CTRLR0_RST_18: SSI_CTRLR0_RST_19) : (SSI_MAX_XFER_SIZE == 16 ? SSI_CTRLR0_RST_16: SSI_CTRLR0_RST_21))
SSI_CTRLR0_RST_16	{SSI_SCPH0_SSTOGGLE_RST,{16{1'b0}} , SSI_DFLT_SCPOL, SSI_DFLT_SCPH, SSI_DFLT_FRF, 4'b0111 }
SSI_CTRLR0_RST_18	{ SSI_SCPH0_SSTOGGLE_RST,1'b0,SSI_DFLT_SPI_FRF, 5'b00000 ,{8{1'b0}} , SSI_DFLT_SCPOL, SSI_DFLT_SCPH, SSI_DFLT_FRF, 4'b0111 }
SSI_CTRLR0_RST_19	{SSI_SCPH0_SSTOGGLE_RST,1'b0,SSI_DFLT_SPI_FRF, 5'b00111 ,{8{1'b0}} , SSI_DFLT_SCPOL, SSI_DFLT_SCPH, SSI_DFLT_FRF, 4'b0000 }

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

221

Table C-1   Internal Parameters (Continued)

Parameter Name	Equals To
SSI_CTRLR0_RST_21	{ SSI_SCPH0_SSTOGGLE_RST,3'b000,5'b00111 ,{8{1'b0}} , SSI_DFLT_SCPOL, SSI_DFLT_SCPH, SSI_DFLT_FRF, 4'b0000 }
SSI_HAS_EXTD_SPI	=(SSI_SPI_MODE == 0 ? 0 : 1)
SSI_INDIVIDUAL	0
SSI_SCPH0_SSTOGGLE_RST	= (SSI_SCPH0_SSTOGGLE ==1 ? 1: 0)
SSI_SPI_MULTIIO	=(SSI_SPI_MODE == 3 ? 8 : (SSI_SPI_MODE == 1 ? 2 : (SSI_SPI_MODE == 2 ? 4 : 1)))
SSI_VERSION_ID	32'~~h3430322a~~h3430332a
TX_ABW	{[-~~function_of:~~::DW_apb_ssi::ssi_calculate_fifo_data_width SSI_TX_FIFO_DEPTH]}

Synopsys, Inc.

D

Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
application design	Overall chip-level design into which a subsystem or subsystems are integrated.
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.
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.
cycle command	A command that executes and causes HDL simulation time to advance.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

223

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.
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.
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.
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.
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.

4.03a
December 2020

Synopsys, Inc.

SolvNetPlus
DesignWare

225

226

SolvNetPlus
DesignWare

Synopsys, Inc.

4.03a
December 2020