



# **DesignWare Cores Synchronous Serial Interface (SSI)**

## **Databook**

---

***Product Codes***

## Copyright Notice and Proprietary Information

© 2023 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

### Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

### Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

### Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at

<https://www.synopsys.com/company/legal/trademarks-brands.html>

All other product or company names may be trademarks of their respective owners.

### Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

### Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.

[www.synopsys.com](http://www.synopsys.com)

# Contents

---

Revision History .....	11
Preface .....	15
Product Codes .....	16
Databook Organization .....	16
Related Documentation .....	16
Web Resources .....	16
Reference Documentation .....	17
Synopsys Statement on Inclusivity and Diversity .....	17
Replacement of Noninclusive Terminology .....	17
Customer Support .....	18
 <b>Chapter 1</b>	
Product Overview .....	21
1.1 General Product Description .....	22
1.2 Standards Compliance .....	23
1.3 Features .....	24
1.4 Verification Environment Overview .....	25
1.5 Area and Power Numbers .....	27
1.6 Controller Requirements .....	28
1.6.1 Clock Requirement .....	28
1.6.2 Reset Requirement .....	28
1.6.3 Memory Requirement .....	28
1.7 Deliverables .....	29
1.8 Where To Go From Here .....	29
 <b>Chapter 2</b>	
Architecture .....	31
2.1 Overview of Architecture .....	33
2.2 Register Bus Interface .....	36
2.2.1 Setting the Register Bus Interface Type .....	36
2.3 Transfer Modes .....	37
2.3.1 Transmit and Receive .....	37
2.3.2 Transmit Only .....	37
2.3.3 Receive Only .....	37

2.3.4 EEPROM Read .....	37
2.4 Partner Connection Interfaces .....	39
2.4.1 Motorola Serial Peripheral Interface (SPI) .....	39
2.4.2 Texas Instruments Synchronous Serial Protocol (SSP) .....	46
2.4.3 National Semiconductor Microwire .....	47
2.5 Data Transfers .....	57
2.6 Serial Controller Mode .....	58
2.6.1 Description of Serial Controller Mode .....	58
2.6.2 Enabling DWC_ssi as a Serial Controller .....	60
2.7 Serial Target Mode .....	61
2.7.1 Description of Serial Target Mode .....	61
2.7.2 Enabling DWC_ssi as a Serial Target .....	62
2.8 Serial Controller and Serial Target Operation (Programmable Mode) .....	63
2.8.1 Description of Programmable Mode .....	63
2.8.2 Enabling DWC_ssi as a Serial Controller and Serial Target .....	63
2.9 Clock Ratios .....	64
2.9.1 Description of Clock Ratios .....	64
2.9.2 Including Enhanced Clock Ratio Architecture .....	68
2.10 Receive and Transmit FIFO Buffers .....	69
2.10.1 Description of Receive and Transmit FIFO Buffers .....	69
2.10.2 External Dual Port SRAM .....	69
2.10.3 Configuring the Receive and Transmit FIFO Buffer Depth .....	72
2.10.4 Registers Related to Receive and Transmit FIFO Buffers .....	72
2.11 RXD Sample Delay .....	74
2.11.1 Description of RXD Sample Delay .....	74
2.11.2 Setting the RXD Delay Value .....	76
2.11.3 Registers Related to RXD Sample Delay .....	77
2.12 DMA Controller Interface .....	78
2.12.1 Description of DMA Controller Interface .....	78
2.12.2 Including DMA Handshaking Interface Signals .....	85
2.12.3 Signals Related to DMA Controller Interface .....	86
2.12.4 Registers Related to DMA Controller Interface .....	86
2.13 Internal DMA Feature for SPI Operation .....	87
2.13.1 Description of DMA Feature for SPI Operation .....	87
2.13.2 SPI Write Operation .....	87
2.13.3 SPI Read Operation .....	90
2.13.4 AXI Manager Interface .....	93
2.13.5 Signals Related to Internal DMA Feature .....	95
2.13.6 Registers Related to Internal DMA Feature .....	96
2.13.7 Enabling Internal DMA Feature .....	96
2.14 SSI Interrupts .....	97
2.14.1 Description of SSI Interrupts .....	97
2.14.2 Description of Safety Interrupts .....	98
2.14.3 Configuring Interrupt Pinout .....	100
2.14.4 Signals Related to SSI Interrupts .....	100
2.14.5 Registers Related to SSI Interrupts .....	100
2.15 Boot Mode .....	101
2.15.1 Description of Boot Mode .....	101
2.15.2 Enabling Boot Mode .....	101

2.16 Enhanced SPI Modes .....	102
2.16.1 Write Operation in Enhanced SPI Modes .....	102
2.16.2 Read Operation in Enhanced SPI Modes .....	106
2.16.3 Variable Chip-select HIGH Period between back-to-back Transfers .....	109
2.16.4 Enabling the Required SPI Mode .....	110
2.16.5 Registers Related to Enhanced SPI Modes .....	110
2.17 Enhanced SPI Operation in Target Mode .....	111
2.17.1 Write Operation .....	111
2.17.2 Read Operation .....	111
2.18 Clock Stretching in Enhanced SPI Transfers .....	113
2.18.1 Description of Clock Stretching in Enhanced SPI Transfers .....	113
2.18.2 Enabling the Clock Stretching in Enhanced SPI Transfers .....	114
2.18.3 Registers Related to Clock Stretching in Enhanced SPI Transfers .....	114
2.19 SPI Dynamic Wait States Feature .....	115
2.19.1 Dynamic Wait States in Legacy SPI Mode .....	115
2.19.2 Dynamic Wait States in Enhanced SPI Mode .....	117
2.19.3 Enabling the Dynamic Wait States Feature .....	119
2.19.4 Registers Related to Dynamic Wait States Feature .....	119
2.20 Dual Data-Rate (DDR) Support .....	120
2.20.1 Description of Dual Data-Rate (DDR) Support .....	120
2.20.2 Transmitting Data in DDR Mode .....	121
2.20.3 Enabling DDR Transfer in SPI Frame Format .....	122
2.20.4 Registers Related to Dual Data-Rate (DDR) .....	122
2.21 Read Data Strobe Signal Support .....	124
2.21.1 Variable Latency Support .....	125
2.21.2 Enabling Read Data Strobe Signal .....	126
2.21.3 Signals Related to Read Data Strobe Signal .....	127
2.21.4 Registers Related to Read Data Strobe Signal .....	127
2.22 Clock Loop Back Support .....	128
2.22.1 Design for Test .....	130
2.23 Data Mask .....	131
2.23.1 Description of Data Mask .....	131
2.23.2 Enabling Data Mask .....	132
2.23.3 Signals Related to Data Mask .....	132
2.23.4 Registers Related to Data Mask .....	132
2.24 HyperBus Protocol Support .....	133
2.24.1 Description of HyperBus Protocol Support .....	133
2.24.2 Read Transactions .....	134
2.24.3 Write Transactions .....	134
2.24.4 Setting the CTRLR0 Register to Enable the HyperBus Feature .....	135
2.24.5 RXDS Signaling Support in Command-Address Phase .....	136
2.24.6 HyperBus 2.0 and 2.0e Support .....	136
2.24.7 HyperBus Connections .....	138
2.24.8 Enabling the HyperBus Feature .....	139
2.24.9 Registers Related to HyperBus Feature .....	139
2.25 JEDEC xSPI Protocol Support .....	140
2.26 SPI Bridge Feature .....	143
2.26.1 Instruction Phase .....	144
2.26.2 Address Phase .....	145

2.26.3 Data Phase .....	145
2.26.4 Bridge Operation .....	145
2.26.5 External SPI Controller Expectations .....	155
2.26.6 AHB Manager Interface .....	156
2.26.7 SPI Interface Timings in Bridge Mode .....	157
2.27 eXecute In Place (XIP) Mode .....	159
2.27.1 XIP Read Usage Model .....	159
2.27.2 XIP Transfers .....	160
2.27.3 AHB WAIT Transfers .....	164
2.27.4 Early Burst Termination .....	166
2.27.5 Hardcode Data Frame Size for XIP Read Transfer .....	166
2.27.6 Mode Bits Support .....	169
2.27.7 XIP Write Model .....	170
2.27.8 Hardcode DFS Support for XIP Write .....	173
2.27.9 Data Mask Support for XIP Write .....	176
2.27.10 External Chip Select Input / or XIP Transfers .....	177
2.27.11 XIP Transfers in HyperBus Mode .....	177
2.27.12 Enabling the XIP Feature in SPI Mode .....	178
2.27.13 Signals Related to eXecute In Place (XIP) Mode .....	178
2.27.14 Registers Related to eXecute In Place (XIP) Mode .....	178
2.28 Concurrent XIP and Non-XIP Operation .....	180
2.28.1 Description of Concurrent XIP and Non-XIP Operation .....	180
2.28.2 Enabling Concurrent XIP and Non-XIP Operation .....	181
2.28.3 Registers Related to Concurrent XIP and Non-XIP Operation .....	181
2.29 Continuous Transfer Mode in XIP .....	182
2.29.1 Description of Continuous Transfer Mode in XIP .....	182
2.29.2 Continuous Incremental Transfer .....	182
2.29.3 Continuous WRAP Transfer .....	183
2.29.4 Data Sampling in Continuous Transfer Mode .....	184
2.29.5 Serial Chip De-select in Continuous Transfer Mode .....	185
2.29.6 Enabling Continuous Transfer Mode in XIP Mode .....	186
2.29.7 Registers Related to Continuous Transfer Mode in XIP .....	186
2.30 Data Pre-fetch in XIP Operations .....	187
2.30.1 Description of Data Pre-fetch in XIP Operations .....	187
2.30.2 Enabling Data Pre-fetch in XIP Operations .....	190
<b>Chapter 3</b> <b>Automotive Safety .....</b>	<b>191</b>
3.1 Parity Protection on AHB Subordinate Interface .....	194
3.1.1 Overview of Parity Protection on AHB Subordinate Interface .....	194
3.1.2 Description of Parity Protection on AHB Subordinate Interface .....	194
3.1.3 Address Parity Checker .....	194
3.1.4 Write Data Parity Checker .....	195
3.1.5 Read Data Parity Generation .....	195
3.1.6 Registers Related to Parity Protection on AHB Interface .....	195
3.1.7 Signals Related to Parity Protection on AHB Interface .....	195
3.2 AXI Manager Interface Protection .....	197
3.2.1 Overview of Safety Features on AXI Interface .....	197

3.2.2 Protection for Write Address and Read Address Channels .....	197
3.2.3 Protection for Write Data Channel .....	197
3.2.4 Protection for Read Data Channel .....	197
3.2.5 Protection for Write Response Channel .....	198
3.2.6 AXI Interface ECC Protection for Write and Read Data .....	198
3.2.7 AXI Parity Protection .....	200
3.2.8 AXI Interface ECC and Parity Generator - Register Slice .....	203
3.2.9 AXI Interface VALID and READY Parity Generator and Checker .....	203
3.3 Finite State Machine (FSM) Timeout Protection.....	204
3.3.1 Overview of FSM Timeout Protection .....	204
3.3.2 Description of FSM Timeout Protection .....	204
3.3.3 Error Injection for FSM Timeout Protection .....	204
3.3.4 Registers Related to FSM Timeout Protection .....	205
3.3.5 Signals Related to FSM Timeout Protection .....	205
3.4 FSM One-hot Protection.....	206
3.4.1 Overview of FSM One-hot Protection .....	206
3.4.2 Description of FSM One-hot Protection .....	206
3.4.3 Error Injection for FSM One-hot Protection .....	206
3.4.4 Registers Related to FSM Parity Protection .....	206
3.4.5 Signals Related to FSM Parity Protection .....	206
3.5 Logic Duplication .....	207
3.5.1 Overview for Logic Duplication .....	207
3.5.2 Description of Logic Duplication .....	207
3.5.3 Error Injection for Logic Duplication .....	207
3.5.4 Registers Related to Logic Duplication Feature .....	208
3.5.5 Signals Related to Logic Duplication Feature .....	208
3.6 TMR Protection for the FIFO Controller .....	209
3.6.1 Overview for TMR Protection for TMR Controller .....	209
3.6.2 Description of Logic Duplication .....	209
3.6.3 Registers Related to Logic Duplication Feature .....	209
3.6.4 Signals Related to Logic Duplication Feature .....	209
3.7 On-Chip Data Protection with Parity.....	210
3.7.1 Overview of On-Chip Data Protection with Parity .....	210
3.7.2 Description On-Chip Data Protection with Parity .....	210
3.7.3 Transmit Data Path Parity Protection .....	210
3.7.4 Receive Data Path Parity Protection .....	210
3.7.5 Error Injection in Data Path Parity Protection .....	211
3.7.6 Registers Related to On-Chip Data Protection with Parity .....	212
3.7.7 Signals Related to On-Chip Data Protection with Parity .....	212
3.8 Data Path ECC Protection .....	213
3.8.1 Overview of Data Path ECC Protection .....	213
3.8.2 Description Data Path ECC Protection .....	213
3.8.3 Transmit Data Path ECC Protection .....	213
3.8.4 Receive Path ECC Protection .....	214
3.8.5 Error Injection in Data Path ECC Protection .....	215
3.8.6 Registers Related to On-Chip Data Protection with ECC .....	217
3.8.7 Signals Related to On-Chip Data Protection with ECC .....	218
3.9 Register Space Protection.....	219
3.9.1 Overview of Register Space Protection .....	219

3.9.2 Description of Register Space Protection .....	219
3.9.3 Error Injection for Register Space Protection .....	219
3.9.4 Registers Related to Register Space Protection .....	220
3.9.5 Signals Related to Register Space Protection .....	221
<b>Chapter 4</b>	
<b>Parameter Descriptions .....</b>	<b>223</b>
4.1 Top Level Parameters .....	224
4.2 Top Level Parameters / Software Interface Configuration Parameters .....	228
4.3 Top Level Parameters / External Configuration Parameters .....	229
4.4 Top Level Parameters / Storage Configuration Parameters .....	230
4.5 Safety Features Parameters .....	232
4.6 Internal DMA Parameters .....	235
4.7 Enhanced SPI Parameters .....	236
4.8 Register Default Value Parameters .....	240
4.9 Synchronization Parameters .....	241
<b>Chapter 5</b>	
<b>Signal Descriptions .....</b>	<b>245</b>
5.1 Serial Interface Signals .....	247
5.2 AHB Subordinate Interface Signals .....	252
5.3 APB Completer Interface Signals .....	256
5.4 DMA Interface Signals .....	258
5.5 AHB Manager Interface Signals .....	260
5.6 DFT Interface Signals .....	263
5.7 Debug Interface Signals .....	264
5.8 Single RAM Interface Signals .....	265
5.9 Transmit RAM Interface Signals .....	268
5.10 Receive RAM Interface Signals .....	271
5.11 AXI Manager Interface Signals .....	274
5.12 Interrupt Signals .....	284
<b>Chapter 6</b>	
<b>Register Descriptions .....</b>	<b>299</b>
6.1 DWC_ssi_address_block Registers .....	305
6.1.1 CTRLR0 .....	306
6.1.2 CTRLR1 .....	314
6.1.3 SSIENR .....	315
6.1.4 MWCR .....	316
6.1.5 SER .....	318
6.1.6 BAUDR .....	319
6.1.7 TXFTLR .....	321
6.1.8 RXFTLR .....	323
6.1.9 TXFLR .....	324
6.1.10 RXFLR .....	325
6.1.11 SR .....	326
6.1.12 IMR .....	329
6.1.13 ISR .....	332
6.1.14 RISR .....	335
6.1.15 TXEICR .....	339

6.1.16 RXOICR .....	340
6.1.17 RXUICR .....	341
6.1.18 MSTICR .....	342
6.1.19 ICR .....	343
6.1.20 DMACR .....	344
6.1.21 DMATDLR .....	347
6.1.22 AXIAWLEN .....	348
6.1.23 DMARDLR .....	349
6.1.24 AXIARLEN .....	350
6.1.25 IDR .....	351
6.1.26 SSIC_VERSION_ID .....	352
6.1.27 DR <sub>x</sub> (for i = 0; i <= 35) .....	353
6.1.28 RX_SAMPLE_DELAY .....	354
6.1.29 SPI_CTRLR0 .....	356
6.1.30 DDR_DRIVE_EDGE .....	362
6.1.31 XIP_MODE_BITS .....	363
6.2 DWC_ssi_bridge_block Registers .....	364
6.2.1 CTRLR0 .....	365
6.2.2 SSIENR .....	367
6.2.3 RXFBTR .....	368
6.2.4 TXFTLR .....	369
6.2.5 RXFTLR .....	370
6.2.6 SR .....	371
6.2.7 IMR .....	372
6.2.8 ISR .....	374
6.2.9 RISR .....	376
6.2.10 TXUICR .....	379
6.2.11 RXOICR .....	380
6.2.12 SPIMECR .....	381
6.2.13 AHBECR .....	382
6.2.14 ICR .....	383
6.2.15 IDR .....	384
6.2.16 SSIC_VERSION_ID .....	385
6.3 DWC_ssi_address_block2 Registers .....	386
6.3.1 XIP_INCR_INST .....	387
6.3.2 XIP_WRAP_INST .....	388
6.3.3 XIP_CTRL .....	389
6.3.4 XIP_SER .....	395
6.3.5 XRXOICR .....	396
6.3.6 XIP_CNT_TIME_OUT .....	397
6.3.7 SPI_CTRLR1 .....	398
6.3.8 SPITECR .....	400
6.3.9 SPIDR .....	401
6.3.10 SPIAR .....	402
6.3.11 AXIAR0 .....	403
6.3.12 AXIAR1 .....	404
6.3.13 AXIECR .....	405
6.3.14 DONECR .....	406
6.3.15 XIP_WRITE_INCR_INST .....	407

6.3.16 XIP_WRITE_WRAP_INST .....	408
6.3.17 XIP_WRITE_CTRL .....	409
6.3.18 SAFETYCR .....	413
6.3.19 SLVIFFSMTOCR .....	416
6.3.20 SSIFSMTOCR .....	417
6.3.21 AXIFSMTOCR .....	419
6.3.22 SAFETYISR .....	420
6.3.23 SAFETYICR .....	424
<b>Appendix A</b>	
<b>Power Consumption and Area .....</b>	<b>427</b>
A.1 Power and Area for typical configurations .....	428
A.2 Power and Area for Safety Prime Profiles .....	431
<b>Appendix B</b>	
<b>Serial Interface Timings.....</b>	<b>435</b>
B.1 SSP Protocol Timings.....	436
B.1.1 Serial Controller Mode .....	436
B.1.2 Serial Target Mode .....	437
B.2 Microwire Protocol Timings.....	438
B.2.1 Serial Controller Mode .....	438
B.2.2 Serial Target Mode .....	438
B.3 SPI Protocol Timings .....	440
B.3.1 Serial Controller Mode .....	440
B.3.2 Serial Target Mode .....	442
<b>Appendix C</b>	
<b>Internal Parameter Descriptions.....</b>	<b>445</b>

# Revision History

---



- A change-bar version of this databook is available on the following IP web page under the “Documentation” section:  
[https://www.synopsys.com/dw/ipdir.php?ds=amba\\_ssi](https://www.synopsys.com/dw/ipdir.php?ds=amba_ssi)
  - Links and references to chapters, sections, tables, figures and page numbers in the Revision History table are assured to be valid only for the current version.
- 

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.00a onwards.

Version	Date	Description
2.00a	January 2023	<p><b>Note:</b> Replaced noninclusive and insensitive terms with globally accepted inclusive terminology. See “<a href="#">Replacement of Noninclusive Terminology</a>” on page <a href="#">17</a></p> <p><b>Added:</b></p> <ul style="list-style-type: none"><li>■ “<a href="#">Product Codes</a>” on page <a href="#">16</a></li><li>■ “<a href="#">Reference Documentation</a>” on page <a href="#">17</a></li><li>■ “<a href="#">Synopsys Statement on Inclusivity and Diversity</a>” on page <a href="#">17</a></li><li>■ “<a href="#">Area and Power Numbers</a>” on page <a href="#">27</a></li><li>■ “<a href="#">Controller Requirements</a>” on page <a href="#">28</a></li><li>■ “<a href="#">Deliverables</a>” on page <a href="#">29</a></li><li>■ “<a href="#">Description of Safety Interrupts</a>” on page <a href="#">98</a></li><li>■ “<a href="#">Write Operation in Enhanced SPI Modes</a>” on page <a href="#">102</a></li><li>■ “<a href="#">Dual Octal SPI Read Transfer</a>” on page <a href="#">109</a></li><li>■ “<a href="#">Read Data Strobe Signal Support</a>” on page <a href="#">124</a></li><li>■ “<a href="#">Clock Loop Back Support</a>” on page <a href="#">128</a></li><li>■ “<a href="#">HyperBus 2.0 and 2.0e Support</a>” on page <a href="#">136</a></li><li>■ “<a href="#">Hardcode DFS Support for XIP Write</a>” on page <a href="#">173</a></li><li>■ “<a href="#">Data Mask Support for XIP Write</a>” on page <a href="#">176</a></li></ul> <p><b>Updated:</b></p> <ul style="list-style-type: none"><li>■ “<a href="#">Motorola Serial Peripheral Interface (SPI)</a>” on page <a href="#">39</a></li><li>■ Date and version numbers</li><li>■ Auto-extracted chapters for <i>Parameter Descriptions</i>, <i>Signal Descriptions</i>, <i>Registers Description</i>, and <i>Internal Parameter Descriptions</i> chapters</li><li>■ “<a href="#">Power Consumption and Area</a>” on page <a href="#">427</a></li></ul> <p><b>Removed:</b></p> <ul style="list-style-type: none"><li>■ Moved “<a href="#">Implementation Guidelines</a>” chapter from the databook to the user guide</li></ul>

Version	Date	Description
1.03a	August 2020	<p><b>Added:</b></p> <ul style="list-style-type: none"> <li>■ “HyperBus Connections” on page 138</li> <li>■ “JEDEC xSPI Protocol Support” on page 140</li> <li>■ “Internal DMA Feature for SPI Operation” on page 87</li> <li>■ “XIP Write Model” on page 170</li> <li>■ “External Chip Select Input /or XIP Transfers” on page 177</li> </ul> <p><b>Updated:</b></p> <ul style="list-style-type: none"> <li>■ Restructured “Features” on page 24</li> <li>■ “Overview of Architecture” on page 33</li> <li>■ “Register Bus Interface” on page 36</li> <li>■ Figure 2-14 on page 46</li> <li>■ Figure 2-15 on page 47</li> <li>■ Restructured Serial “Master and Slave Modes”</li> <li>■ “SSIC_ENH_CLK_RATIO = 0” on page 65</li> <li>■ Figure 2-48 on page 74</li> <li>■ Figure 2-49 on page 76</li> <li>■ “Description of Clock Stretching in Enhanced SPI Transfers” on page 113</li> <li>■ “Hardcode Data Frame Size for XIP Read Transfer” on page 166</li> <li>■ Auto-extracted chapters for <i>Parameter Descriptions</i>, <i>Signal Descriptions</i>, <i>Registers Description</i>, and <i>Internal Parameter Descriptions</i> chapters.</li> <li>■ Date and version number</li> </ul> <p><b>Removed:</b></p> <ul style="list-style-type: none"> <li>■ Moved “Synchronizer Methods” chapter contents from the databook to the “Clock Domain Crossing” chapter of the user guide</li> <li>■ Moved “Clocks and Resets” section from the databook to the “Clock Domain Crossing” chapter of the user guide</li> <li>■ Removed the “XIP Mode with Instruction Phase” section</li> </ul>
1.02a	October 2018	<p><b>Added:</b></p> <ul style="list-style-type: none"> <li>■ “External Dual Port SRAM” on page 69</li> <li>■ “Serial Controller and Serial Target Operation (Programmable Mode)” on page 63</li> <li>■ “Enhanced SPI Modes” on page 102</li> <li>■ “SPI Bridge Feature” on page 143</li> <li>■ “SPI Interface Timings in Bridge Mode” on page 157</li> </ul>

Version	Date	Description
1.01a	August 2017	<p><b>Note:</b> All Contents from “Functional Description” chapter has been moved to the “Architecture” chapter.</p> <p><b>Added:</b></p> <ul style="list-style-type: none"> <li>■ <a href="#">“Data Mask” on page 131</a></li> <li>■ <a href="#">“HyperBus Protocol Support” on page 133</a></li> <li>■ <a href="#">“Clock Stretching in Enhanced SPI Transfers” on page 113</a></li> <li>■ <a href="#">“Data Pre-fetch in XIP Operations” on page 187</a></li> <li>■ <a href="#">“Concurrent XIP and Non-XIP Operation” on page 180</a></li> <li>■ <a href="#">“Serial Chip De-select in Continuous Transfer Mode” on page 185</a></li> <li>■ <a href="#">“Power and Area for Maximum Configuration”</a></li> </ul> <p><b>Updated:</b></p> <ul style="list-style-type: none"> <li>■ <a href="#">“Features” on page 24</a></li> <li>■ <a href="#">“RxD Sample Delay” on page 74</a></li> <li>■ <a href="#">“Power Consumption and Area” on page 427</a></li> <li>■ Auto-extracted chapters for <i>Parameter Descriptions</i>, <i>Signal Descriptions</i>, <i>Registers Description</i>, and <i>Internal Parameter Descriptions</i> chapters.</li> </ul> <p><b>Removed:</b></p> <ul style="list-style-type: none"> <li>■ The “DWC_ssi_clk_mux.v Module” section</li> <li>■ The “Functional Description” chapter</li> </ul>
1.00a	August 2016	Initial version

# Preface

---

This databook provides information about DesignWare® Synchronous Serial Interface, referred to as DWC\_ssi throughout the remainder of this databook.

The information in this databook includes a functional description and descriptions of signal, parameters and registers.

This chapter has the following sections:

- “[Product Codes](#)” on page [16](#)
- “[Databook Organization](#)” on page [16](#)
- “[Related Documentation](#)” on page [16](#)
- “[Web Resources](#)” on page [16](#)
- “[Reference Documentation](#)” on page [17](#)
- “[Synopsys Statement on Inclusivity and Diversity](#)” on page [17](#)
- “[Customer Support](#)” on page [18](#)

## Product Codes

This section describes the Product Codes in the following table.

Base License or Add-On	Product Code
<b>Base License</b>	
Synopsys High Performance Synchronous Serial Interface	B858-0
<b>Add-On</b>	
Synopsys SSI Controller DWC SPI to AHB Bridge	D470-0
Synopsys SSI Controller DWC SPI Internal DMA	F801-0
Synopsys SSI Controller x16 lane	H337-0
Synopsys SSI Controller Mission Critical	H106-0

## Databook Organization

The chapters of this databook are organized as follows:

- [Chapter 1, “Product Overview”](#) provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- [Chapter 2, “Architecture”](#) provides an overview of DWC\_ssi architecture.
- [Chapter 3, “Automotive Safety”](#) provides descriptions for automotive features in DWC\_ssi.
- [Chapter 4, “Parameter Descriptions”](#) identifies the configurable parameters supported by the DWC\_ssi.
- [Chapter 5, “Signal Descriptions”](#) provides a list and description of the DWC\_ssi signals.
- [Chapter 6, “Register Descriptions”](#) describes the programmable registers of the DWC\_ssi.
- [Appendix A, “Power Consumption and Area”](#) discusses the power consumption and area for various configurations.
- [Appendix B, “Serial Interface Timings”](#) describes the serial interface timings of DWC\_ssi.
- [Appendix C, “Internal Parameter Descriptions”](#) provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Parameters, Signals, and Registers chapters.

## 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

## Web Resources

For more information, check the following Synopsys online 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>

## Reference Documentation

The following references contain useful information concerning the protocols addressed by this IP:

- AMBA Specification, Revision 2.0, ARM Ltd. for AHB, APB interface
- AMBA4 AXI and ACE protocol specification, February 2013, ARM Ltd for AXI interface
- AMBA3 APB Protocol Specification, v1.0, ARM Ltd for APB3 interface
- eXpanded Serial Peripheral Interface (xSPI) Protocol (JESD251B)
- Cypress Semiconductors Hyperbus Protocol
- Micron XcelaBus Specification

## Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

## Replacement of Noninclusive Terminology

Following is the list and definition of the noninclusive terminology used in this document:

**Table 1 Replacement of Noninclusive Terminology**

Noninclusive Term	Replacement Term	Definition
Master	Manager, Controller, Requester	Device or model that initiates and controls another device or peripheral.
Slave	Subordinate, Target, Completer	Device or model that is controlled by and responds to a manager.



To comply to the AMBA Specifications and to aid in seamless integration, Synopsys will not change parameter, signal and register names.

## Customer Support

To obtain support for your product, prepare the required files and contact the support center using one of the methods described:

- 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**
  - For simulation issues outside of coreConsultant or coreAssembler:
    - Create a waveform 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.

---

- 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**.  
Make sure 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 [mailto:support\\_center@synopsys.com](mailto: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 Overview

---

The Synopsys DesignWare Cores Synchronous Serial Interface (DWC\_ssi) is a configurable, synthesizable, and programmable component that is a full-duplex controller or target synchronous serial interface. This component is part of the family of DesignWare Synthesizable Components. Depending on the configuration, this component can be one of the following:

- AHB Subordinate
- AHB Subordinate and AXI Manager
- AHB Subordinate and AHB Manager
- APB Requester and AHB Manager

This chapter introduces DWC\_ssi and its features. It discusses the following sections:

- “[General Product Description](#)” on page [22](#)
- “[Standards Compliance](#)” on page [23](#)
- “[Features](#)” on page [24](#)
- “[Verification Environment Overview](#)” on page [25](#)
- “[Area and Power Numbers](#)” on page [27](#)
- “[Controller Requirements](#)” on page [28](#)
- “[Deliverables](#)” on page [29](#)
- “[Where To Go From Here](#)” on page [29](#)

## 1.1 General Product Description

DWC\_ssi is a programmable Synchronous Serial Interface (SSI) peripheral. DWC\_ssi is part of family of DesignWare Synthesizable Components. Component conforms to the AMBA Specification, Revision 2.0 and the AMBA AXI Protocol Specification, Version 2.0 from Arm.

The host processor accesses data, control, and status information on the DWC\_ssi through the AHB or APB interface. The DWC\_ssi may also interface with a DMA Controller using an optional set of DMA signals, which can be selected during configuration.

The DWC\_ssi can be configured in one of these modes of operations:

- Serial controller
- Serial target
- Programmable serial controller or target

You can connect, configure, synthesize, and verify the DWC\_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 DWC\_ssi component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

## 1.2 Standards Compliance

The DWC\_ssi component conforms to the [AMBA Specification, Revision 2.0](#) for the subordinate. You must be familiar with this specification.

## 1.3 Features

The following list highlights the features of the DWC\_ssi Synchronous Serial Interface component:

- AHB Subordinate Interface for software programming – allows easy integration into DesignWare Synthesizable components for AMBA 3 implementation
- AHB Manager (AMBA 3 complaint) Interface for SPI bridge operation to convert all the incoming SPI transactions into corresponding AHB transactions
- APB Requester Interface for software programming in SPI bridge configuration
- AXI Manager Interface for Internal DMA transfers
  - AMBA 3 AXI/AMBA 4 AXI-compliant
  - Configurable AXI data width
  - Configurable AXI address width
- Serial-controller or target operation – Enables serial communication with serial-controller or serial-target peripheral devices
- DWC\_SSI supports the following standards:
  - Motorola SPI
    - Standard/Dual/Quad/Octal/Dual Octal SPI
    - JEDEC xSPI (JESD251B) version 1.0
    - XcelaBus by Micron
    - Hyperbus by Cypress Semiconductors
  - Texas Instruments Synchronous Serial Protocol (SSP)
  - National Semiconductors Microwire
- Dynamic Wait State support for SPI transfers
- Execute In Place (XIP) mode support for SPI read and write transfers
- Enhanced (Dual/Quad/Octal/Dual Octal) SPI features:
  - Programmable address, instruction, wait cycles, and data frame size
  - Supported for Controller configurations
  - Clock Loop Back support
  - Dual Data Rate (DDR) support
  - Dual/Quad/Octal SPI support for Target configurations
  - Read Data Strobe support for SDR and DDR transfer
  - Data Mask support for DDR write transfers
  - Clock Stretching support
  - Continuous transfer mode in XIP operation (for reads)
  - Data prefetch support in XIP operation (for reads)
  - Concurrent XIP support
  - Dedicated input pin for target selection in XIP operations

- External DMA controller interface – Enables the DWC\_SSI to interface to a DMA controller over the bus using handshaking interface for transfer requests
- Internal DMA controller – Enables DWC\_ssi to interface with internal AXI subordinates and SPI serial interface without any software intervention
- Multi-controller contention detection – Informs the processor of multiple serial-controller accesses on the serial bus
- Programmable delay on the sample time of received serial data bit (rxd), when configured in serial controller 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 Semiconductors Microwire
  - Clock bit rate - Dynamic control of the serial bit rate of the data transfer under the control of the programmer
  - Data item size (4 to 32 bits) – Item size of each data transfer under control of the programmer
  - Programmable Serial Controller or Target support
- Configurable features:
  - FIFO depth – Configurable depth of the transmit and receive FIFO buffers from 8 to 256 words deep. The FIFO width is fixed at 32 bits
  - External Dual Port RAM interface
  - Number of chip select outputs – When operating as a serial controller, 1 to 16 serial chip-select output signals can be generated
  - Hardware/software chip select – Dedicated hardware -select lines can be used, or software control can be used to target the serial-target device
  - SPI Bridge configuration - for SPI bridge operation to convert all the incoming SPI transactions into corresponding AHB transactions
  - Serial Controller/Target programmable configuration
  - External DMA configuration
  - Internal DMA configuration
  - Combined or individual interrupt lines – You may choose to bring all individual interrupt lines or one combined interrupt line from the DWC\_ssi to the interrupt controller
  - Interrupt polarity – This configuration option selects the active level of the output interrupt lines
  - Register Default values – This configuration option is used to set default values of programming registers directly after reset
  - Boot mode support – device wakes up just after the reset

## 1.4 Verification Environment Overview

The DWC\_ssi includes an extensive verification environment, which sets up and uses Synopsys VCS tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation.

The *DesignWare Cores Synchronous Serial Interface (SSI) User Guide* discusses the specific procedures for verifying DWC\_ssi.

## 1.5 Area and Power Numbers

For information on power and area requirements in DWC\_ssi, see “[Power Consumption and Area](#)” on page [427](#).

## 1.6 Controller Requirements

Provides information on clock, reset and memory requirements of the DWC\_ssi controller.

### 1.6.1 Clock Requirement

For information on clock requirement, see the “Clock Structure” section in the *DesignWare Cores Synchronous Serial Interface (SSI) User Guide*.

### 1.6.2 Reset Requirement

For information on reset requirements, see the “Reset Structure” section in the *DesignWare Cores Synchronous Serial Interface (SSI) User Guide*.

### 1.6.3 Memory Requirement

For more information on memory requirements, see “[External Dual Port SRAM](#)” on page [69](#).

## 1.7 Deliverables

This section describes the contents of the .run file.

The deliverables included in the DWC\_ssi software package help in the integration of the controller with RTL, verification testbenches, IP-XACT component, and so on.

The DWC\_ssi .run file includes the following:

### RTL

Verilog RTL source is provided for the DWC\_ssi controller. You configure the DWC\_ssi in coreConsultant and coreAssembler to generate the RTL.

### Testbench

DWC\_ssi is packaged with the UVM testbench and testcases to verify the configured RTL. For more information on the UVM testbench in DWC\_ssi, refer to *DesignWare Cores Serial Peripheral Interface (DWC\_ssi) User Guide*.

### Packaging

DWC\_ssi is packaged in a DesignWare installation for use with the coreConsultant or the coreAssembler tool.



**Note** The VC VIP library license shipped with DWC\_ssi is intended only to verify the interface level testing of the specific IP configuration out-of-the-box in coreConsultant, to confirm that DWC\_ssi meets the requirements. The VC VIP library license shipped with DWC\_ssi is not intended to be used as a full featured verification model at the subsystem or SOC level.

## 1.8 Where To Go From Here

At this point, you may want to get started working with the DWC\_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 Overview*.

For more information about implementing your DWC\_ssi component within a DesignWare subsystem using coreAssembler, see *DesignWare Cores Synchronous Serial Interface (SSI) Installation Guide*.

For more information about configuring, synthesizing, and verifying just your DWC\_ssi component, see *DesignWare Cores Synchronous Serial Interface (SSI) User Guide*.



# 2

# Architecture

---

This chapter describes the DWC\_ssi architecture and various features supported in DWC\_ssi. The following sections describe how you can configure these features using the coreConsultant tool:

- “Overview of Architecture” on page 33
- “Register Bus Interface” on page 36
- “Transfer Modes” on page 37
- “Partner Connection Interfaces” on page 39
- “Data Transfers” on page 57
- “Serial Controller Mode” on page 58
- “Serial Target Mode” on page 61
- “Serial Controller and Serial Target Operation (Programmable Mode)” on page 63
- “Clock Ratios” on page 64
- “Receive and Transmit FIFO Buffers” on page 69
- “RXD Sample Delay” on page 74
- “DMA Controller Interface” on page 78
- “Internal DMA Feature for SPI Operation” on page 87
- “SSI Interrupts” on page 97
- “Boot Mode” on page 101
- “Enhanced SPI Modes” on page 102
- “Enhanced SPI Operation in Target Mode” on page 111
- “Clock Stretching in Enhanced SPI Transfers” on page 113
- “SPI Dynamic Wait States Feature” on page 115
- “Dual Data-Rate (DDR) Support” on page 120
- “Read Data Strobe Signal Support” on page 124
- “Data Mask” on page 131
- “HyperBus Protocol Support” on page 133
- “JEDEC xSPI Protocol Support” on page 140
- “SPI Bridge Feature” on page 143
- “eXecute In Place (XIP) Mode” on page 159
- “Concurrent XIP and Non-XIP Operation” on page 180

- “[Continuous Transfer Mode in XIP](#)” on page [182](#)
- “[Data Pre-fetch in XIP Operations](#)” on page [187](#)

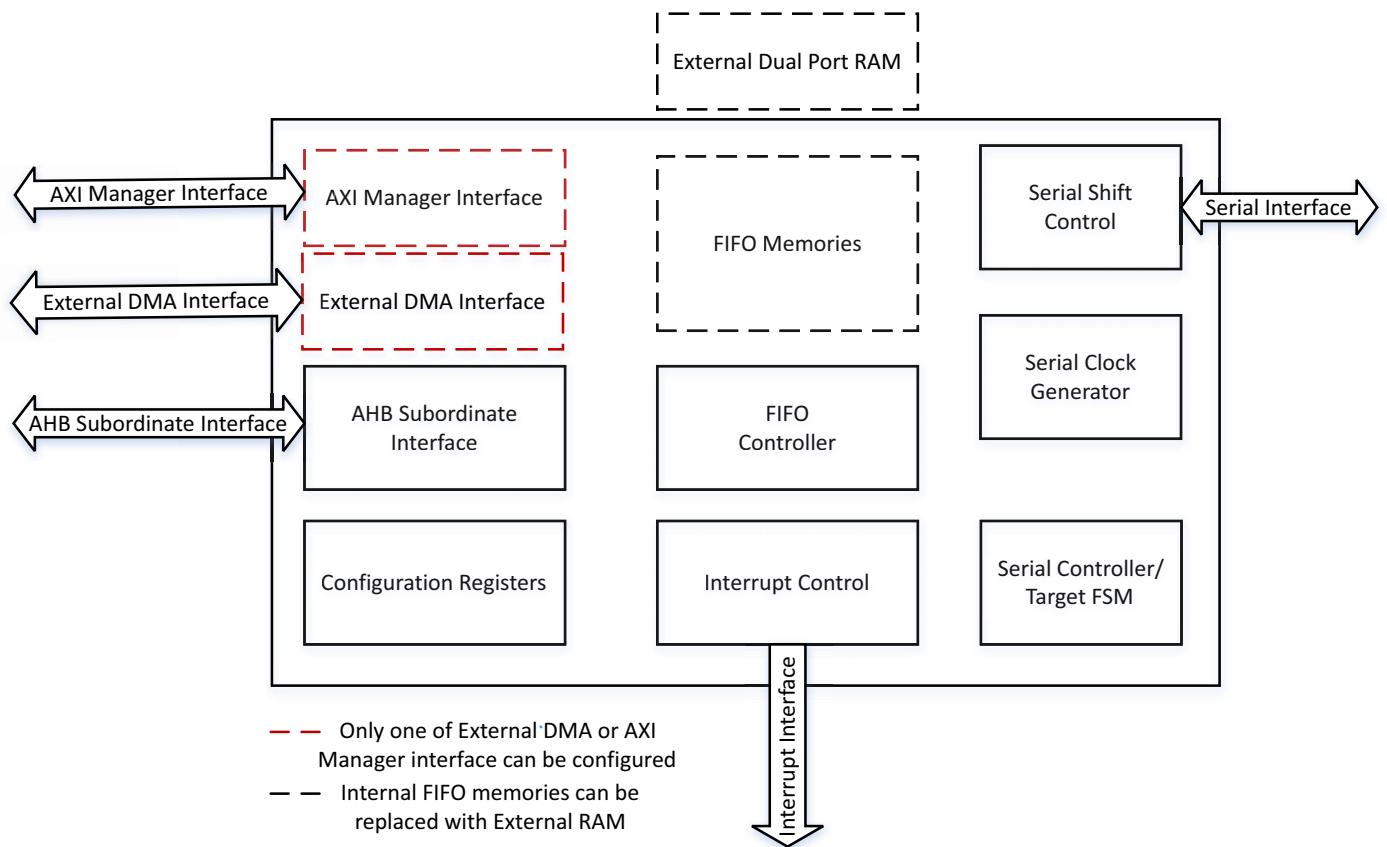
## 2.1 Overview of Architecture

DWC\_ssi can be configured in the following three major modes:

- Serial Controller or Target configuration with AHB subordinate programming interface.
- Serial Controller configuration with AHB subordinate programming interface and AXI Manager interface for data transfers.
- Bridge mode configuration for Serial Target with AHB subordinate/APB completer programming interface and AHB Manager interface for data transfers.

For Serial Controller or Serial Target configuration, the host processor accesses data, control, and status information on DWC\_ssi through the AHB interface. DWC\_ssi might also interface with a DMA controller using an optional set of DMA signals, which can be selected at the time of configuration. In Serial Controller configuration, DWC\_ssi can be configured for internal DMA mode, which uses AXI Manager interface to transmit the data from internal AXI subordinate interface to serial peripheral and vice versa. [Figure 2-1](#) shows the block diagram for the DWC\_ssi block in Serial Controller or Target configuration. Following are the functional blocks in the diagram:

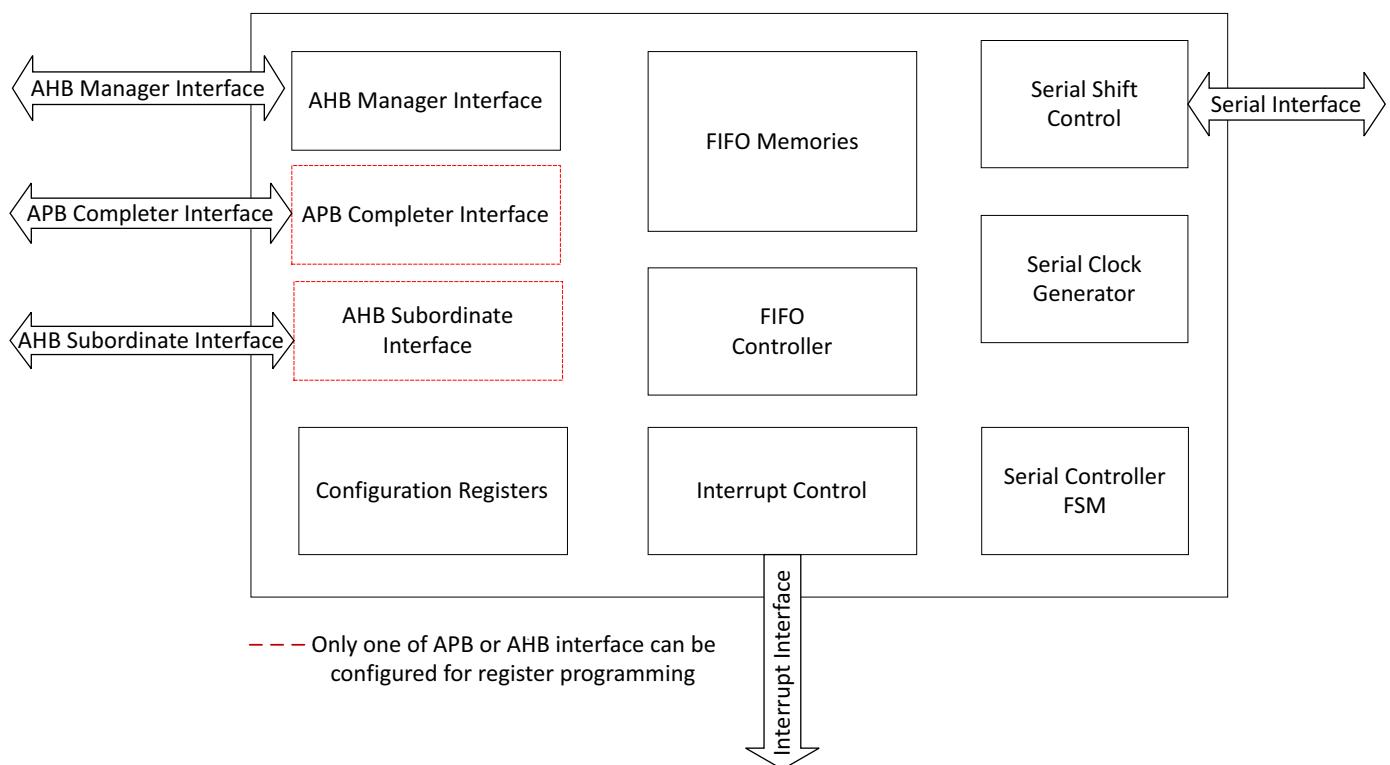
- AHB Subordinate interface for register programming
- Transmit/Receive FIFO controller
- Internal FIFO memories or external RAM for data storage
- Configuration register block
- Shift control
- Serial Controller/Target FSM
- Configurable external or internal DMA interface

**Figure 2-1 DWC\_ssi Block Diagram in Serial Controller/Serial Target Configuration**

DWC\_ssi can connect to any serial-controller or serial-target peripheral device using one of the following interfaces:

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

In bridge mode configuration, host processor accesses status and control information using AHB subordinate interface or APB completer interface, and data transfer happens using the AHB Manager interface. This mode is supported only for the SPI Controller devices. [Figure 2-2](#) shows the block diagram for bridge mode configuration.

**Figure 2-2 DWC\_ssi Block Diagram in SPI Bridge Configuration**
**Note**

- Bridge mode configuration is only supported in Serial Target configuration.
- External RAM is not supported in Bridge configuration.

## 2.2 Register Bus Interface

The Register Bus Interface Module implements the logic to access the internal registers of DWC\_ssi.

DWC\_ssi provides two configurable options for Register Bus interface AHB and APB.

The interface type can be set using the configuration parameter SSIC\_SLVIF\_TYPE. This module supports only the little-endian scheme for data transfer. The Register Bus Interface module supports up to 32-bit data bus:

- APB data width is configured using APB\_DATA\_WIDTH parameter
- AHB data width is fixed to 32 bits

The transfer size should be smaller than or equal to 32 bits. The Register Bus interface is used for both register access and performs Execute in Place operations to the end device.

For more information about the Execute in Place mode, see “[eXecute In Place \(XIP\) Mode](#)” on page [159](#).

DWC\_ssi provides an error response on the programming interface when any of the following conditions are met:

- Invalid Read or Write access on configuration registers
- If transmit FIFO is full, and a write operation is attempted in the Data Register (DR)
- If receive FIFO is empty, and a read operation is attempted in the Data Register (DR)
- If read or write operation is attempted in the Data Register when DWC\_ssi is not enabled (SSIENR register set to 0)
- Under XIP operation (`xip_en == 1`) the following conditions result in error response:
  - If SSIC\_XIP\_WRITE\_EN parameter is set to 0, and any write operation occurs
  - If an XIP read operation is attempted while another Controller is active on the SPI bus (Controller contention)
  - If DW\_ssi is disabled and XIP transaction is attempted
  - If XIP transfer is attempted and the transmit FIFO is not empty
  - the Transmit FIFO underflow interrupt is set and XIP write transfer is attempted



**Note** The error responses on Register Bus interface are not provided in the Serial Target Bridge mode configuration (SSIC\_SPI\_BRIDGE = 1)

### 2.2.1 Setting the Register Bus Interface Type

To set the register bus interface type, select AHB or APB in the **Register Bus Interface Type** field using the **Software Interface Configuration** tab of the **Top Level Parameters** tab during the Specify Configuration activity in coreConsultant.

## 2.3 Transfer Modes

This section describes the different modes in which DWC\_ssi operates when transferring data on the serial bus. The transfer mode (TMOD) is set by writing to control register 0 (CTRLR0), as described in the *Registers Description* chapter.



**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.

### 2.3.1 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.

### 2.3.2 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.

### 2.3.3 Receive Only

When TMOD = 2'b10, the transmit data are invalid. When configured as a serial target, 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.

### 2.3.4 EEPROM Read



**Note** This transfer mode is only valid for serial controller 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 DWC\_ssi serial controller is transmitting data on its txd line, data on the rxd line is ignored). The DWC\_ssi serial controller continues to transmit data until the transmit FIFO is empty. Therefore, you should 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 DWC\_ssi serial controller matches the value of the NDF field in the CTRLR1 register + 1.



**Note** EEPROM read mode is not supported when DWC\_ssi is configured to be in the SSP mode.

---

## 2.4 Partner Connection Interfaces

DWC\_ssi to connect to a serial-controller or serial-target peripheral device, the peripheral must have at least one of the following interfaces:

- “[Motorola Serial Peripheral Interface \(SPI\)](#)” on page 39
- “[Texas Instruments Synchronous Serial Protocol \(SSP\)](#)” on page 46
- “[National Semiconductor Microwire](#)” on page 47

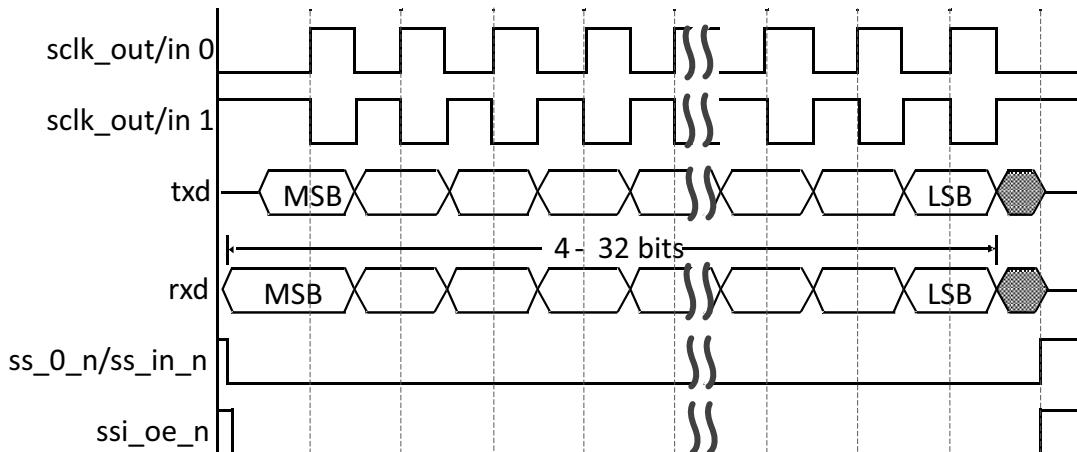
### 2.4.1 Motorola Serial Peripheral Interface (SPI)

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 bits to 32 bits.

When the configuration parameter SCPH = 0, data transmission begins on the falling edge of the chip select signal. The first data bit is captured by the serial controller and serial target 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 2-3](#) shows a timing diagram for a single SPI data transfer with SCPH = 0. The serial clock is shown for configuration parameters SCPOL = 0 and SCPOLE = 1.

**Figure 2-3 SPI Serial Format (SCPH=0)**



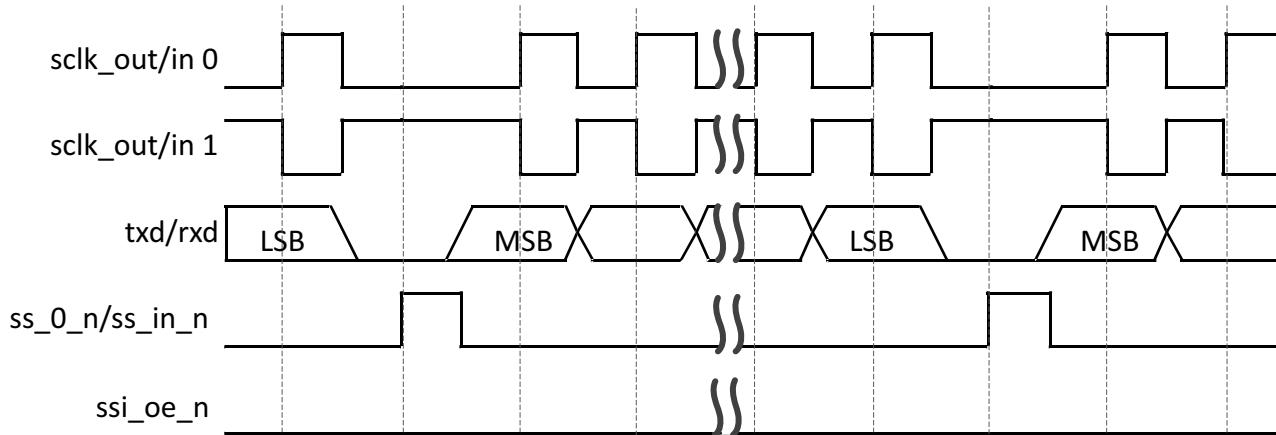
The following signals are illustrated in the timing diagrams in this section:

- `sclk_out` – serial clock from DWC\_ssi serial controller (serial controller configuration only)
- `sclk_in` – serial clock to DWC\_ssi serial target (serial target configuration only)
- `ss_0_n` – chip select signal from DWC\_ssi serial controller (serial controller configuration only)
- `ss_in_n` – chip select input to the DWC\_ssi serial target
- `ss_oe_n` – output enable for the DWC\_ssi serial controller/ target
- `txd` – transmit data line for the DWC\_ssi serial controller/ target
- `rxd` – receive data line for the DWC\_ssi serial controller/ target

Two different modes of continuous data transfers are supported when SCPH = 0; the selection of the desired operation mode is done by configuring CTRLR0 field of the SSTE register:

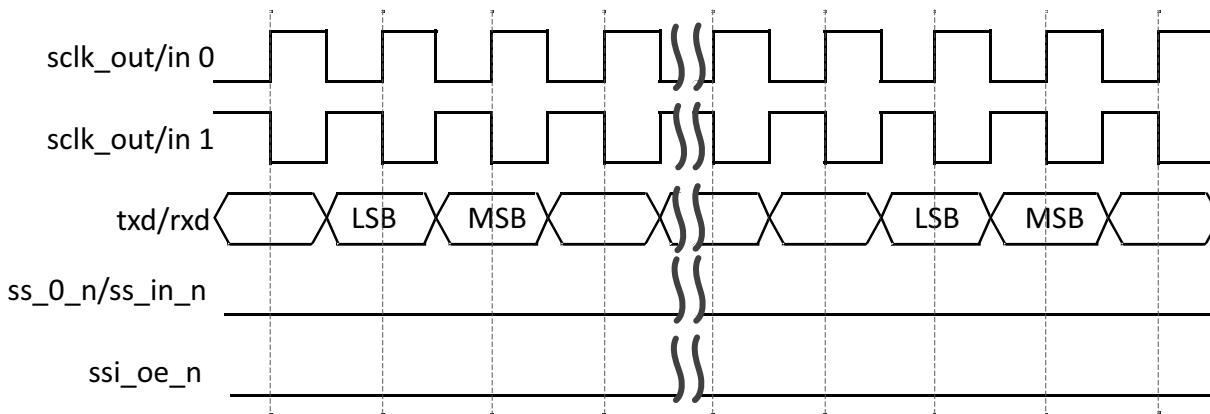
- When CTRLR0 field of the SSTE register is programmed to 1, DWC\_ssi toggles the chip select signal between frames, and the serial clock is held to its default value while the chip select signal is active; this operating mode is illustrated in [Figure 2-4](#).

**Figure 2-4 Serial Format Continuous Transfers (SCPH = 0) when CTRLR0.SSTE = 1**



- When CTRLR0 field of the SSTE register is programmed to 0, the chip select signal stays low and the serial clock runs continuously for the duration of the transfer; this operating mode is illustrated in [Figure 2-5](#).

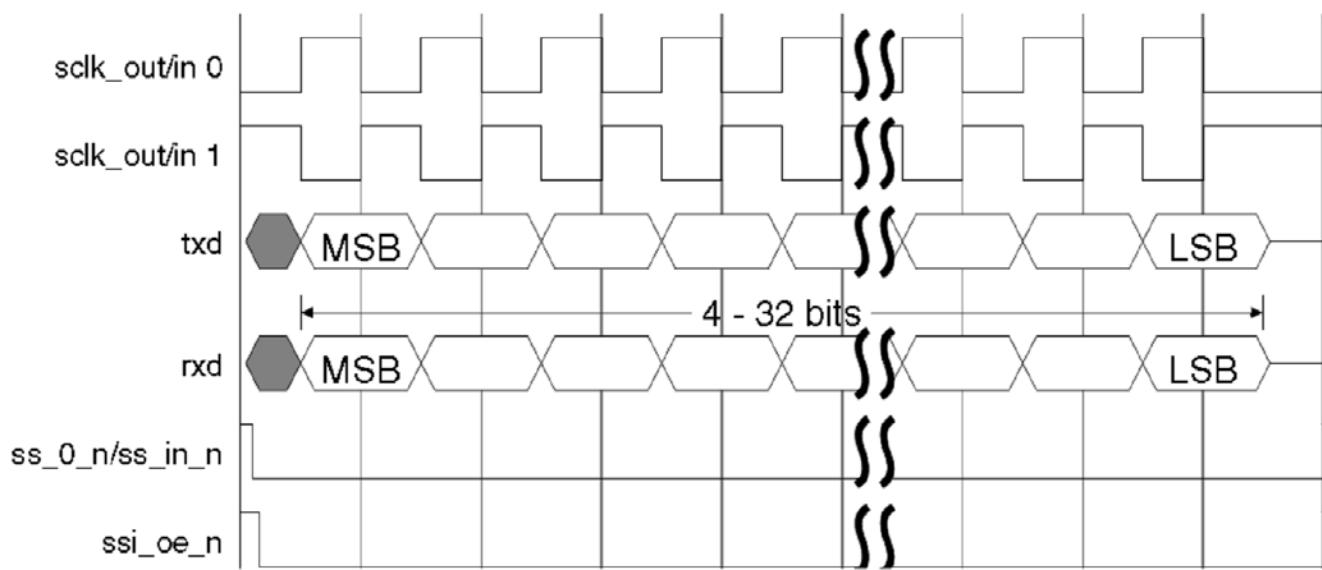
**Figure 2-5 Serial Format Continuous Transfers (SCPH = 0) when CTRLR0.SSTE = 0**



A chip select toggle is not supported when SSIC\_ENH\_CLK\_RATIO parameter is set to 1.

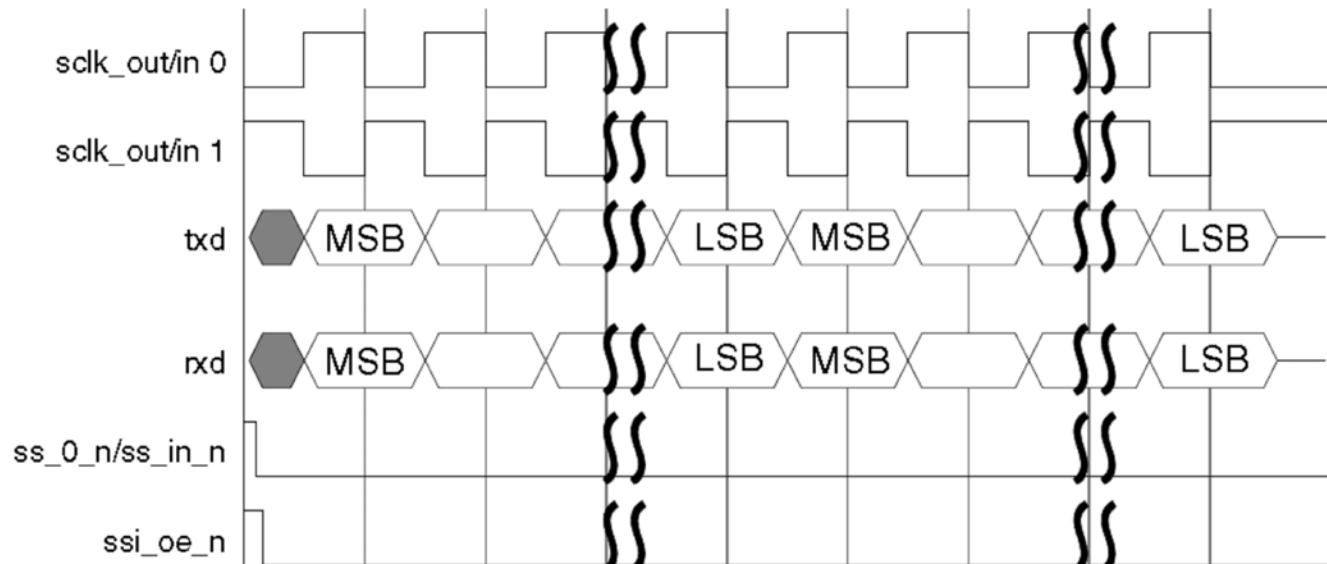
When the configuration parameter SCPH = 1, both serial controller and target peripherals begin transmitting data on the first serial clock edge after the chip select line is activated. The first data bit is captured on the second (trailing) serial clock edge. Data are propagated by the serial controller and target peripherals on the leading edge of the serial clock. During continuous data frame transfers, the chip select line may be held active-low until the last bit of the last frame has been captured.

[Figure 2-6](#) shows the timing diagram for the SPI format when the configuration parameter SCPH = 1.

**Figure 2-6 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 chip select signal is held active for the duration of the transfer.

[Figure 2-7](#) shows the timing diagram for continuous SPI transfers when the configuration parameter  $\text{SCPH} = 1$ .

**Figure 2-7 SPI Serial Format Continuous Transfer (SCPH = 1)**

There are four possible transfer modes on the DWC\_ssi for performing SPI serial transactions; see “[Transfer Modes](#)” on page 37. For *transmit and receive transfers* (transfer mode field (11:10) of the Control Register 0 = 2'b00), data transmitted from the DWC\_ssi to the external serial device is written into the transmit FIFO. Data received from the external serial device into the DWC\_ssi is pushed into the receive FIFO.

DWC\_ssi considers the SPI\_CTRLR0 programming for the standard SPI format. This is applicable when the CTRLR0 . TMOD register field is set to TX\_ONLY (0x1) or RX\_ONLY (0x2). All the fields from the SPI\_CTRLR0 register are applicable for standard SPI format except for DDR related transfers. Following modes are applicable for the standard SPI format:

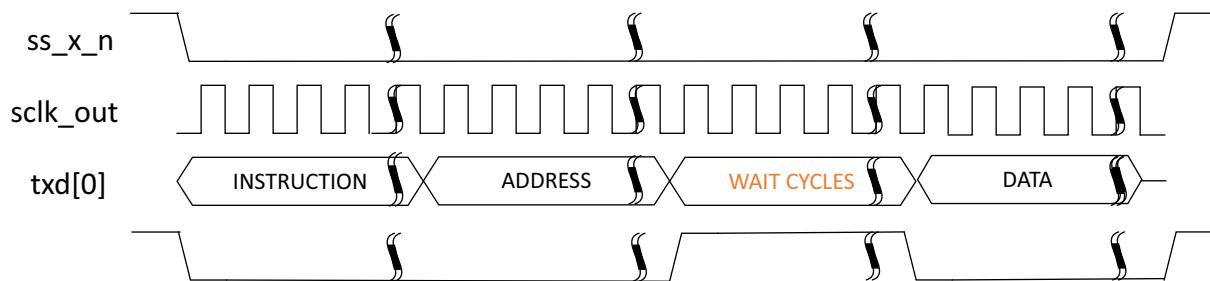
- TX\_AND\_RX (TMOD = 0x0)
- TX\_ONLY (TMOD = 0x1 and SPI\_CTRLR0 . ADDR\_L = 0 and SPI\_CTRLR0 . INST\_L= 0 )
- RX\_ONLY (TMOD = 0x2 and SPI\_CTRLR0 . ADDR\_L = 0 and SPI\_CTRLR0 . INST\_L= 0 )
- EEPROM\_READ (TMOD = 0x3)
- SPI\_WRITE (TMOD = 0x1 and SPI\_CTRLR0 . ADDR\_L!= 0 or SPI\_CTRLR0 . INST\_L!= 0 )
- SPI\_READ (TMOD= 0x2 and SPI\_CTRLR0 . ADDR\_L != 0 or SPI\_CTRLR0 . INST\_L!=0 )

DWC\_ssi considers the new programming fields when TMOD is set to 0x1 or 0x2.



- The SPI\_WRITE and SPI\_READ transfer modes are only applicable when SSIC\_SPI\_MODE != .0.
- DDR transfers are not applicable for Standard SPI transfers.
- Read data strobe is not applicable for Standard SPI transfers.

**Figure 2-8 SPI\_WRITE Transfer Type for Standard SPI Mode**



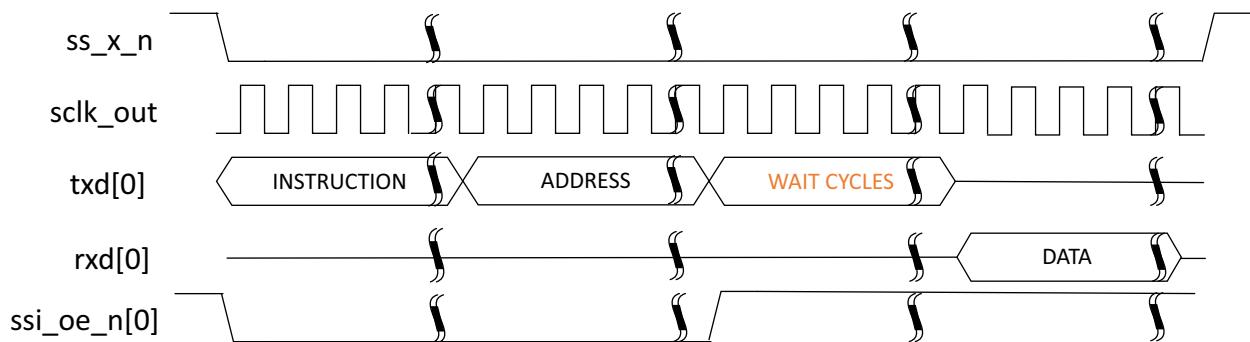
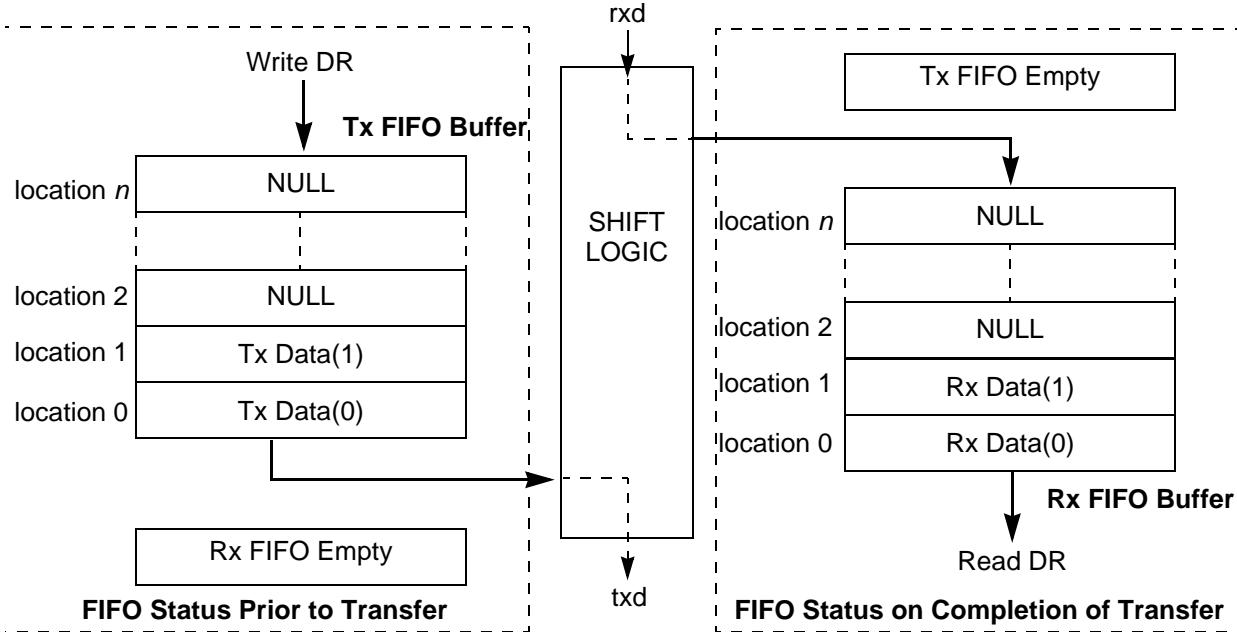
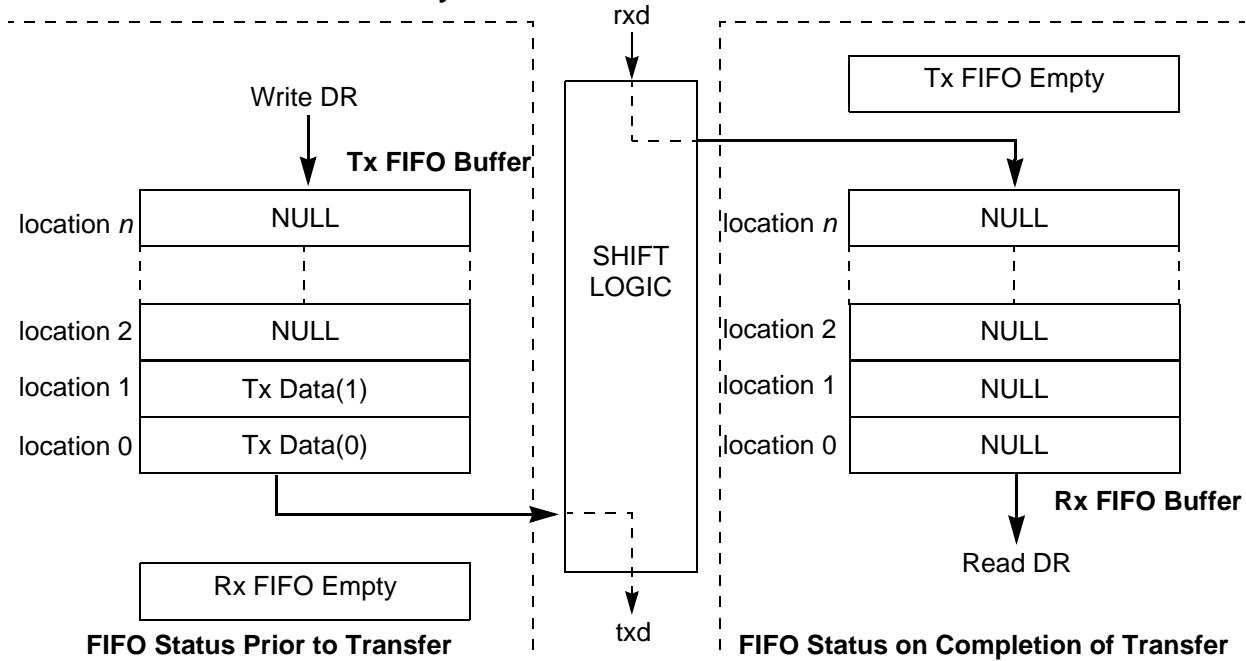
**Figure 2-9 SPI\_READ Transfer Type for Standard SPI Mode**

Figure 2-10 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 DWC\_ssi to the external serial device in a continuous transfer. The external serial device also responds with two data words for the DWC\_ssi.

**Figure 2-10 FIFO Status for Transmit & Receive SPI and SSP Transfers**

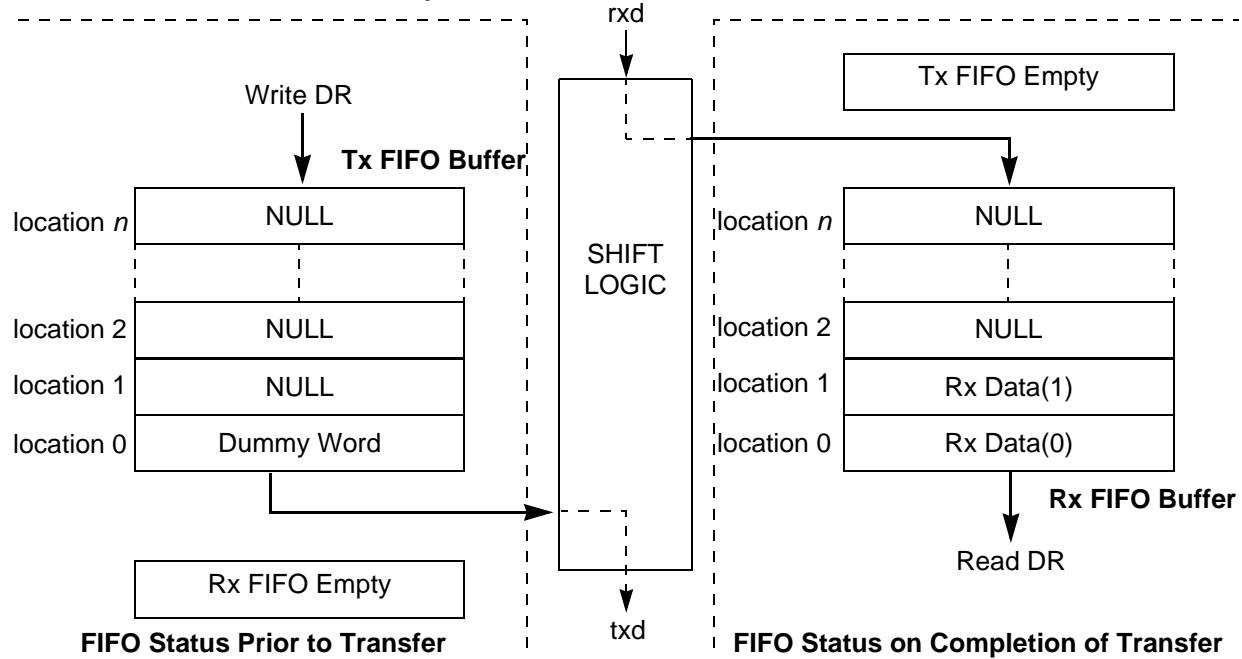
For transmit only transfers (transfer mode field (11:10) of the Control Register 0 = 2'b01), data transmitted from the DWC\_ssi to the external serial device is written into the transmit FIFO. As the data received from the external serial device is deemed invalid, it is not stored in the DWC\_ssi receive FIFO.

Figure 2-11 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 DWC\_ssi to the external serial device in a continuous transfer.

**Figure 2-11 FIFO Status for Transmit Only SPI and SSP Transfers**

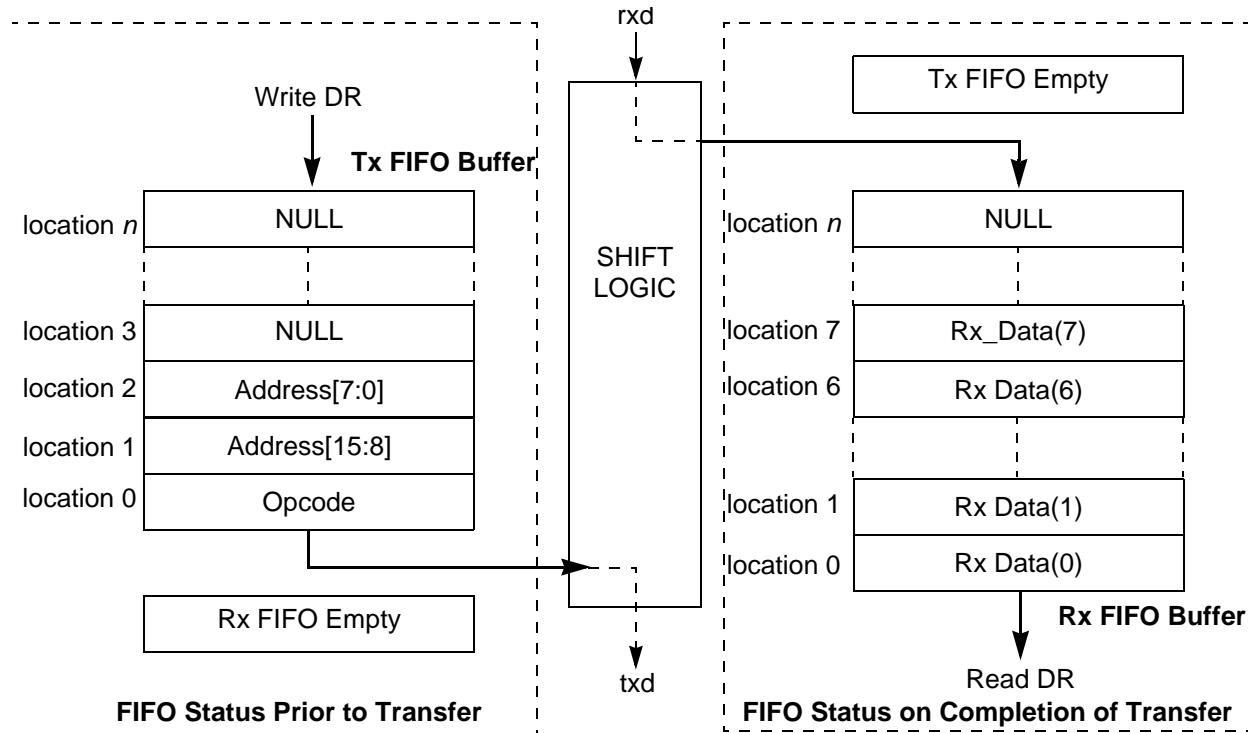
For receive only transfers (transfer mode field (11:10) of the Control Register 0 = 2'b10), data transmitted from the DWC\_ssi to the external serial device is invalid, so a single temporary word is written into the transmit FIFO to begin the serial transfer. The txd output from the DWC\_ssi is held at a constant logic level for the duration of the serial transfer. Data received from the external serial device into the DWC\_ssi is pushed into the receive FIFO.

[Figure 2-12](#) 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 DWC\_ssi from the external serial device in a continuous transfer.

**Figure 2-12 FIFO Status for Receive Only SPI and SSP Transfers**

For eeprom\_read transfers (transfer mode field [11:10] 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 DWC\_ssi serial controller. After the control frames have been transmitted, receive data from the EEPROM device is stored in the receive FIFO.

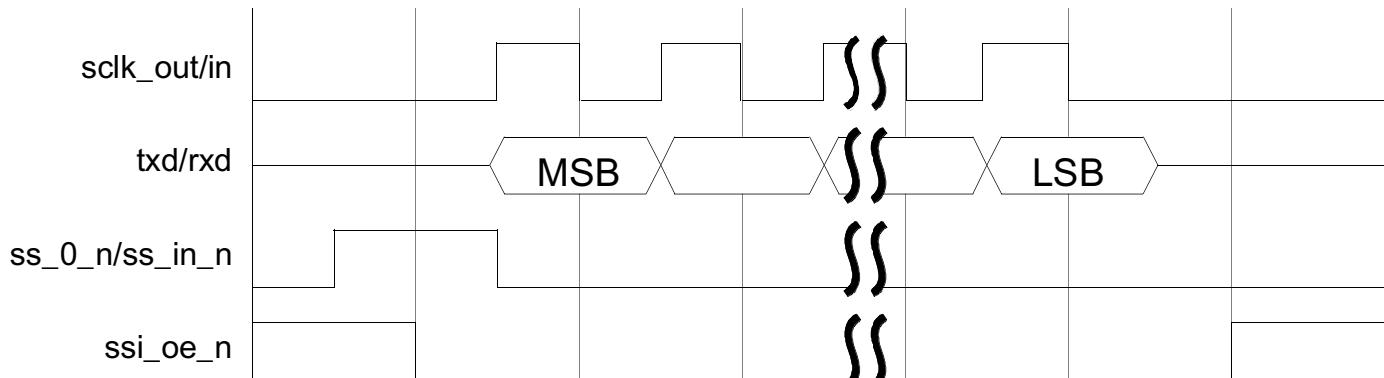
Figure 2-13 shows the FIFO levels prior to the beginning of a serial transfer and the FIFO levels on completion of the transfer. In this example, one opcode and an upper and lower address are transmitted to the EEPROM device, and eight data frames are read from the EEPROM device and stored in the receive FIFO of the DWC\_ssi serial controller.

**Figure 2-13 FIFO Status for EEPROM Read Transfer Mode**

## 2.4.2 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 serial target 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 bits to 32 bits.

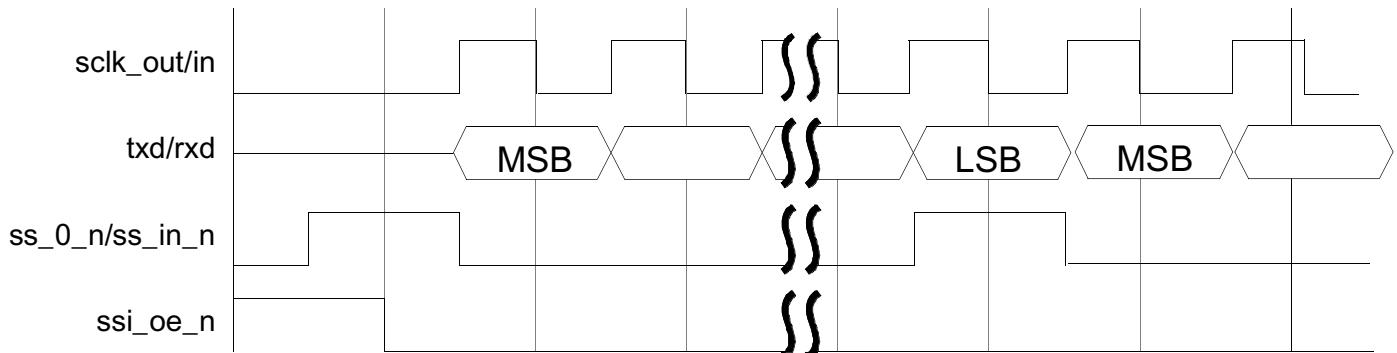
Figure 2-14 shows the timing diagram for a single SSP serial transfer.

**Figure 2-14 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 2-15 shows the timing for a continuous SSP transfer.

**Figure 2-15 SSP Serial Format Continuous Transfer**



### 2.4.3 National Semiconductor Microwire

When the DWC\_ssi is configured as a serial controller, data transmission begins with the falling edge of the chip 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 the CTRLR0 register. The remainder of the control word is transmitted (propagated on the falling edge of **sclk\_out**) by the DWC\_ssi serial controller. During this transmission, no data are present (high impedance) on the serial controller's **rxd** line.

The direction of the data word is controlled by the MDD bit field (bit 1) in the Microwire Control (MWCR) Register. When **MDD** = 0, this indicates that the DWC\_ssi serial controller receives data from the external serial target device. One clock cycle after the LSB of the control word is transmitted, the serial target peripheral responds with a temporary 0 bit, followed by the data frame, which can be 4 bits to 32 bits in length. Data are propagated on the falling edge of the serial clock and captured on the rising edge.

The chip select signal is held active-low during the transfer and is de-asserted one-half clock cycle later, after the data are transferred. Figure 2-16 shows the timing diagram for a single DWC\_ssi serial controller read from an external serial target device.

**Figure 2-16 Single DWC\_ssi Serial Controller Microwire Serial Transfer (MDD = 0)**

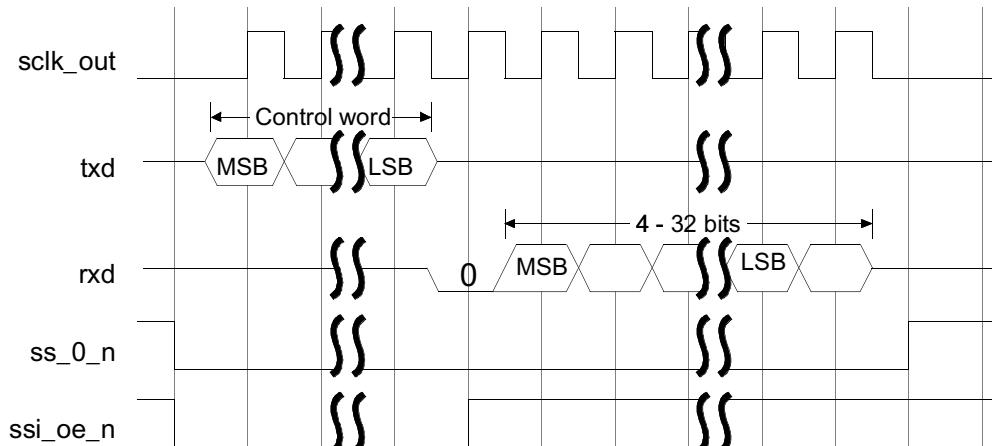
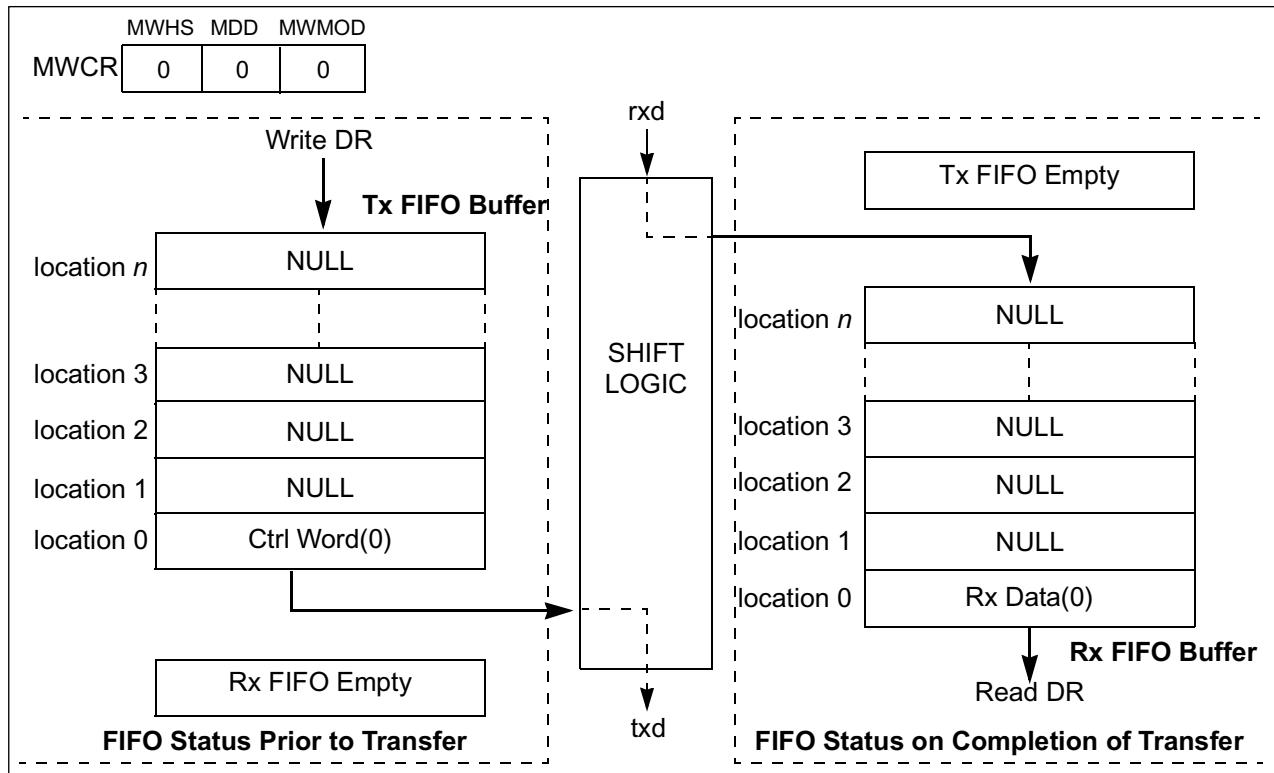


Figure 2-17 shows how the data and control frames are structured in the transmit FIFO prior to the transfer; the value programmed into the MWCR register is also shown.

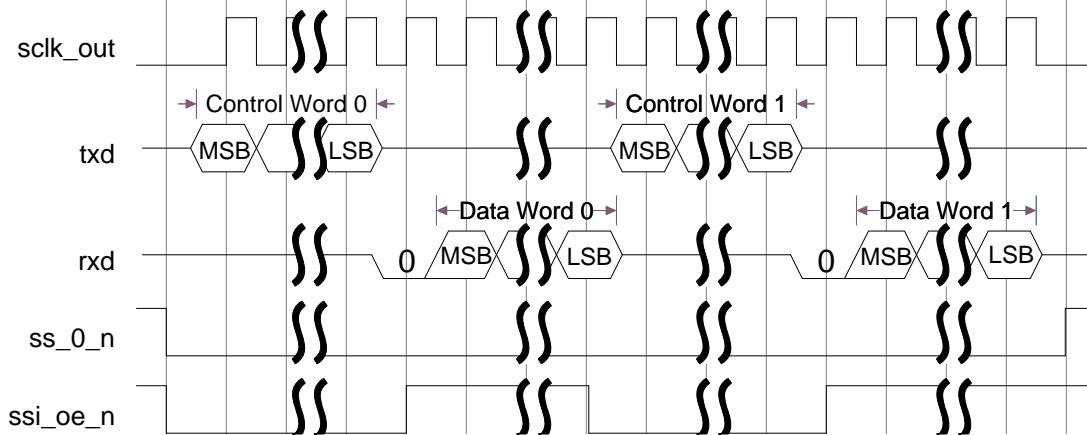
**Figure 2-17 FIFO Status for Single Microwire Transfer (Receiving Data Frame)**



Continuous transfers for the Microwire protocol can be sequential or non-sequential, and are controlled by the MWMOD bit field (bit 0) in the MWCR register.

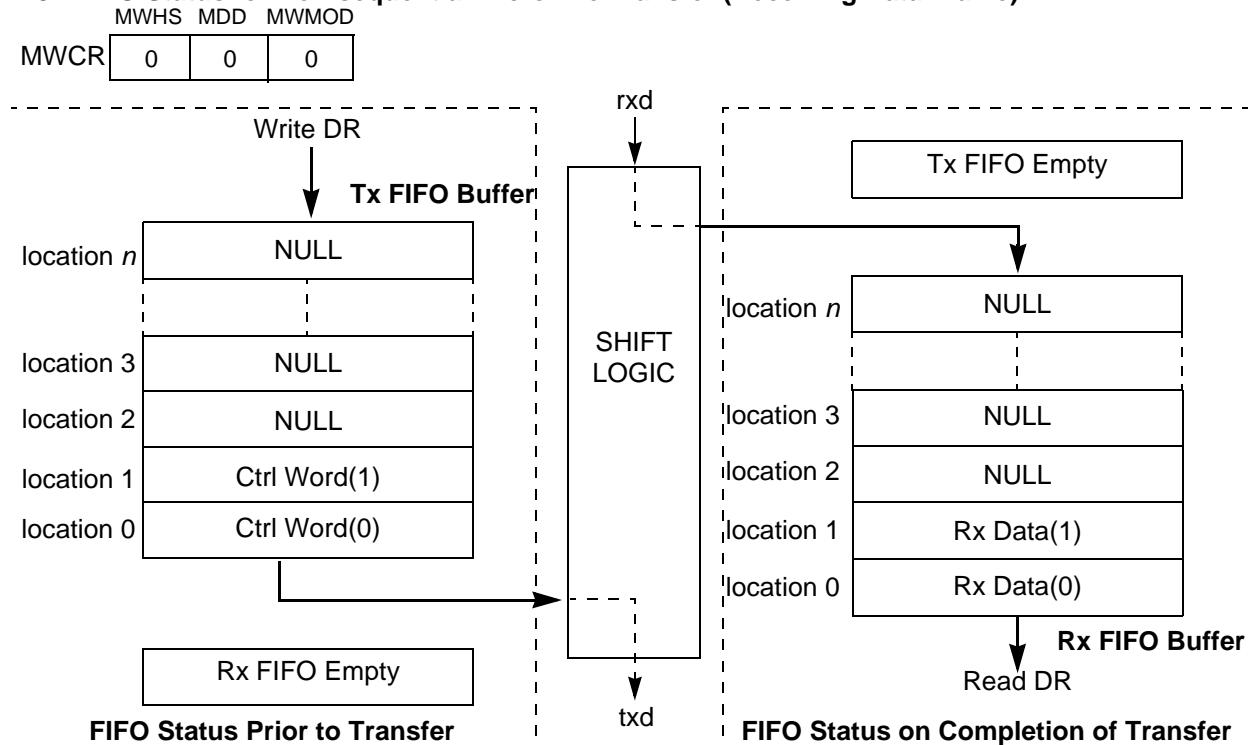
Non-sequential continuous transfers occur as illustrated in Figure 2-18, with the control word for the next transfer following immediately after the LSB of the current data word.

**Figure 2-18 Continuous Non-sequential Microwire Transfer (Receiving Data Frame)**



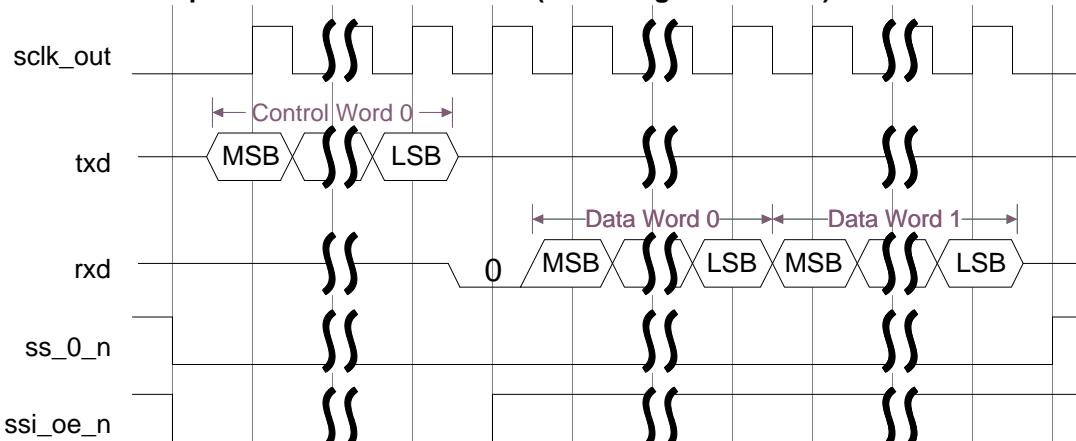
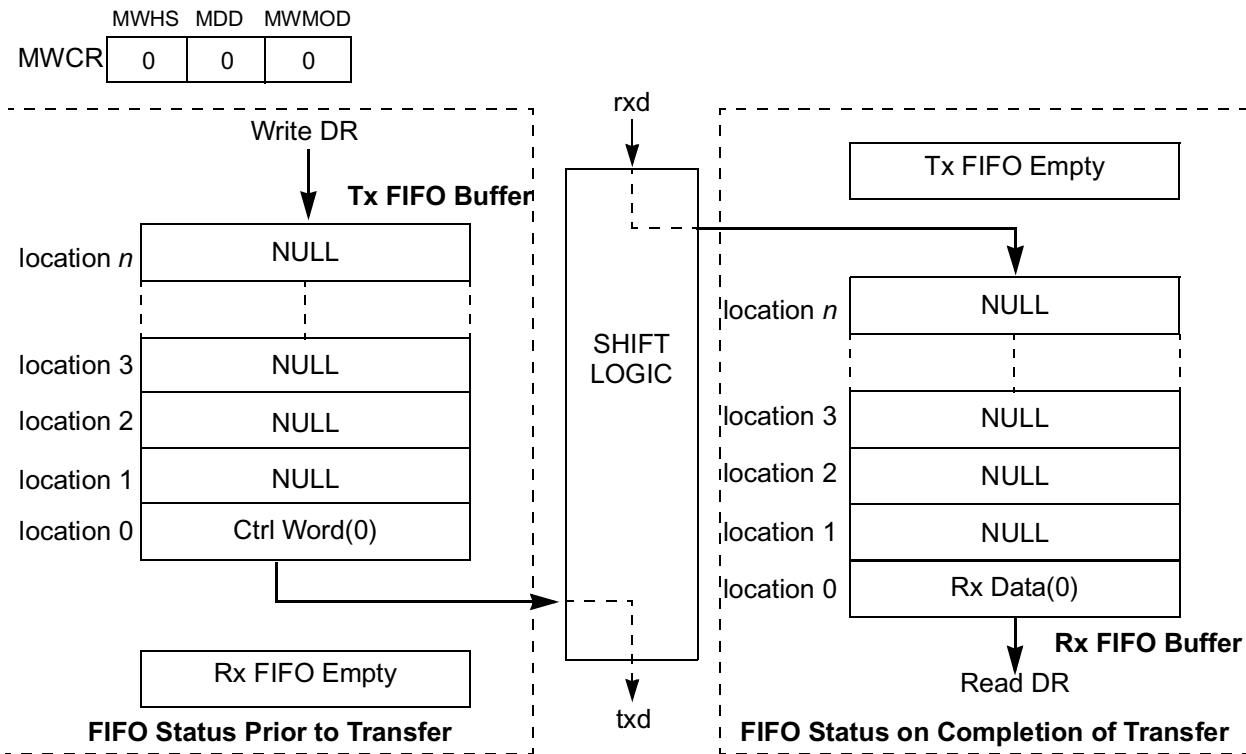
The only modification needed to perform a continuous non-sequential transfer is to write more control words into the transmit FIFO buffer; this is illustrated in [Figure 2-19](#). In this example, two data words are read from the external serial-target device.

**Figure 2-19 FIFO Status for Non-sequential Microwire Transfer (Receiving Data Frame)**



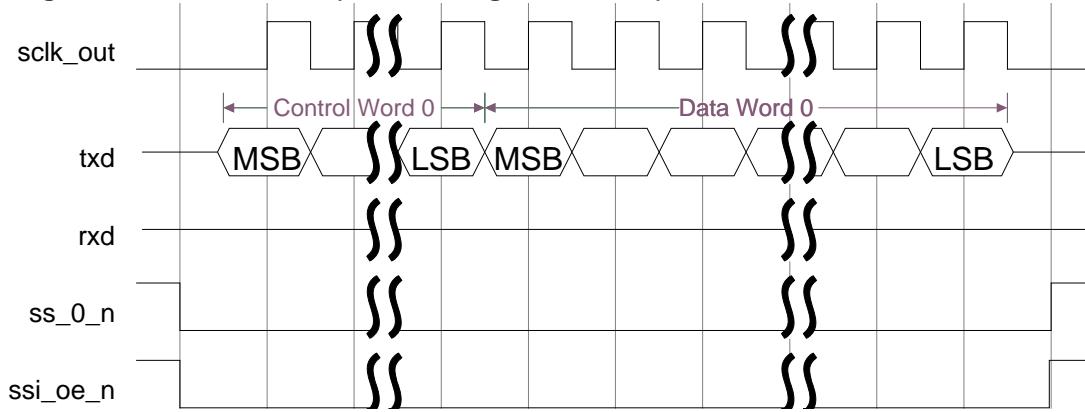
During sequential continuous transfers, only one control word is transmitted from the DWC\_ssi serial controller. The transfer is started in the same manner as with non-sequential read operations, but the cycle is continued to read further data. The serial target device automatically increments its address pointer to the next location and continues to provide data from that location. Any number of locations can be read in this manner; the DWC\_ssi serial controller 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 2-20](#) and example in [Figure 2-21](#) show a continuous sequential read of two data frames from the external serial target device.

**Figure 2-20 Continuous Sequential Microwire Transfer (Receiving Data Frame)****Figure 2-21 FIFO Status for Sequential Microwire Transfer (Receiving Data Frame)**

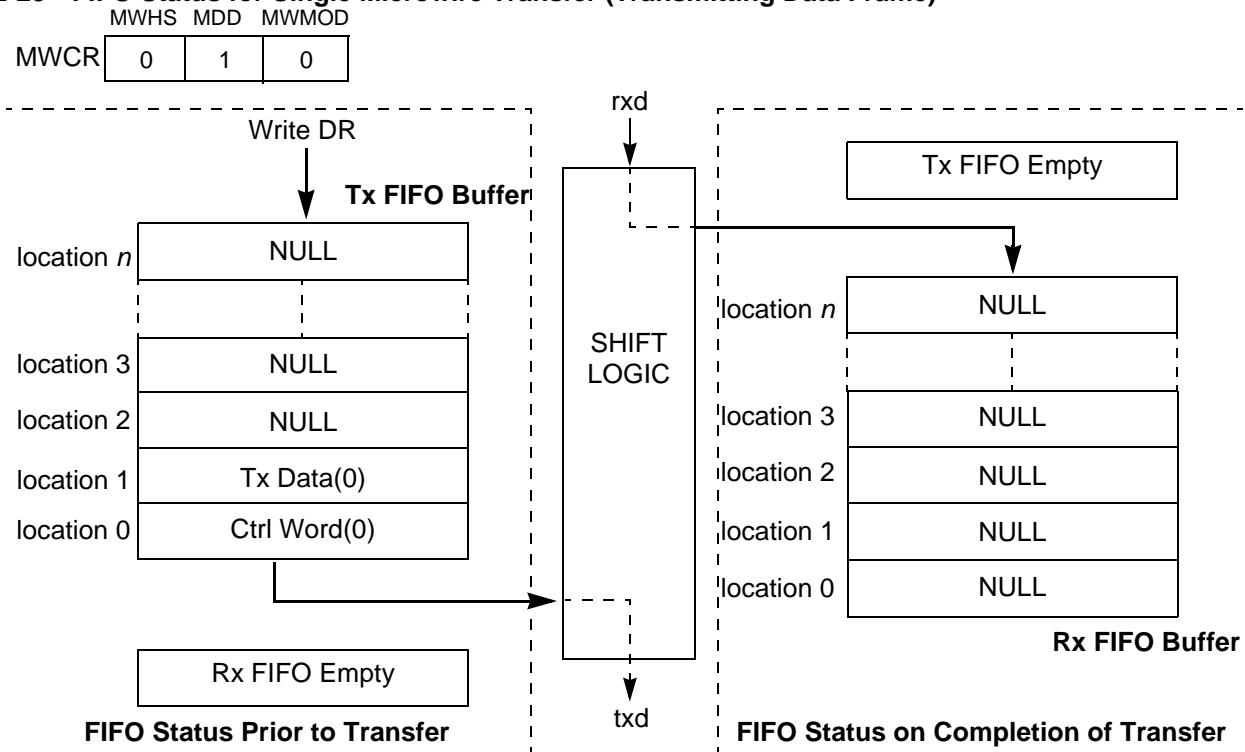
When MDD register is set to 1, this indicates that the DWC\_ssi serial controller transmits data to the external serial target device. Immediately after the LSB of the control word is transmitted, the DWC\_ssi begins transmitting the data frame to the serial target peripheral.

Figure 2-22 shows the timing diagram for a single DWC\_ssi serial controller write to an external serial target device.

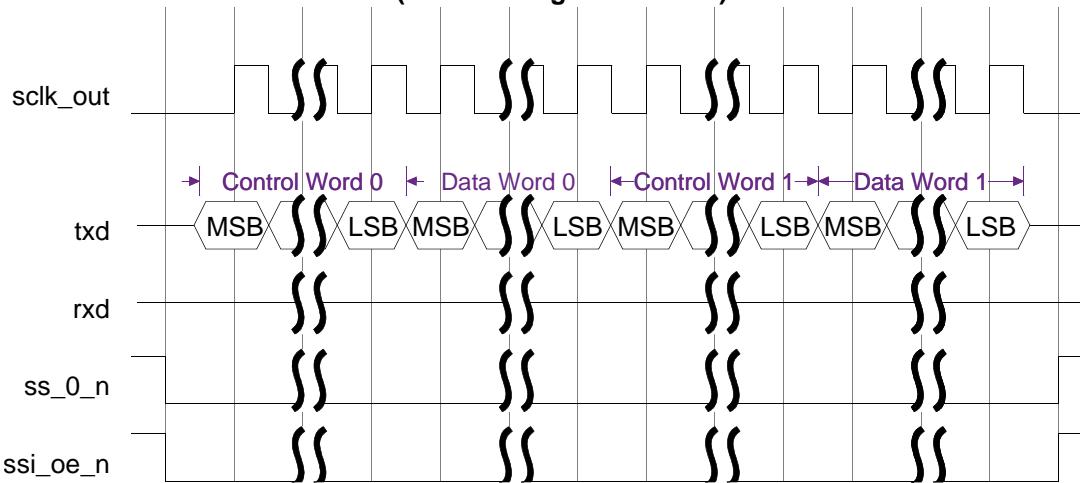
**Figure 2-22 Single Microwire Transfer (Transmitting Data Frame)**

The DWC\_ssi does not support continuous sequential Microwire writes, where MDD = 1 and MWMOD = 1.

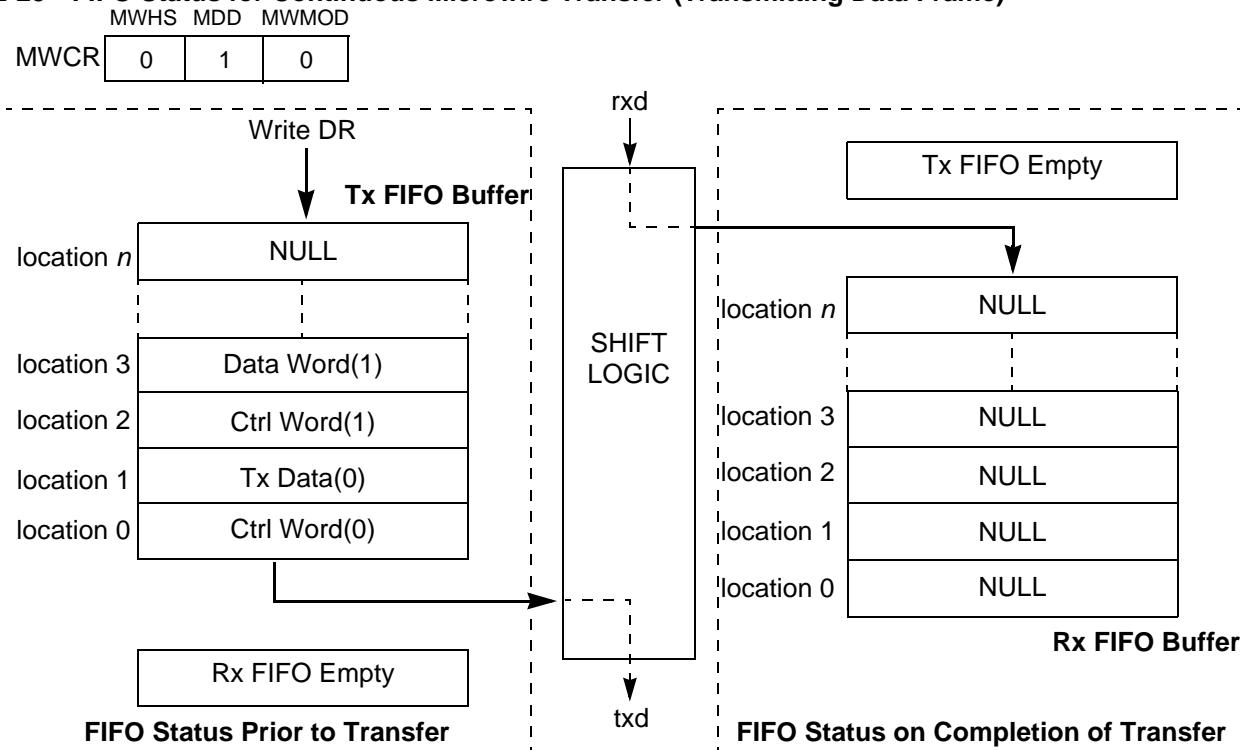
Figure 2-23 shows how the data and control frames are structured in the transmit FIFO prior to the transfer, also shown is the value programmed into the MWCR register.

**Figure 2-23 FIFO Status for Single Microwire Transfer (Transmitting Data Frame)**

Continuous transfers occur as shown in Figure 2-24, with the control word for the next transfer following immediately after the LSB of the current data word.

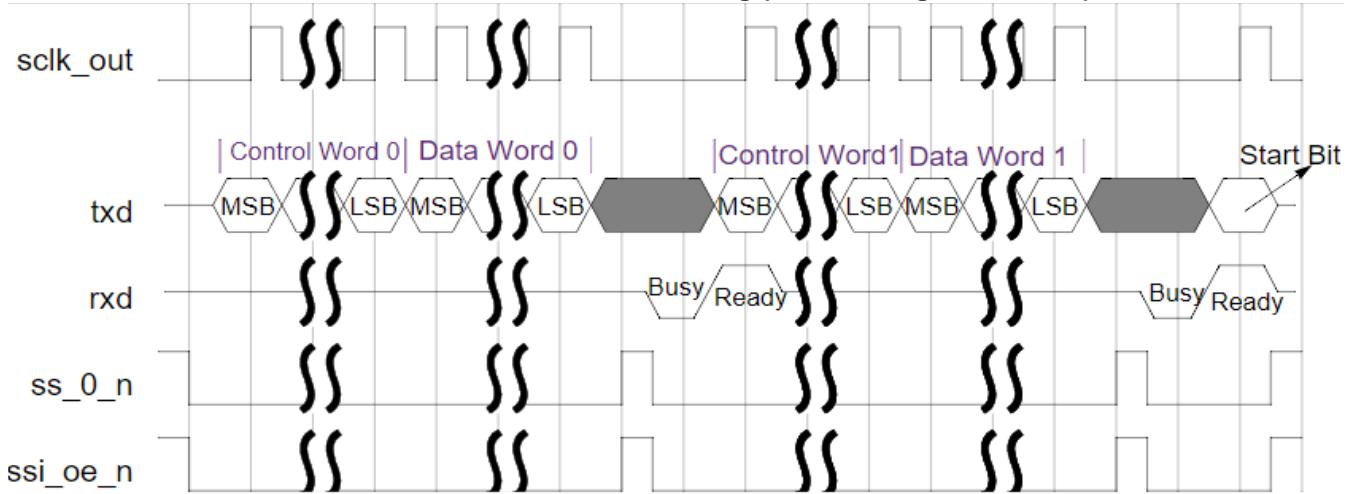
**Figure 2-24 Continuous Microwire Transfer (Transmitting Data Frame)**

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 2-25](#). This example shows two data words are written to the external serial target device.

**Figure 2-25 FIFO Status for Continuous Microwire Transfer (Transmitting Data Frame)**

The Microwire handshaking interface can also be enabled for DWC\_ssi serial controller write operations to external serial-target 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 DWC\_ssi serial controller checks for a ready status from the target device before completing the transfer, or transmitting the next control word for continuous transfers.

[Figure 2-26](#) shows an example of a continuous Microwire transfer with the handshaking interface enabled.

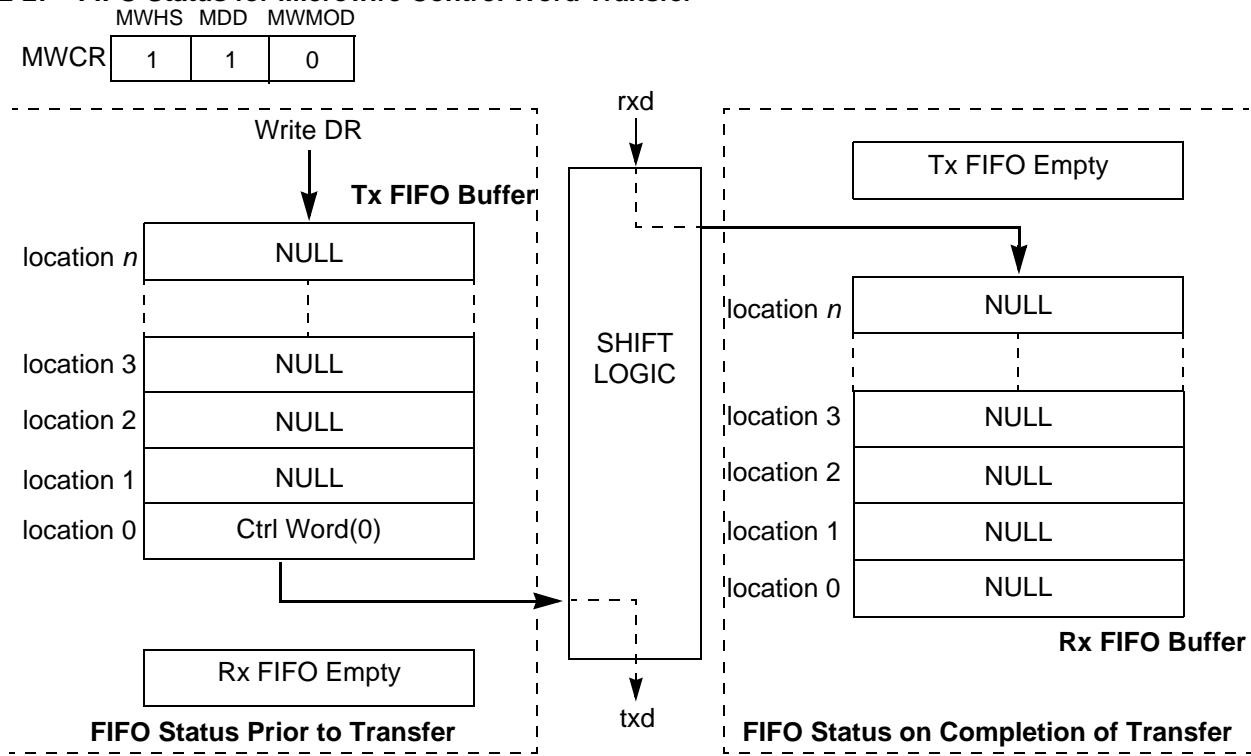
**Figure 2-26 Continuous Microwire Transfer with Handshaking (Transmitting Data Frame)**

After the first data word has been transmitted to the serial-target device, the DWC\_ssi serial controller polls the rxd input waiting for a ready status from the serial target device. Upon reception of the ready status, the DWC\_ssi serial controller begins transmission of the next control word. After transmission of the last data frame has completed, the DWC\_ssi serial controller transmits a start bit to clear the ready status of the serial target device before completing the transfer. The FIFO status for this transfer is the same as in [Figure 2-25](#), except that the MWHS bit field is set (1).

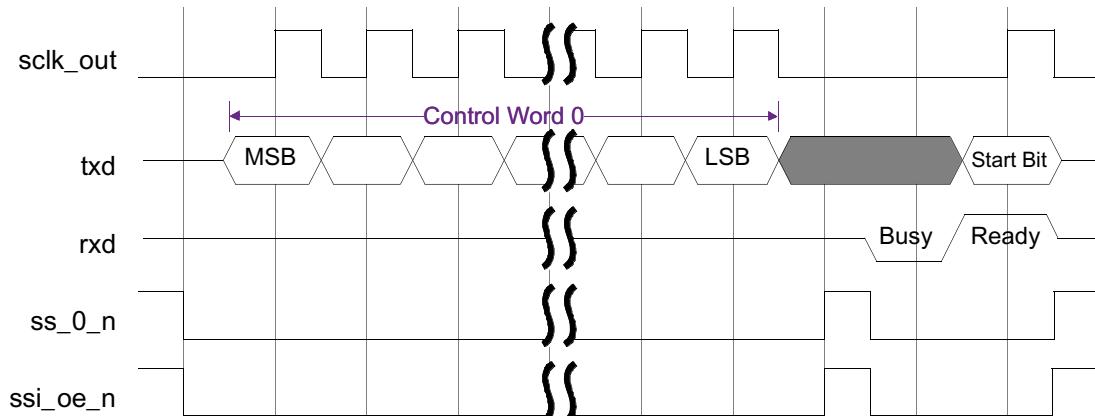
To transmit a control word (not followed by data) to a serial-target device from the DWC\_ssi serial controller, there must be only one entry in the transmit FIFO buffer. It is impossible to transmit two control words in a continuous transfer, as the shift logic in the DWC\_ssi treats the second control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

In the example shown in [Figure 2-27](#) and in the timing diagram in [Figure 2-28](#), the handshaking interface is enabled. If the handshaking interface is disabled (MHS = 0), the transfer is terminated by the DWC\_ssi serial controller one sclk\_out cycle after the LSB of the control word is captured by the serial target device.

**Figure 2-27 FIFO Status for Microwire Control Word Transfer**



**Figure 2-28 Microwire Control Word**



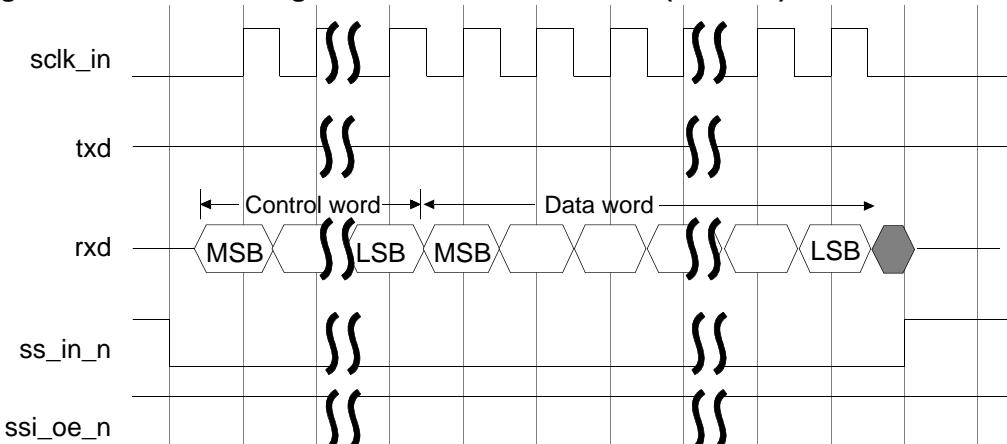
When the DWC\_ssi is configured as a serial target, data transmission begins with the falling edge of the chip select signal (`ss_in_n`). One-half serial clock (`sclk_in`) period later, the first bit of the control is present on the `rxn` line. The length of the control word can be in the range of 1 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 controller. The remainder of the control word is received (captured on the rising edge of `sclk_in`) by the DWC\_ssi serial target. During this reception, no data are driven (high impedance) on the serial target's `txn` 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 DWC\_ssi serial target is to receive data from the external serial controller. Immediately after the control word is transmitted, the serial controller begins to drives the data frame onto the DWC\_ssi serial target rxd line. Data are propagated on the falling edge of the serial clock and captured on the rising

edge. The chip-select signal is held active-low during the transfer and is de-asserted one-half clock cycle later after the data are transferred. The DWC\_ssi serial target output enable signal (`ssi_oe_n`) is held inactive for the duration of the transfer.

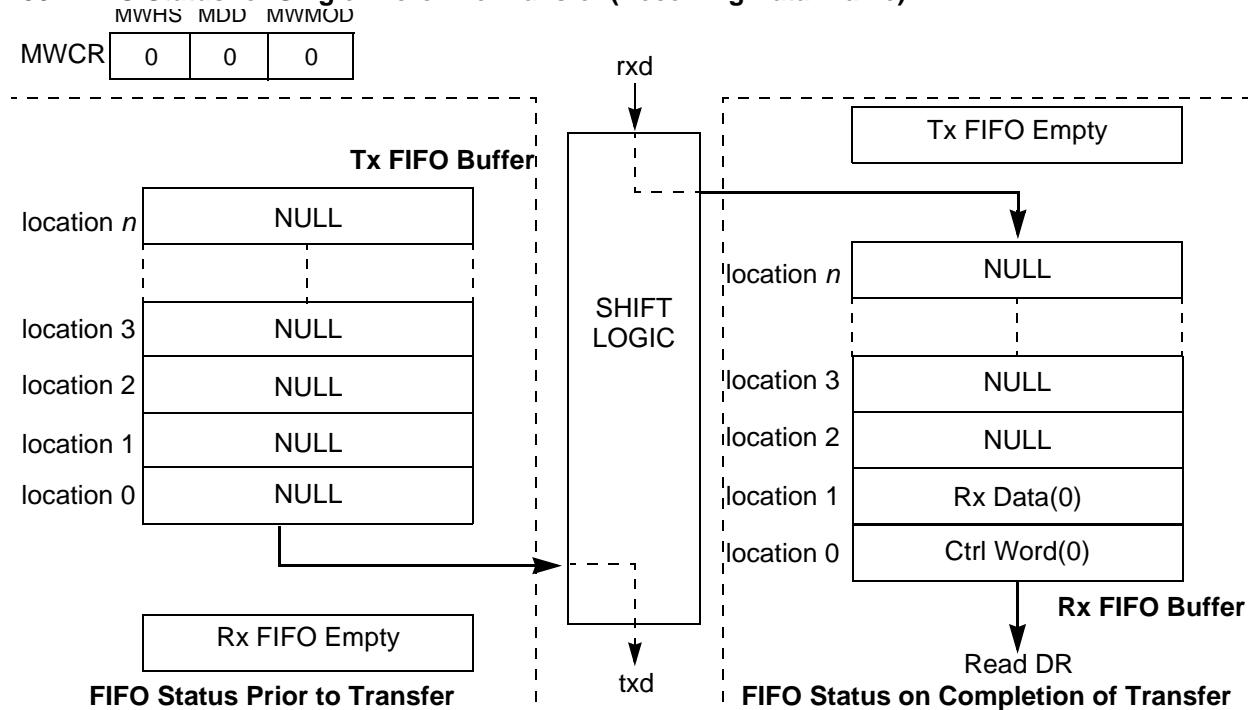
[Figure 2-29](#) shows the timing diagram for single DWC\_ssi serial target read from an external serial controller.

**Figure 2-29 Single DWC\_ssi Serial Target Microwire Serial Transfer (MDD = 0)**



[Figure 2-30](#) 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.

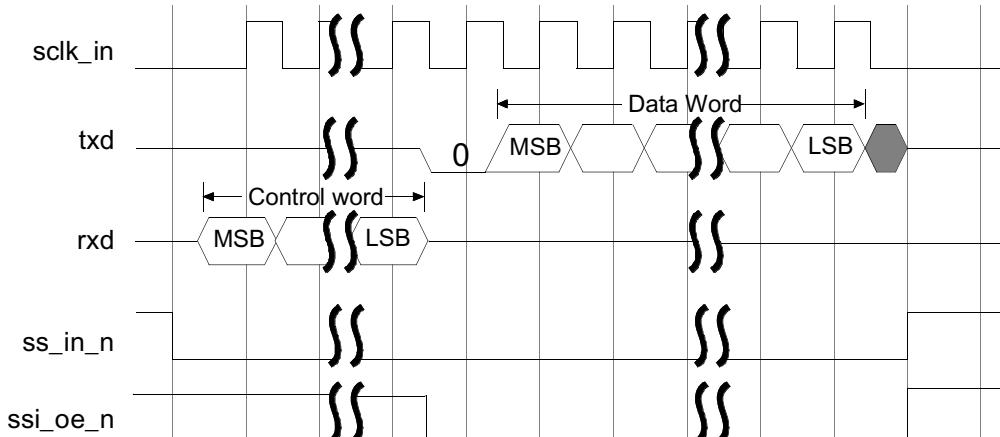
**Figure 2-30 FIFO Status for Single Microwire Transfer (Receiving Data Frame)**



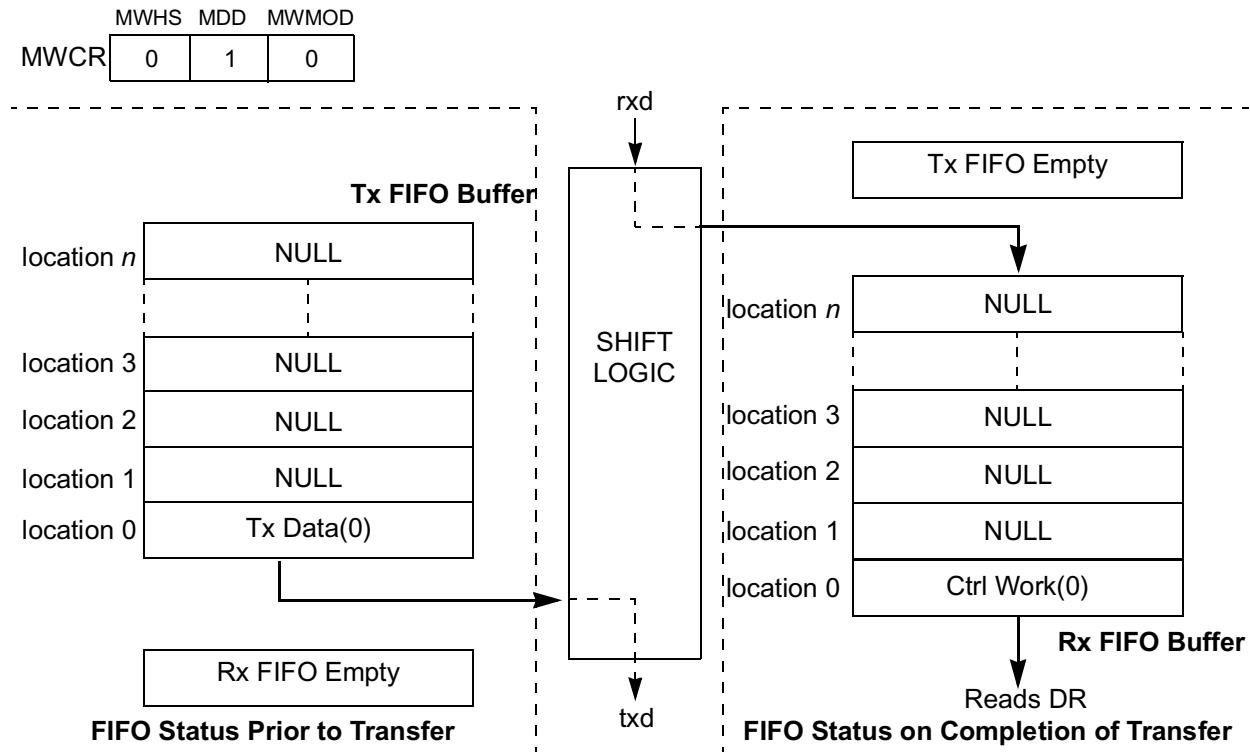
When `MDD = 1`, this indicates that the DWC\_ssi serial target transmits data to the external serial controller. Immediately after the LSB of the control word is transmitted, the DWC\_ssi serial target transmits a temporary 0-bit, followed by the 4-bits to 32-bit data frame on the `txd` line.

Figure 2-31 shows the timing diagram for a single DWC\_ssi serial target write to an external serial controller.

**Figure 2-31 Single DWC\_ssi Serial Target Microwire Serial Transfer (MDD = 1)**



**Figure 2-32 FIFO Status for Single Microwire Transfer (Transmitting Data Frame)**



Continuous transfers for a DWC\_ssi serial target occur in the same way as those specified for the DWC\_ssi serial controller configuration. The DWC\_ssi serial target configuration does not support the handshaking interface, as there is never a busy period.

## 2.5 Data Transfers

Data transfers are started by the serial-controller device when:

- DWC\_ssi is enabled (`SSI_EN = 1`) and
- Serial-target device is enabled (`SER` field)
- Number of data entries in transmit FIFO is greater than `TXFTHR` field of the `TXFTLR` register

While actively transferring the 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.



The `BUSY` status is not set when the data are written into the transmit FIFO. This bit gets set only when the target device 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.

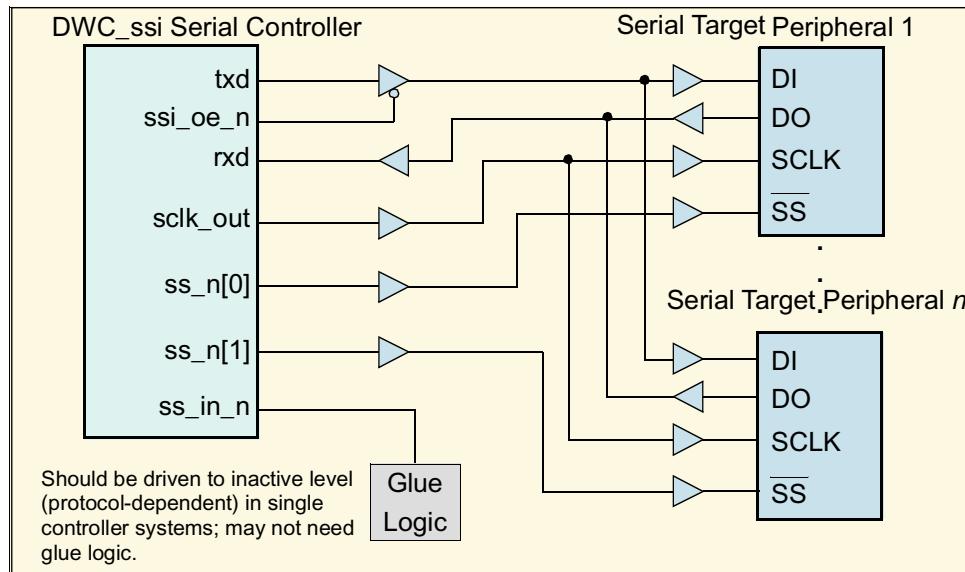
## 2.6 Serial Controller Mode

DWC\_ssi can be configured to function in the serial controller mode that enables serial communication with serial target peripheral devices. This section describes the Serial Controller mode of configuration for DWC\_ssi.

### 2.6.1 Description of Serial Controller Mode

This mode enables serial communication with serial-target peripheral devices. When configured as a serial-controller device (`SSIC_IS_MASTER=1`), the DWC\_ssi initiates and controls all serial transfers. Figure 2-33 shows an example of the DWC\_ssi configured as a serial controller with all other devices on the serial bus configured as serial target devices.

**Figure 2-33** DWC\_ssi Configured as Controller Device

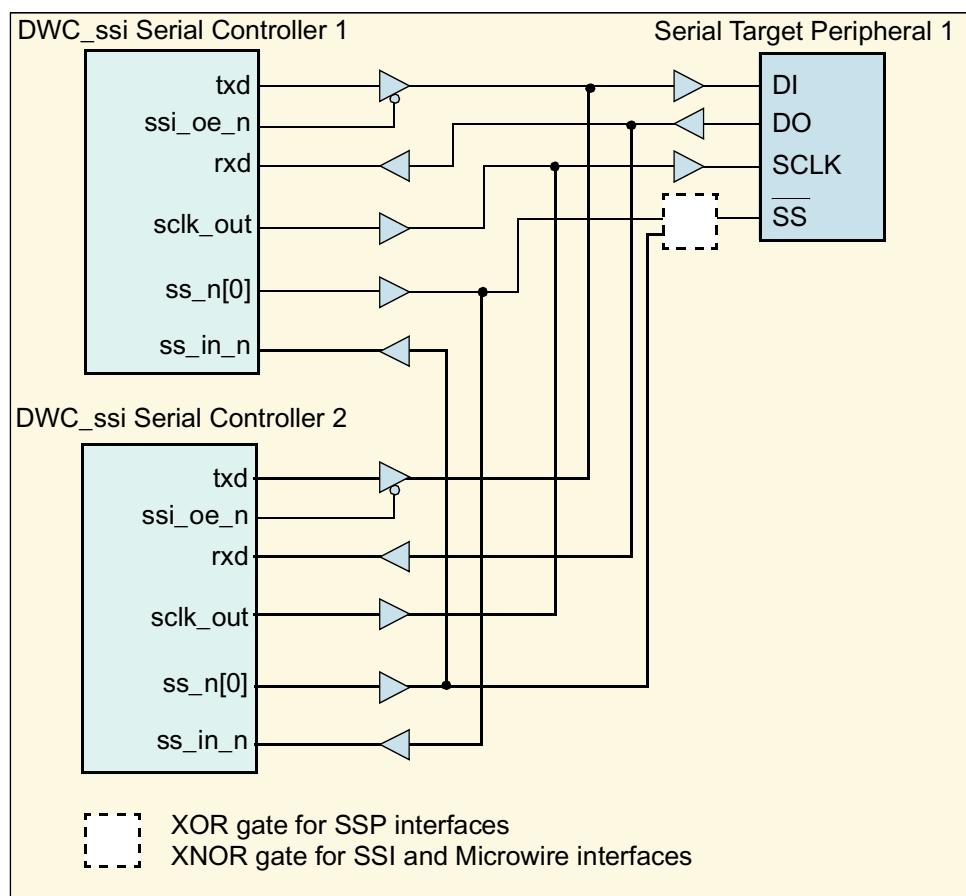


The serial bit-rate clock, generated and controlled by the DWC\_ssi, is driven out on the `sclk_out` line. When the DWC\_ssi is disabled (`SSIC_EN = 0`), no serial transfers can occur and `sclk_out` is held in Inactive state, as defined by the serial protocol under which it operates.

#### 2.6.1.1 Serial Controller Contention Input

The DWC\_ssi serial controller configuration has a serial chip select input, `ss_in_n`, that can be used to inform the DWC\_ssi serial controller that another serial controller is active on the bus. When this input is active—the active level depends on the serial protocol—the DWC\_ssi serial controller 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 controllers. A simple usage example is shown in Figure 2-34.

**Figure 2-34 Arbitration Between Multiple Serial Controllers**

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 serial controllers assert their chip 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 controller chip select outputs, chip select inputs and controller chip select inputs, to prevent any potential bus locking occurrences.

If the **DWC\_ssi** serial controller is the only controller device on the serial bus, you might need to insert some glue logic to control the logic level on the controller **ss\_in\_n** input.

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

- If you never intend to dynamically change the serial protocol that the **DWC\_ssi** serial controller 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.

## 2.6.2 Enabling DWC\_ssi as a Serial Controller

To configure DWC\_ssi in serial controller mode, select the **Serial Controller** option in the **Serial controller or serial target configuration** field using the **Top Level Parameters** tab during the Specify Configuration Activity in coreConsultant.

## 2.7 Serial Target Mode

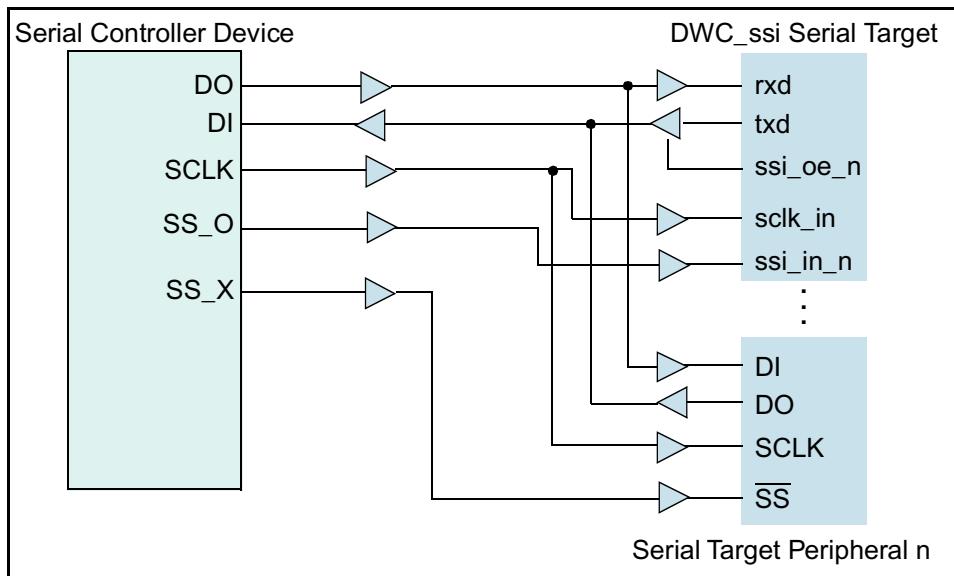
DWC\_ssi can be configured to function in the serial target mode that enables serial communication with serial controller peripheral devices. This section describes the Serial Target mode of configuration for DWC\_ssi.

### 2.7.1 Description of Serial Target Mode

This mode enables serial communication with serial controller peripheral devices. When the DWC\_ssi is configured as a serial target device (`SSIC_IS_MASTER=0`), all serial transfers are initiated and controlled by the serial bus controller. [Figure 2-35](#) shows an example of the DWC\_ssi configured as a serial target in a single-serial controller bus system.

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

**Figure 2-35** DWC\_ssi Configured as Serial Target Device



When the DWC\_ssi serial target is not selected, it must not interfere with data transfers between the serial-controller and other serial-target devices. When the serial target is not selected, its txd output is buffered, resulting in a high impedance drive onto the serial controller rxd line. The buffers shown in [Figure 2-35](#) are external to DWC\_ssi.

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

## 2.7.2 Enabling DWC\_ssi as a Serial Target

To configure DWC\_ssi in serial target mode, select the required **Serial Target** option in the **Serial controller or serial target configuration** field using the **Top Level Parameters** tab during the Specify Configuration Activity in coreConsultant.

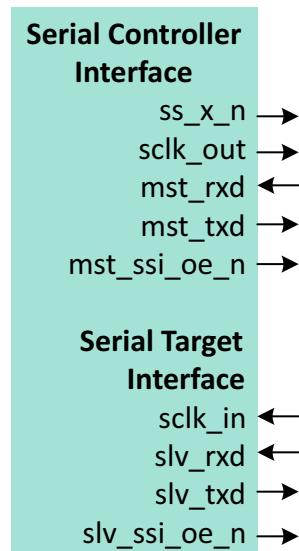
## 2.8 Serial Controller and Serial Target Operation (Programmable Mode)

DWC\_ssi can be configured to function in the serial controller or serial target that enables serial communication with serial target or serial controller peripheral devices respectively. This section describes the Serial Target mode of configuration for DWC\_ssi.

### 2.8.1 Description of Programmable Mode

DWC\_ssi supports serial controller and target operations through programming. The mode of operation can be changed by programming the SSIC\_IS\_MASTER field of the CTRLR0 register. When SSIC\_IS\_MASTER=2, two separate serial controller and serial target interfaces are instantiated within DWC\_ssi. All serial controller and serial target specific registers are available in the register map, and appropriate register is active depending on the mode of operation.

**Figure 2-36 Serial Interface in Serial Controller/Target Configuration**



Serial target interface signals are not used in serial controller mode of operation; slv\_rxd signal and sclk\_in signal are not sampled by DWC\_ssi, and txd\_slv signal is driven to high-impedance level. The same is applicable for controller Interface signals in serial target mode. Having different signals for serial controller and serial target modes makes it easier to connect the IP with I/O pads and helps separate the paths internal to the IP.

ss\_in\_n pin is used in both serial controller and target modes. In serial controller mode, ss\_in\_n pin is used to detect serial controller contention; and in serial target mode of operation, ss\_in\_n signal is used as chip select signal. The same TX and RX FIFO are used by DWC\_ssi serial controller and target.

### 2.8.2 Enabling DWC\_ssi as a Serial Controller and Serial Target

To configure DWC\_ssi for serial controller and serial target operations, select the **Programmable** option in the **Serial controller or serial target configuration** field using the **Top Level Parameters** tab during the Specify Configuration Activity in coreConsultant.

## 2.9 Clock Ratios

DWC\_ssi works on an oversampling architecture. For the serial controller mode of operation, the peripheral clock (`sclk_out`) period is a multiple of the internal core clock (`ssi_clk`). In the serial target mode of operation, a synchronized peripheral clock is used by DWC\_ssi to receive or transmit the data. This section describes the details of clock ratio between the peripheral clock and internal clock for both Serial Controller and target modes.

### 2.9.1 Description of Clock Ratios

When the DWC\_ssi macrocell is configured as a serial controller device, the maximum frequency of the bit-rate clock (`sclk_out`) is one-half the frequency of `ssi_clk` signal. This is to allow the shift control logic to capture data on one clock edge of `sclk_out` signal and propagate data on the opposite edge; this is illustrated in [Figure 2-37](#).

The frequency of `sclk_out` signal can be derived from the following equation.

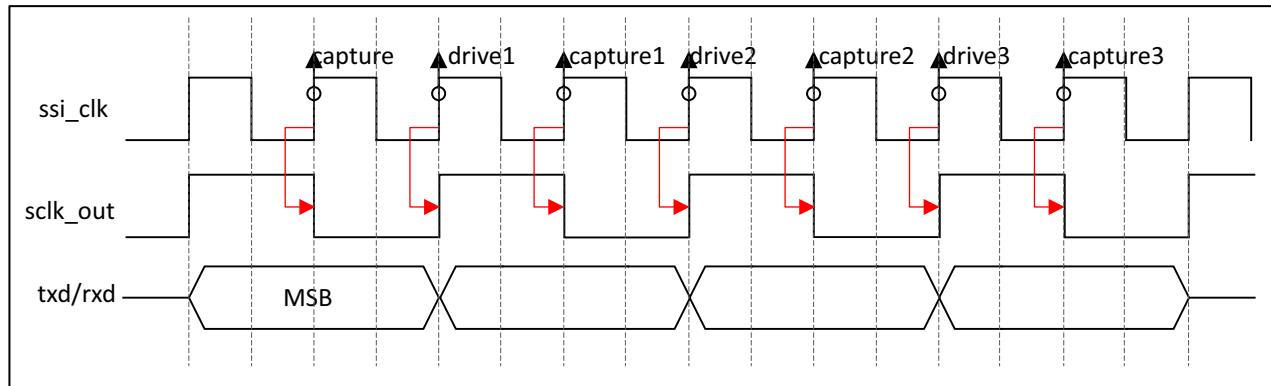
$$F_{sclk\_out} = F_{ssi\_clk}/SCKDV$$

Where, `SCKDV` is a programmable register holding any even value in the range 0 – 65,534. If `SCKDV` = 0, `sclk_out` is disabled.

The `sclk_out` line only toggles when an active transfer is in progress. At all other time, it is held in inactive state, as define by the serial protocol under which it operates.

[Figure 2-37](#) illustrates the maximum ratio between `sclk_out` signal and `ssi_clk` signal.

**Figure 2-37 Maximum `sclk_out/ssi_clk` Ratio**



When the DWC\_ssi macrocell is configured as a serial target device the minimum frequency of the `ssi_clk` signal depends on the `SSIC_ENH_CLK_RATIO` configuration parameter and the operation of the serial target peripheral.

This section describes the following sections:

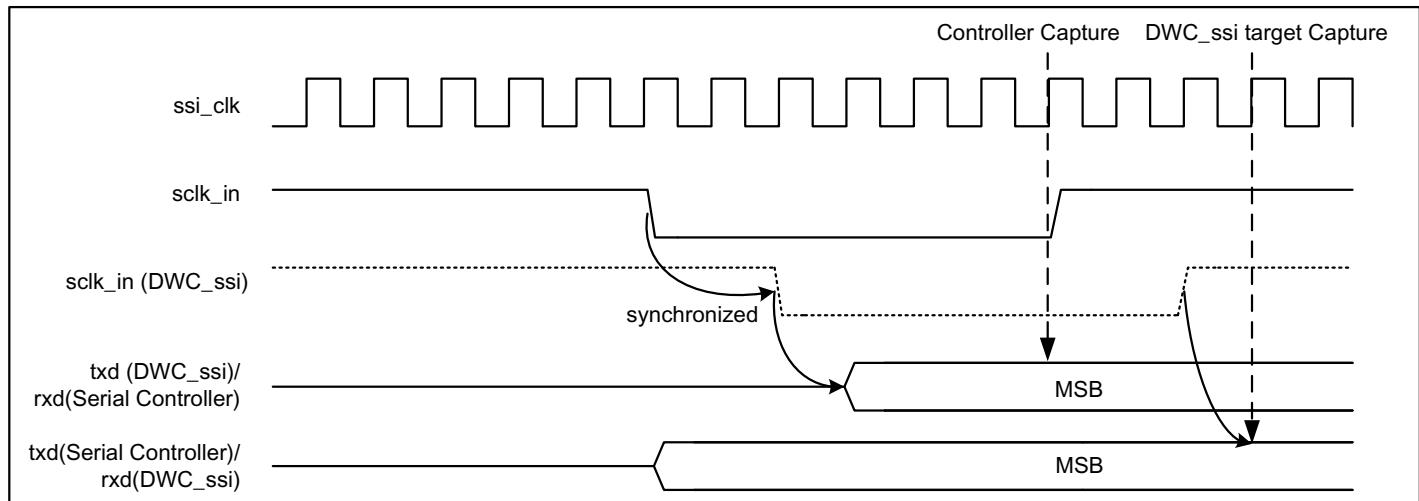
- “[SSIC\\_ENH\\_CLK\\_RATIO = 0](#)” on page [65](#)
- “[SSIC\\_ENH\\_CLK\\_RATIO = 1](#)” on page [65](#)

### 2.9.1.1 SSIC\_ENH\_CLK\_RATIO = 0

If the serial target device is *receive only*, the minimum frequency of `ssi_clk` signal is six times the maximum expected frequency of the bit-rate clock from the serial controller device (`sclk_in`). The `sclk_in` signal is double synchronized to bring it across to the `ssi_clk` domain and then the edge is detected.

If the serial target device is transmit and receive, the minimum frequency of `ssi_clk` signal is eight times the maximum expected frequency of the bit-rate clock from the serial controller device (`sclk_in`). This minimum frequency is to ensure that data on the serial controller's `rx` line is stable before the serial controller's shift control logic captures the data. The 8:1 ratio ensures that the serial target has driven data onto the serial controller's `rx` line two `ssi_clk` cycles before the data is captured, which is indicated by `tc` (time before capture) in [Figure 2-38](#).

**Figure 2-38 Serial Target `ssi_clk/sclk_in` Ratio**



For high clock ratio of `ssi_clk` and `sclk_in` (more than 1:8), then we would recommend setting `SSIC_ENH_CLK_RATIO` parameter to 0.

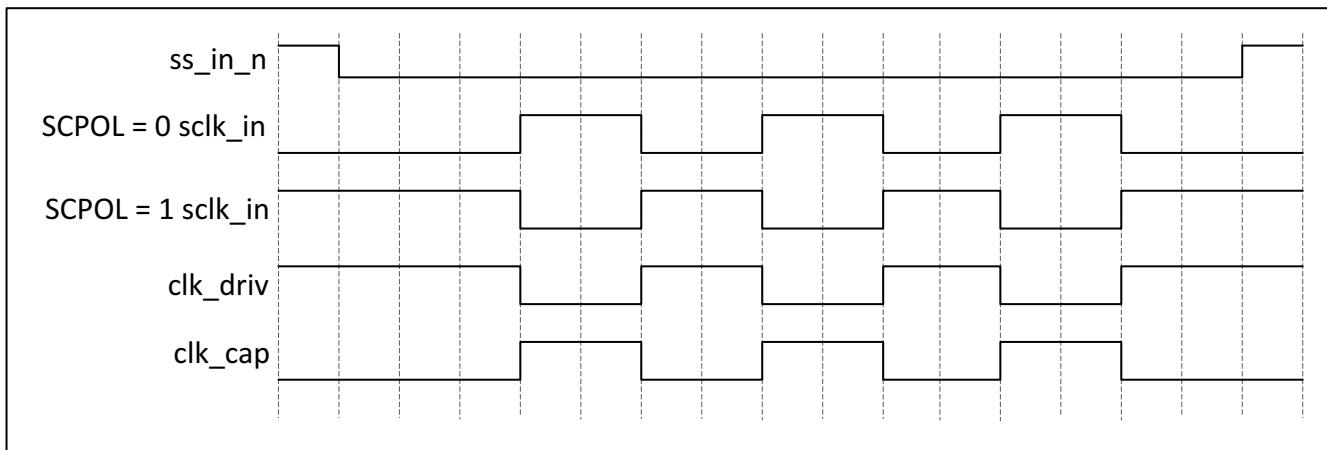
### 2.9.1.2 SSIC\_ENH\_CLK\_RATIO = 1

This mode includes the advanced architecture for Transmit and Receive shift registers. These registers work directly with bit-rate clock from the serial controller device (`sclk_in`), thus, eliminating the need for synchronization.

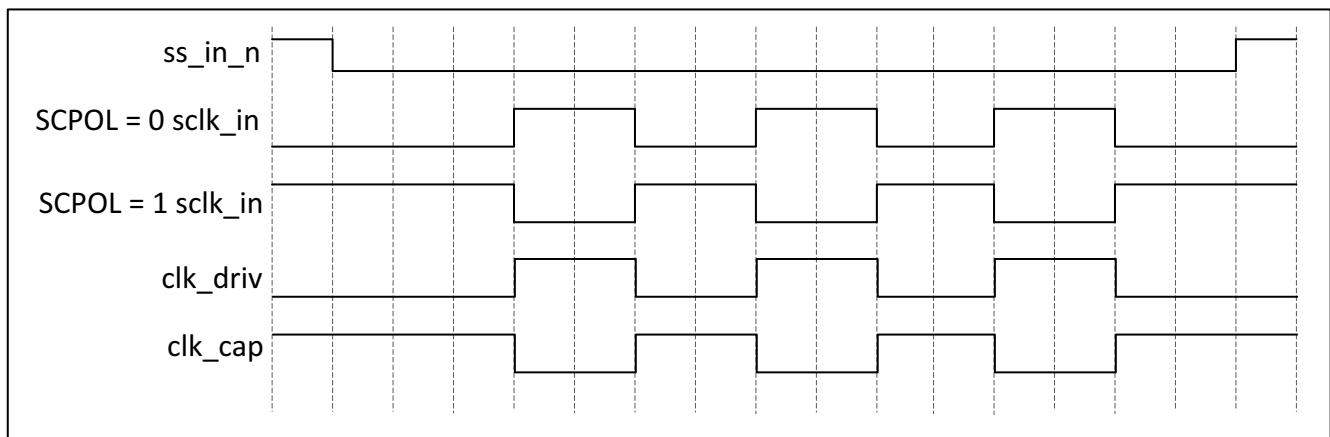
To reduce the synchronization delay, the synchronization scheme uses two flip-flops: one works on the positive edge of `ssi_clk` signal; and other works on the negative edge of `ssi_clk` signal. These flip-flops reduce the synchronization delay to one `ssi_clk` cycle and enable `DWC_ssi` to work on lower clock ratios.

As different frame formats use different capture and driving edge, the bit-rate clock is gated accordingly to align with the protocol specification. The capture clock is used in receive shifter to capture the data on the `rx` line, and driving clock is used in transmit shifter to drive the data on the `tx` line. Waveforms in [Figure 2-39 – Figure 2-42](#) illustrate how the capture and driving clocks are derived from bit-rate clock for different frame formats.

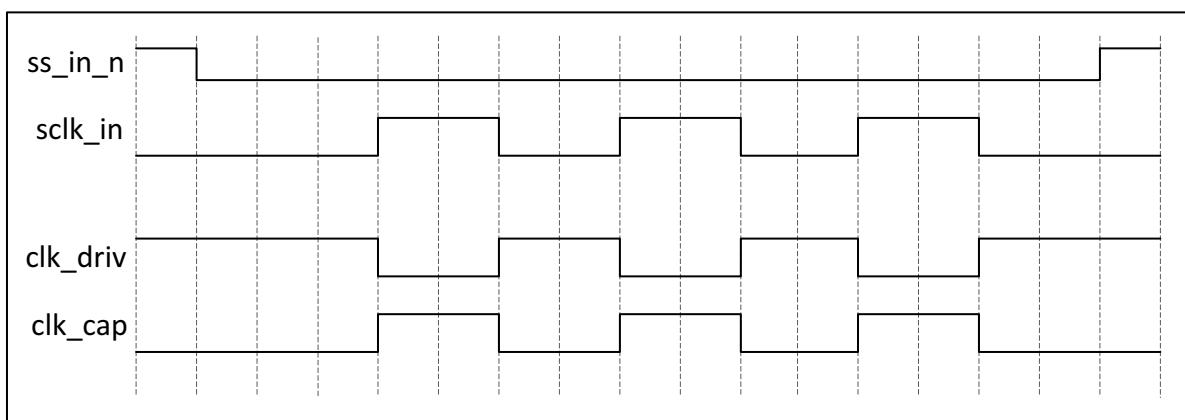
**Figure 2-39 Frame Format for Motorola Serial Peripheral Interface (SPI) with SCPH = 0**

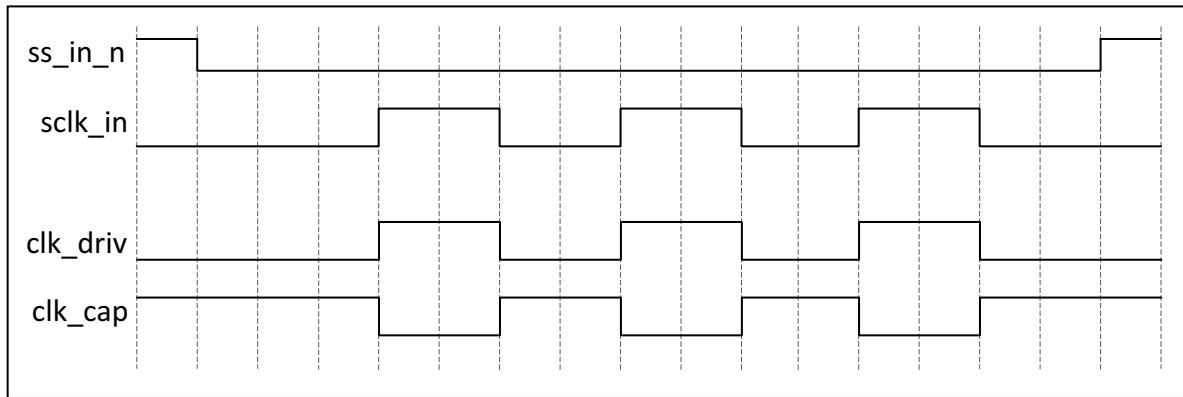


**Figure 2-40 Frame Format for Motorola Serial Peripheral Interface (SPI) with SCPH = 1**



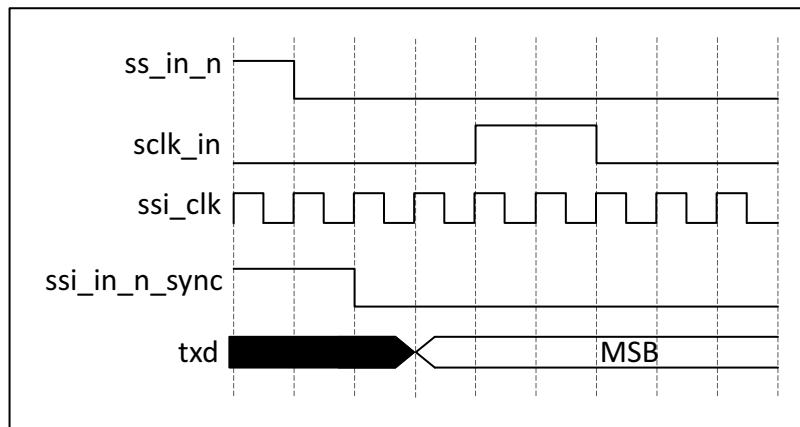
**Figure 2-41 National Semiconductors Microwire Frame Format**



**Figure 2-42 Texas Instruments Synchronous Serial Protocol (SSP)**

When **SSIC\_ENH\_CLK\_RATIO = 1**, **DWC\_ssi** serial target device minimum frequency of **ssi\_clk** signal is four times the maximum expected frequency of bit-rate clock (**sclk\_in**).

If the frame format is programmed for SPI with **SCPH = 0**, then before the first edge arrives, the data must be present on the **txd** line. Internally to drive data on **txd** line, the synchronized version of the chip 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** signal. 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 serial target device is selected (**ss\_in\_n**).

**Figure 2-43 Driving TxD When SCPH = 0**

### 2.9.1.3 Frequency Ratio Summary

A summary of the frequency ratio restrictions between the bit-rate clock (**sclk\_out/sclk\_in**) and the **DWC\_ssi** peripheral clock (**ssi\_clk**) are as follows:

- Serial Target **SSIC\_ENH\_CLK\_RATIO = 0**
  - Receive only:  $F_{ssi\_clk} \geq 6 \times (\text{maximum } F_{sclk\_in})$
  - Transmit and Receive:  $F_{ssi\_clk} \geq 8 \times (\text{maximum } F_{sclk\_in})$
- Serial Target **SSIC\_ENH\_CLK\_RATIO = 1**

- $F_{ssi\_clk} \geq 4 \times (\text{maximum } F_{sclk\_in})$

#### 2.9.1.4 Design For Test

As explained in “[SSIC\\_ENH\\_CLK\\_RATIO = 1](#)” on page 65, when SSIC\_ENH\_CLK\_RATIO is set to 1, then `sclk_in` signal is used as capture and drive clock for `rx` and `tx` lines, respectively. The polarity of clock is changed based on frame format being used. During the scan testing, these flip-flops may remain uncovered. Therefore, you must connect `scan_mode` signal to chip-level scan mode. During scan mode (`scan_mode = 1`), the clock input for these flip-flops are connected to `ssi_clk` signal, rather than the internally derived `clk_driv` and `clk_cap`. This makes register testable and subsequent down stream points controllable.

#### 2.9.2 Including Enhanced Clock Ratio Architecture

To configure DWC\_ssi to include the enhanced clock ratio architecture, choose the **Include Enhanced Clock Ratio Architecture?** field using the **Top Level Parameters** tab during the Specify Configuration Activity in coreConsultant.

This field provides the following options:

- Yes – Configures the device to include new architecture for Transmit and Receive FIFO. This enables the device to work on clock ratios of 4 and 6 between the `ssi_clk` and `sclk_in` signals
- No – Does not include the enhanced clock ratio architecture

## 2.10 Receive and Transmit FIFO Buffers

The FIFO buffers used by the DWC\_ssi are internal D-type flip-flops that can be configured in depth between 8 to 256. The widths of both transmit and receive FIFO buffers are fixed at 32 bits due to the serial specifications, which state that a serial transfer (data frame) can be 4 to 32 bits in length. Data frames that are less than 32 bits 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.

### 2.10.1 Description of Receive and Transmit FIFO Buffers

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 24 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 DWC\_ssi is disabled (`SSIC_EN = 0`) or when it is reset (`hresetn`).

The transmit FIFO is loaded by AHB write commands to the DWC\_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 (through `ssi_txe_intrsignal`) 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 (through `ssi_txo_intr`) is generated if you attempt to write data into an already full transmit FIFO.

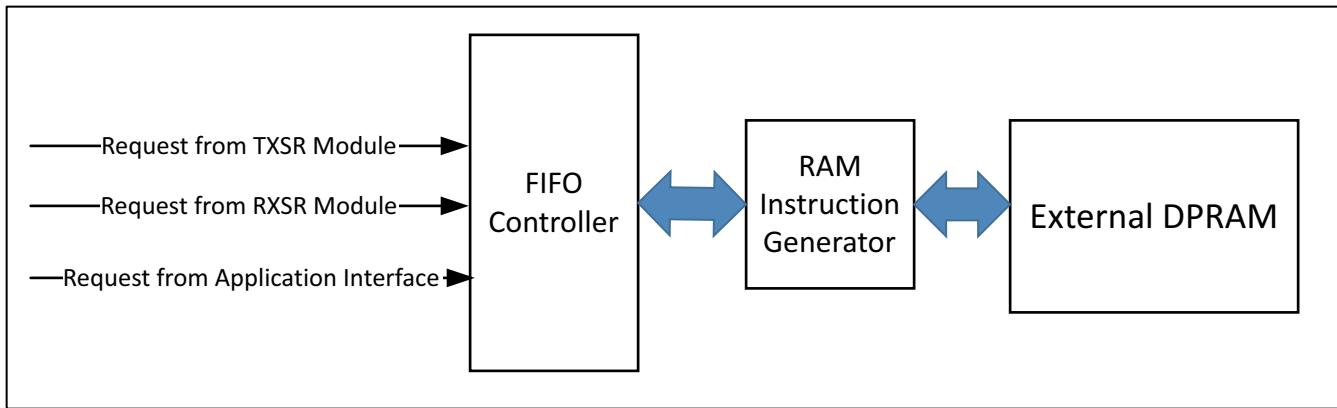
Data are popped from the receive FIFO by AHB read commands to the DWC\_ssi data register (DR). The receive FIFO is loaded from the receive shift register by the shift control logic. The receive FIFO generates a FIFO-full interrupt request (`ssi_rxf_intr`) when the number of entries in the FIFO is greater 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.

### 2.10.2 External Dual Port SRAM

DWC\_ssi can be configured to support external Dual Port SRAM. This can be configured using coreConsultant parameter `SSIC_HAS_EXT_RAM`. After `SSIC_HAS_EXT_RAM` parameter is enabled, ports are added on DWC\_ssi to interface with external SRAM. Data Register (DR) is used to interface with SRAM as usual.

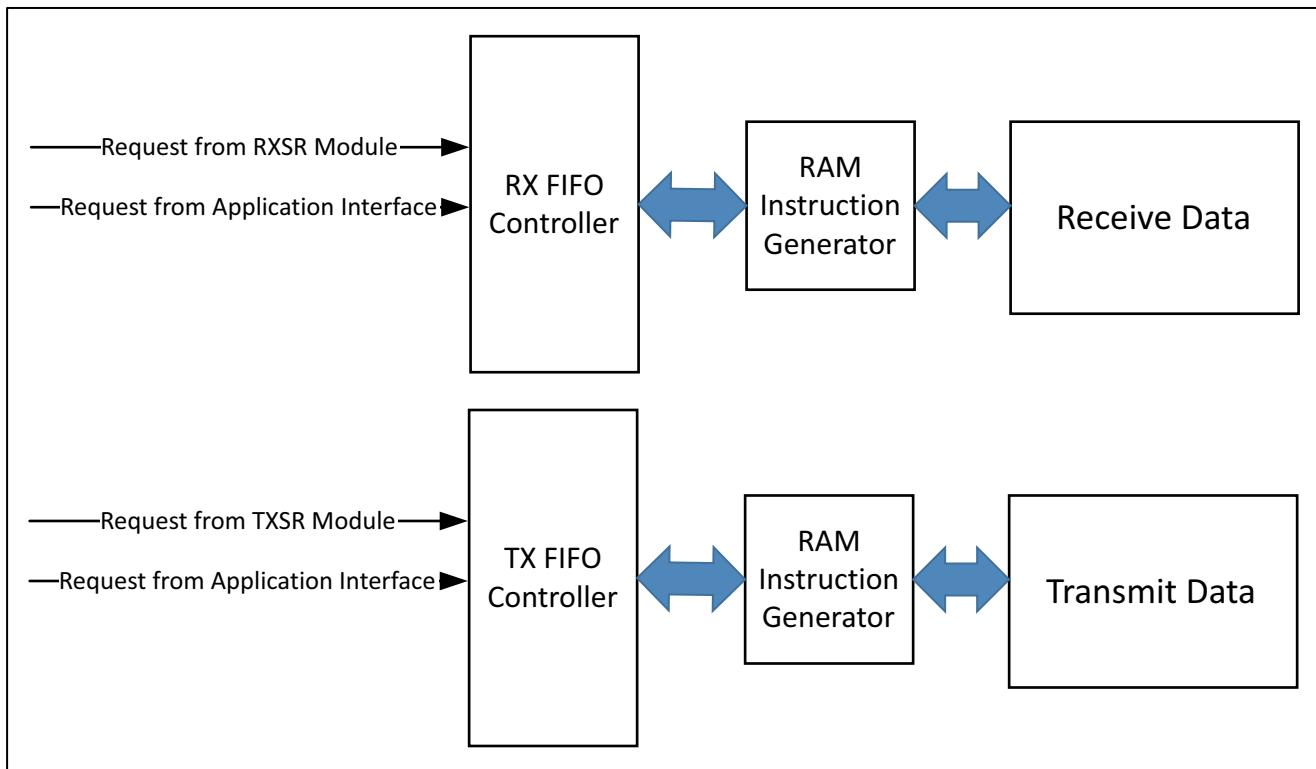
Single SRAM interface is available to store both Transmit and Receive data. DWC\_ssi takes care of arbitration internally to ensure that only one transaction is issued at a particular point of time. [Figure 2-44](#) shows the control path structure of SRAM control block.

**Figure 2-44 Block Diagram for External Dual Port SRAM Module Control Path**

To perform transmit and receive operations, DWC\_ssi has two separate SRAM interfaces to store write and read data separately. You need to enable this feature using the `SSIC_HAS_RX_RX_EN` parameter. The depth of these external RAMs are configured through the `SSIC_RX_FIFO_DEPTH` and `SSIC_TX_FIFO_DEPTH` parameters.

This increases the amount logic inside the IP because two FIFO controllers are required to keep track of both receive and transmit data. [Figure 2-45](#) shows the control path block diagram where RX and TX data is simultaneously stored inside the external RAM. Choose this type of configuration if you plan to use the following transfer types:

- For SPI/SSP transfers where TMOD field of the CTRLR0 register is set to 0 (Receive and transmit)
- When Microwire mode is selected, the following transfers are not applicable:
  - Non-Sequential Read in Serial Controller mode; `MWCR.MWMOD = 0` and `MWCR.MDD = 0`.
  - Non-Sequential Write in Serial Target mode; `MWCR.MWMOD = 0` and `MWCR.MDD = 1`.

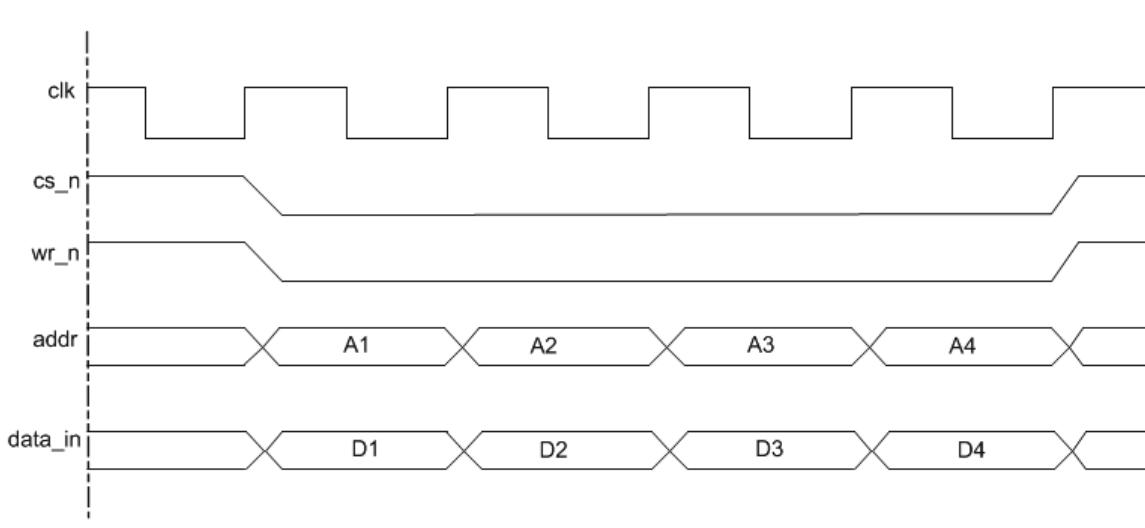
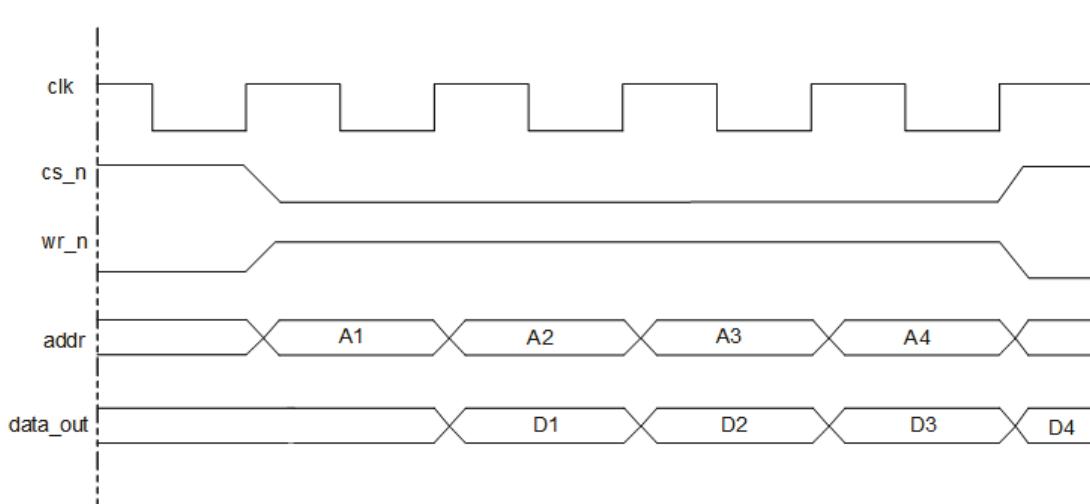
**Figure 2-45 Block Diagram for External SRAM Module Control Path for TX and RX Storage****Note**

If Concurrent SPI mode is enabled (`SSIC_CONCURRENT_XIP_EN = 1`), then an external SRAM is used to store the transfer information non-XIP transfer. A separate FIFO is instantiated inside `DWC_ssi` to store intermediate data for XIP transfers.

All output and input ports from the RAM are registered to meet the timing requirements. Therefore the data read and write takes an extra cycle when the `SSIC_HAS_EXT_RAM` parameter is set to 1.

RAM uses synchronous read and write ports. One of the ports is connected to the AHB clock, `hclk` and the second port is connected to the core clock (`ssi_clk`). `DWC_ssi` provides clock and reset output to connect with respective ports of the RAM. RAM width is always fixed to 32 bits, and depending on the depth the address pointer width is chosen. See the *Signals Description* chapter for more details.

The SRAM interface timings are shown in [Figure 2-46](#) and [Figure 2-47](#).

**Figure 2-46 SRAM Timing for WRITE****Figure 2-47 SRAM Timing for READ**

### 2.10.3 Configuring the Receive and Transmit FIFO Buffer Depth

To configure the receive and transmit FIFO buffer depth, choose a buffer depth from the **Receive FIFO buffer depth** and **Transmit FIFO buffer depth** fields using the Top Level Parameters tab during the Specify Configuration Activity in coreConsultant. The available buffer depth values for these fields are 8, 16, 32, 64, 128, 256.

### 2.10.4 Registers Related to Receive and Transmit FIFO Buffers

The following are the registers related to Receive and Transmit FIFO Buffers:

- RXFTLR
- TXFTLR

- TXFLR
- RXFLR

For more information about these registers, see the *Registers Descriptions* chapter.

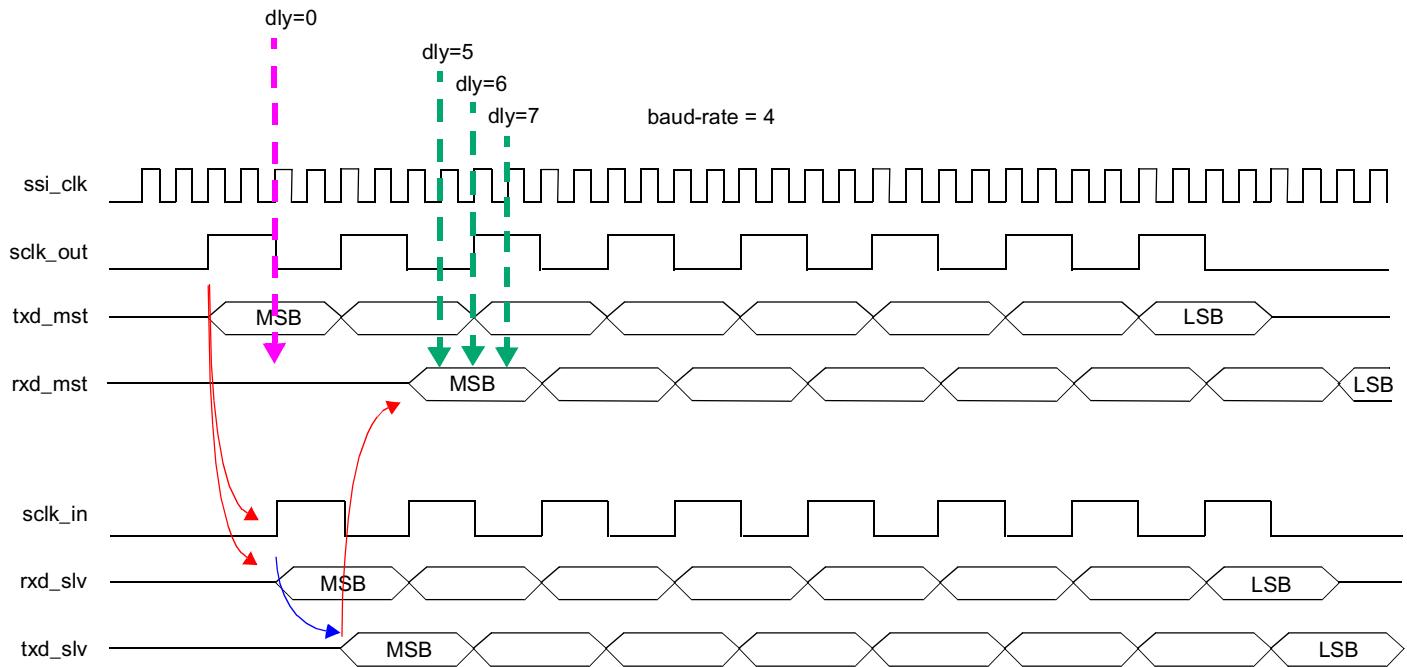
## 2.11 RXD Sample Delay

When DWC\_ssi is configured as a serial controller, 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.

### 2.11.1 Description of RXD Sample Delay

Round trip routing delays on the sclk\_out signal from the serial controller and the rxd signal from the target can mean that the timing of the rxd signal—as seen by the serial controller—has moved away from the normal sampling time. [Figure 2-48](#) illustrates this situation.

**Figure 2-48 Effects of Round-Trip Routing Delays on the sclk\_out Signal**



Red arrows indicate routing delay between serial controller and target devices

Blue arrow indicates sampling delay within target from receiving slk\_in to driving txd out

The serial target uses the sclk\_out signal from the serial controller 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 serial target device can mean that the rxd bit has not stabilized to the correct value before the serial controller samples the rxd signal. [Figure 2-48](#) 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 serial controller samples the port.

Without the RXD Sample Delay logic, you must 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 serial target and increments the serial controller’s RXD Sample Delay value until the correct data is received by the serial controller.

RXD sample delay logic can be configured to use both positive and negative edge of `ssi_clk` to sample `rxrd` signal. This could increase the number of sampling points within a `sclk_out` period and help in meeting timings in higher frequencies.

DWC\_ssi uses `RX_SAMPLE_DELAY` register to change the sampling point of `rxrd` signal.

- If `RX_SAMPLE_DELAY` field of the `SE` register is set to 0, then DWC\_ssi delays the sampling point by the programmed number of `ssi_clk` cycles.
- If `RX_SAMPLE_DELAY` field of the `SE` register is set to 1, then DWC\_ssi delays sampling point by programmed number of `ssi_clk` cycles +  $0.5 * (\text{ssi\_clk period})$ . Thus, the sampling is done on negative edge of `ssi_clk` signal as described in the figure [Figure 2-49](#). This gives user more sampling points within single `sclk_out` clock period.



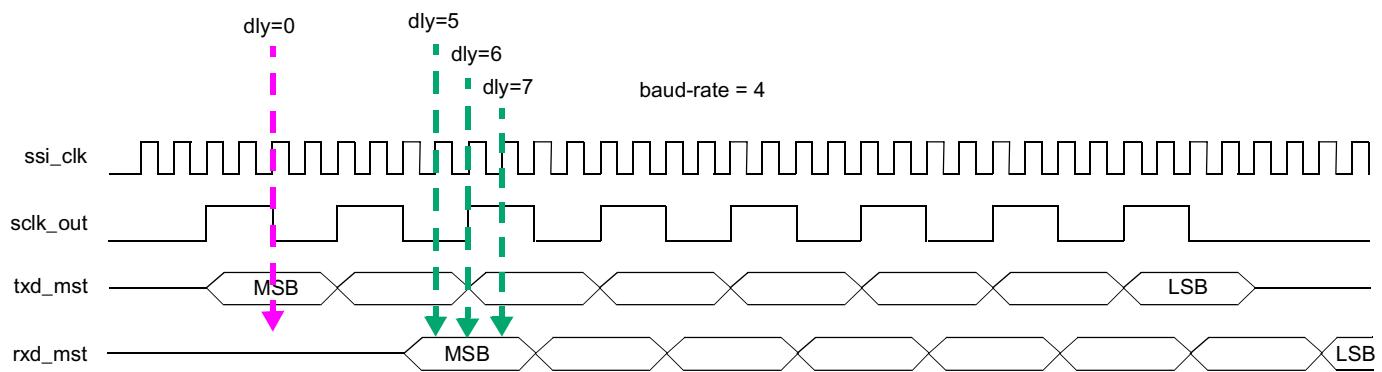
`RX_SAMPLE_DELAY` field of the `SE` register is available only if `SSIC_HAS_RX_SAMPLE_DELAY` configuration parameter is set to 2 (Value Use Both Positive and Negative Edges). Otherwise DWC\_ssi uses only positive edge of `ssi_clk` to sample `rxrd` signal.



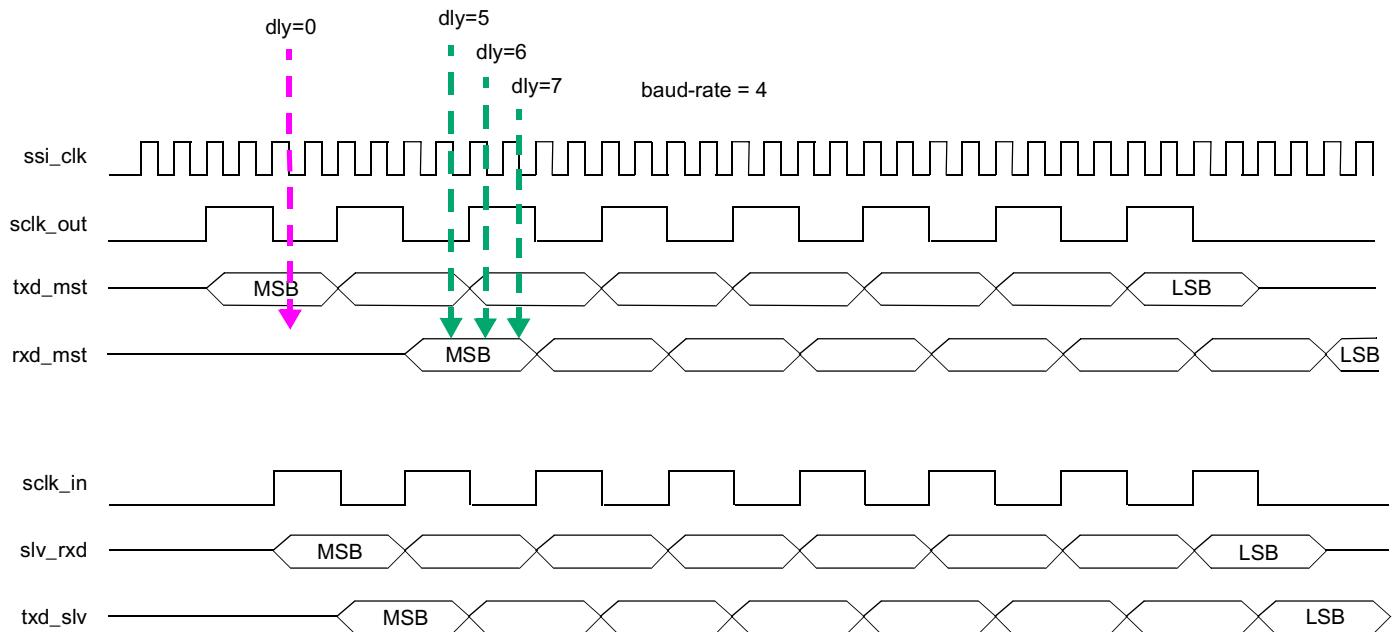
Only RXD sampling logic works on negative edge of `ssi_clk` signal depending on the programming, the rest of the logics work on the positive edge of `ssi_clk` signal. The `DWC_ssi_clk_mux_2x1.v` module implements a clock multiplexer and is used to multiplex the `ssi_clk` signal and `ssi_clk_n` signal. It is recommended to replace this module with a balanced multiplexer from the technology library.

**Figure 2-49 Sampling of rxd Signal by DWC\_ssi**

RX\_SAMPLE\_DLY.SE=0



RX\_SAMPLE\_DLY.SE=1



## 2.11.2 Setting the RXD Delay Value

To configure the maximum RXD sample delay in DWC\_ssi, specify the delay value in the **Maximum RXD Sample Delay** field using the **Top Level Parameters** tab during the Specify Configuration Activity in coreConsultant. The delay value can be a number from 4 to 255.

You can set this value only if the **Include Programmable RXD Sample Logic** field in the **Top Level Parameters** tab is set to **True**. The same parameter can be used to select if DWC\_ssi uses both positive and negative edge of `ssi_clk` signal to sample `rxd` signal.

### 2.11.3 Registers Related to RXD Sample Delay

The following are the registers related to RXD Sample Delay:

- `RX_SAMPLE_DELAY`

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.12 DMA Controller Interface

The DWC\_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 AHB bus is used to perform the data transfer to or from the DMA. While the DWC\_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.

### 2.12.1 Description of DMA Controller Interface

This section describes the settings of the DW\_ahb\_dmac that are relevant to the operation of the DWC\_ssi and the bit fields in the DW\_ahb\_dmac channel control register (CTL $x$ ) where  $x$  is the channel number.



**Note** When the DWC\_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 DWC\_ssi uses two DMA channels, one for the transmit data and one for the receive data. The DWC\_ssi has these DMA registers:

- DMACR – Control register to enable DMA operation.
- DMATDLR – Register to set the transmit 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 DWC\_ssi uses the following handshaking signals to interface with the DMA controller.

- dma\_tx\_req
- dma\_rx\_req
- dma\_tx\_ack
- dma\_rx\_ack
- dma\_tx\_single
- dma\_rx\_single

For more information about these signals, see the *Signals Description* chapter. They are described further in the “[Handshaking Interface Operation](#)” on page [83](#).

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 Description*” chapter in the *DesignWare DW\_ahb\_dmac Databook*. The DWC\_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 DWC\_ssi, you must write the DMA Control Register (DMACR). Writing 1 into the TDMAE bit field of DMACR register enables the DWC\_ssi transmit handshaking interface. Writing a 1 into the RDMAE bit field of the DMACR register enables the DWC\_ssi receive handshaking interface.

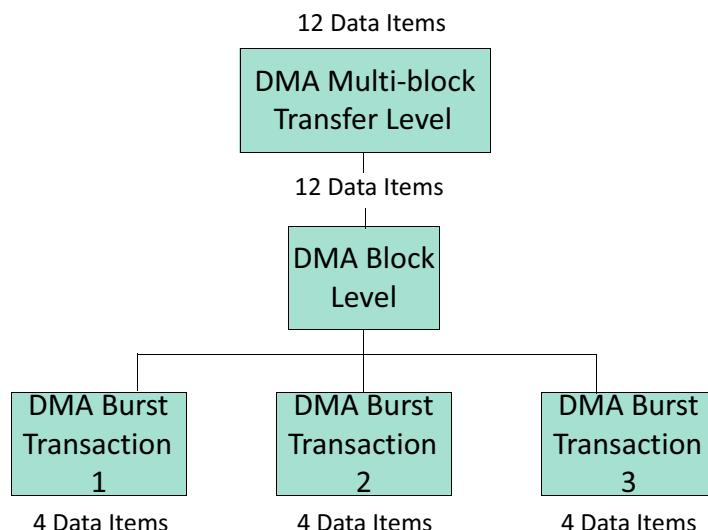
### 2.12.1.1 Overview of Operation

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 DWC\_ssi; this is programmed into the BLOCK\_TS field of the CTLx register. The block size includes the instructions, address, and data in case of the Enhanced SPI mode of operation.

The block is broken into a number of transactions, each initiated by a request from the DWC\_ssi. The DMA Controller must also be programmed with the number of data items (in this case, DWC\_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 CTLx register for source and destination, respectively.

[Figure 2-50](#) 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 2-50 Breakdown of DMA Transfer into Burst Transactions**



Block Size: DMA.CTLx.BLOCK\_TS=12  
 Number of data items per source burst transaction: DMA.CTLx.SRC\_MSIZE = 4  
 SSI receive FIFO watermark level: SSI.DMARDLR + 1 = DMA.CTLx.SRC\_MSIZE = 4  
 (for more information please see the discussion on page 63)

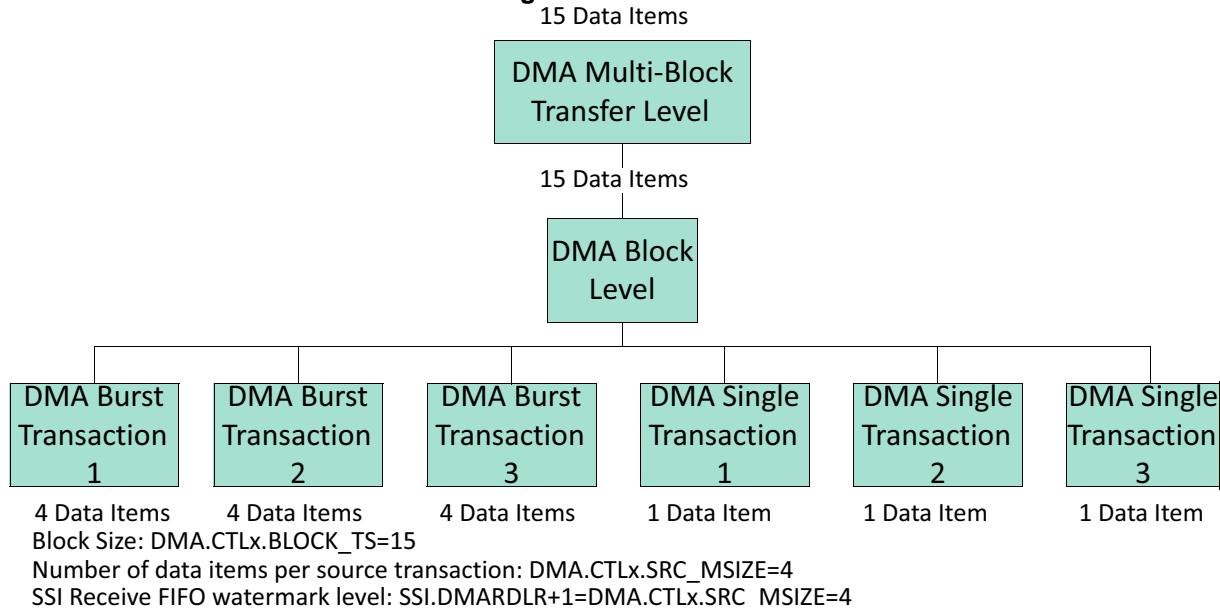
If the DWC\_ssi makes a transmit request to this channel, four data items are written to the DWC\_ssi transmit FIFO. Similarly, if the DWC\_ssi makes a receive request to this channel, four data items are read from the DWC\_ssi receive FIFO. Three separate requests must be made to this DMA channel before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



The source and destination transfer width settings in the DW\_ahb\_dmac – DMA.CTLx.SRC\_TR\_WIDTH and DMA.CTLx.DEST\_TR\_WIDTH – should be set to 3'b010 because the DWC\_ssi FIFOs are 32 bits wide.

When the block size programmed into the DMA Controller is not a multiple of the burst transaction length, as shown in [Figure 2-51](#), a series of burst transactions followed by single transactions are needed to complete the block transfer.

**Figure 2-51 Breakdown of DMA Transfer into Single and Burst Transactions**



### 2.12.1.2 Transmit Watermark Level and Transmit FIFO Underflow

During DWC\_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.

### 2.12.1.3 Choosing the Transmit Watermark Level

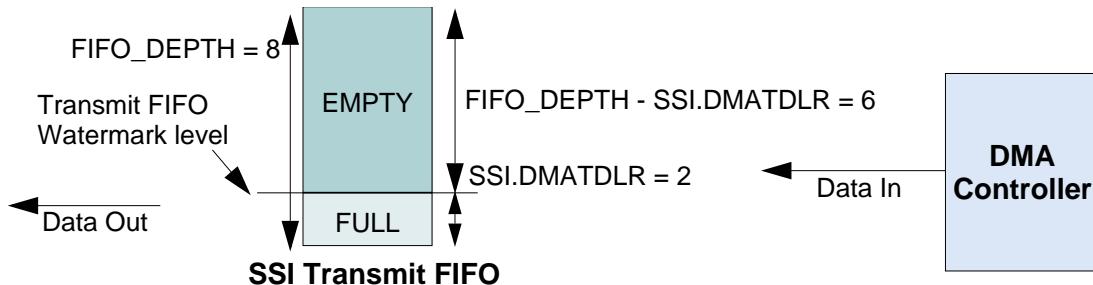
Consider the example where the assumption is made:

```
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.

## Case 1: DMATDLR = 2

**Figure 2-52 Case 1 Watermark Levels**



- Transmit FIFO watermark level =  $SSI.DMATDLR = 2$
- $DMA.CTLx.DEST\_MSIZE = FIFO\_DEPTH - SSI.DMATDLR = 6$
- $SSI \text{ transmit FIFO}_DEPTH = 8$
- $DMA.CTLx.BLOCK\_TS = 30$

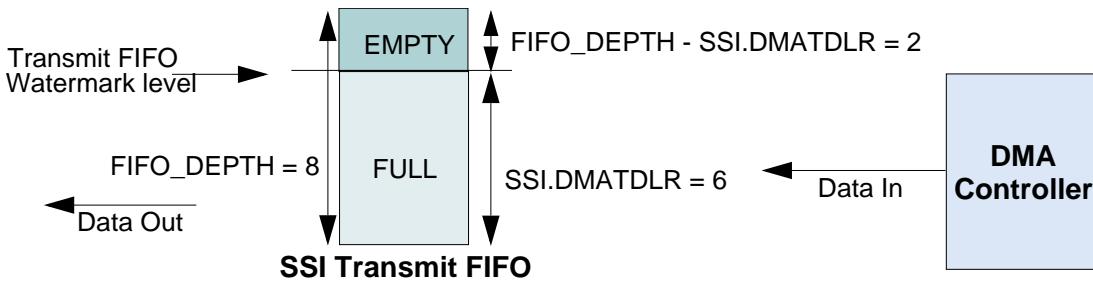
Therefore, the number of burst transactions needed equals the block size divided by the number of data items per burst:

$$DMA.CTLx.BLOCK\_TS / DMA.CTLx.DEST\_MSIZE = 30 / 6 = 5$$

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.

## Case 2: DMATDLR = 6

**Figure 2-53 Case 2 Watermark Levels**



- Transmit FIFO watermark level =  $SSI.DMATDLR = 6$
- $DMA.CTLx.DEST\_MSIZE = FIFO\_DEPTH - SSI.DMATDLR = 2$
- $SSI \text{ transmit FIFO}_DEPTH = 8$
- $DMA.CTLx.BLOCK\_TS = 30$

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

$$DMA.CTLx.BLOCK\_TS / DMA.CTLx.DEST\_MSIZE = 30 / 2 = 15$$

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 manager interface to the highest priority manager 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.

#### 2.12.1.4 Selecting DEST\_MSIZEx and Transmit FIFO Overflow

As can be seen from [Figure 2-53 on page 81](#), programming DMA.CTLx.DEST\_MSIZEx 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:

$$\text{DMA.CTLx.DEST_MSIZEx} \leq \text{SSI.FIFO_DEPTH} - \text{SSI.DMATDLR} \quad (1)$$

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\_MSIZEx. Thus, the transmit FIFO may be full, but not overflowed, at the completion of the burst transaction.

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

$$\text{DMA.CTLx.DEST_MSIZEx} = \text{SSI.FIFO_DEPTH} - \text{SSI.DMATDLR} \quad (2)$$

This is the setting used in [Figure 2-51 on page 80](#).

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 will not be 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.

#### 2.12.1.5 Receive Watermark Level and Receive FIFO Overflow

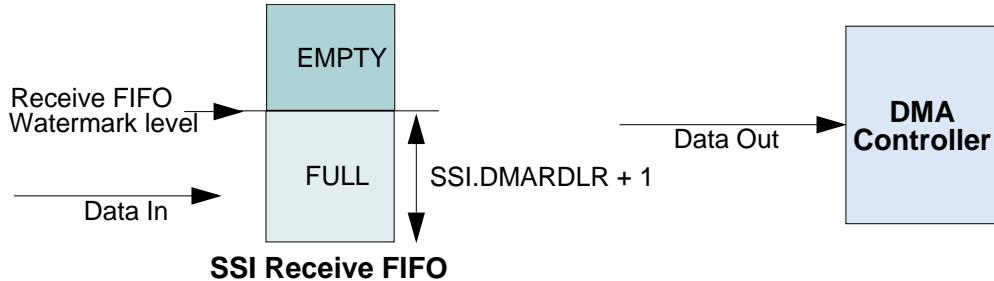
During DWC\_ssi serial transfers, receive FIFO requests are made to the DW\_ahb\_dmac whenever the number of entries in the receive FIFO is at or above the DMA Receive Data Level Register; that is, DMARDLR+1. This is known as the watermark level. The DW\_ahb\_dmac responds by fetching a burst of data from the receive FIFO buffer of length CTLx.SRC\_MSIZEx.

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 overflows. To prevent this condition, you must correctly set the watermark level.

### 2.12.1.6 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 2-54](#). It is a trade-off between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

**Figure 2-54 DWC\_ssi Receive FIFO**



### 2.12.1.7 Selecting SRC\_MSIZEx and Receive FIFO Underflow

As seen in [Figure 2-54](#), 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) 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_MSIZEx` – the receive FIFO may be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, `DMA.CTLx.SRC_MSIZEx` should be set at the watermark level; that is:

$$\text{DMA.CTLx.SRC_MSIZEx} = \text{SSI.DMARDLR} + 1 \quad (3)$$

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



The receive FIFO will not be 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.

### 2.12.1.8 Handshaking Interface Operation

The following sections describe the DWC\_ssi handshaking interface.

`dma_tx_req`, `dma_rx_req`

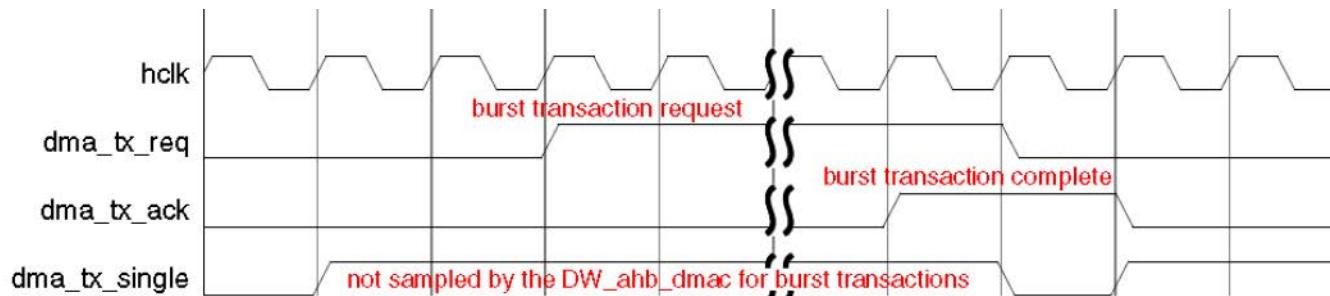
The request signals for source and destination, `dma_tx_req` and `dma_rx_req` signals are activated when their corresponding FIFOs reach the watermark levels as described earlier.

The DW\_ahb\_dmac uses rising-edge detection of the `dma_tx_req/dma_rx_req` signal 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 DWC\_ssi de-asserts the burst request signals, `dma_tx_req/dma_rx_req`, until `dma_tx_ack/dma_rx_ack` signal is de-asserted by the DW\_ahb\_dmac.

When the DWC\_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 2-55 shows a timing diagram of a burst transaction where `pclk = hclk`.

**Figure 2-55 Burst Transaction**



The handshaking loop is as follows:

- `dma_tx_req/dma_rx_req` asserted by DWC\_ssi
- `dma_tx_ack/dma_rx_ack` asserted by DW\_ahb\_dmac
- `dma_tx_req/dma_rx_req` de-asserted by DWC\_ssi
- `dma_tx_ack/dma_rx_ack` de-asserted by DW\_ahb\_dmac
- `dma_tx_req/dma_rx_req` re-asserted by DWC\_ssi, if back-to-back transaction is required

Two things to keep in mind:

- The burst request lines, `dma_tx_req/dma_rx_req`, once asserted remain asserted until their corresponding `dma_tx_ack/dma_rx_ack` signal is received even if the respective FIFO's 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.

### `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 2-51 on page 80. 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 DWC\_ssi are not sampled by the DW\_ahb\_dmac otherwise. This is illustrated in the following example.

Consider an example where the receive FIFO channel of the DWC\_ssi is as follows:

```
DMA.CTLx.SRC_MSIZEx = SSI.DMARDLR + 1 = 4
```

```
DMA.CTLx.BLOCK_TS = 12
```

For the example in [Figure 2-50](#) on page [79](#), 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 thrice during the DWC\_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:

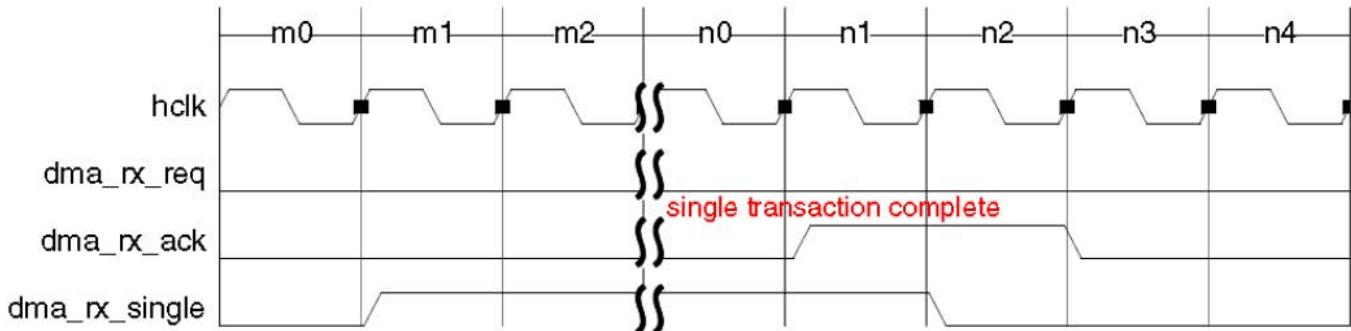
```
DMA.CTLx.SRC_MSIZEx = SSI.DMARDLR + 1 = 4
```

```
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 2-56](#) shows a single transaction.

**Figure 2-56 Single Transaction**



The handshaking loop is as follows:

- `dma_tx_single/dma_rx_single` asserted by DWC\_ssi
- `dma_tx_ack/dma_rx_ack` asserted by DW\_ahb\_dmac
- `dma_tx_single/dma_rx_single` de-asserted by DWC\_ssi
- `dma_tx_ack/dma_rx_ack` de-asserted by DW\_ahb\_dmac

## 2.12.2 Including DMA Handshaking Interface Signals

To include DMA handshaking interface signals in DWC\_ssi, click the check-box for the **Include DMA Interface?** field using the **Top Level Parameters** tab during the Specify Configuration Activity in coreConsultant.

### 2.12.3 Signals Related to DMA Controller Interface

The following signals are related to DMA Controller Interface:

- dma\_tx\_req
- dma\_rx\_req
- dma\_tx\_ack
- dma\_rx\_ack
- dma\_tx\_single
- dma\_rx\_single

For more information about these signals, see the *Signals Description* chapter.

### 2.12.4 Registers Related to DMA Controller Interface

The following are the registers related to DMA Controller Interface:

- DMACR
- DMATDLR
- DMARDLR

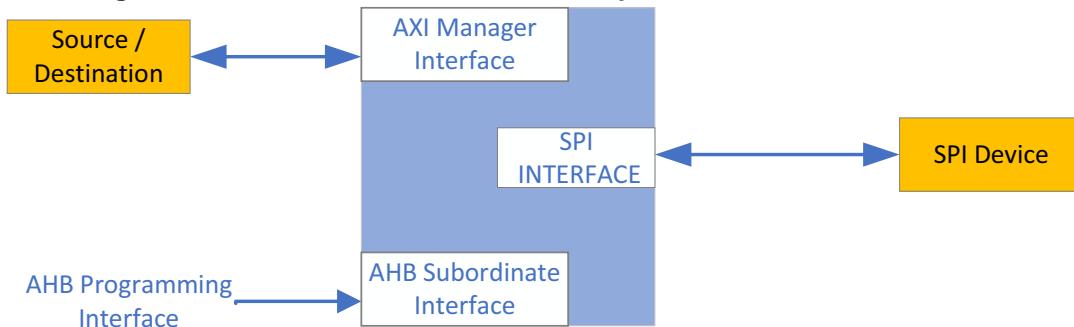
For more information about these registers, see the *Registers Descriptions* chapter.

## 2.13 Internal DMA Feature for SPI Operation

### 2.13.1 Description of DMA Feature for SPI Operation

The DWC\_ssi internal DMA feature is responsible for data transfer from an internal memory (or on chip memory) to an external SPI device, and vice versa. DWC\_ssi uses AXI manager interface on one side and SPI interface on the other, which acts as a source or a destination depending on the programming.

**Figure 2-57 Block Diagram for Internal DMA Feature for SPI Operation**



After DWC\_ssi has been programmed, it takes care of completing the transfer between the internal memory and external SPI device.



- The internal DMA operation is supported only in Enhanced SPI (Dual/Quad/Octal/Dual Octal) and Standard SPI (with non-zero instruction or address length) mode of operation.
- When internal DMA is enabled, any read or write operation to Data register (DR) results in error response from the AHB subordinate interface.
- Data Mask feature is not supported in internal DMA mode.
- For internal DMA operation, the address length on SPI interface can be maximum of 32 bits.

### 2.13.2 SPI Write Operation

In SPI write operation, the SPI device acts as destination and internal memory acts as a source. DWC\_ssi fetches the data using the AXI manager interface and transmits the data on the SPI device. Following is the programming requirement for this operation:

1. Program `CTRLR0.TMOD = 1` (SPI write).
2. Program the `CTRLR0.SPI_FRF` register field with the frame format for SPI transfer.
3. Set transfer characteristics for SPI operation in the `CTRLR0.BAUDR`, `TXFTLR.TXFTHR`, and `SPI_CTRLR0` registers.
4. Program `CTRLR1.NDF` with the amount of data to be transmitted to the device.
5. Enable DMA transfers using the `DMACR.IDMAE` register field.
6. Set the source transfer width in the `DMACR.ATW` field.
7. Set AXI transfer characteristics using the `DMACR.AID`, `DMACR.AINC`, `DMACR.APROT`, `DMACR.ACACHE`, and `AXIARLEN.ARLEN` register fields.
8. Set Source address in the `AXIAR0`, and `AXIAR1` registers.
9. Set device destination address in the `SPIAR` register.

10. Set instruction code in the `SPIDR.SPI_INST` register field.

11. Enable `DWC_ssi` using the `SSIENR` register.

After `DWC_ssi` is enabled, `DWC_ssi` starts fetching the data from the source using transfer characteristics set for the AXI interface. AXI interface fetches the data using the `AXIARLEN.ARLEN` register field burst length until the space left in FIFO is less than the `AXIARLEN.ARLEN` register field decode value. To get the maximum utilization of transmit FIFO, it is recommended to use the `AXIARLEN.ARLEN` register field value such that the amount of data read in one burst is multiple of FIFO depth. If the `DMACR.AINC` register field is set to 0, AXI address in the `AXIAR0` register increments to the next address after every AXI burst request.



**Note** Before starting the SPI write operation from the controller, ensure that write is enabled in the SPI target device, and it is not busy; otherwise the transfer may not complete successfully on SPI target device.

---

After `DWC_ssi` has enough amount of data in the transmit FIFO (specified in `TXFTLR.TXFTHR` field), it starts the SPI operation using the instruction code set in the `SPIDR.SPI_INST` register field and device address set in the `SPIAR.SDAR` register field. The whole block transfer is completed in single block.

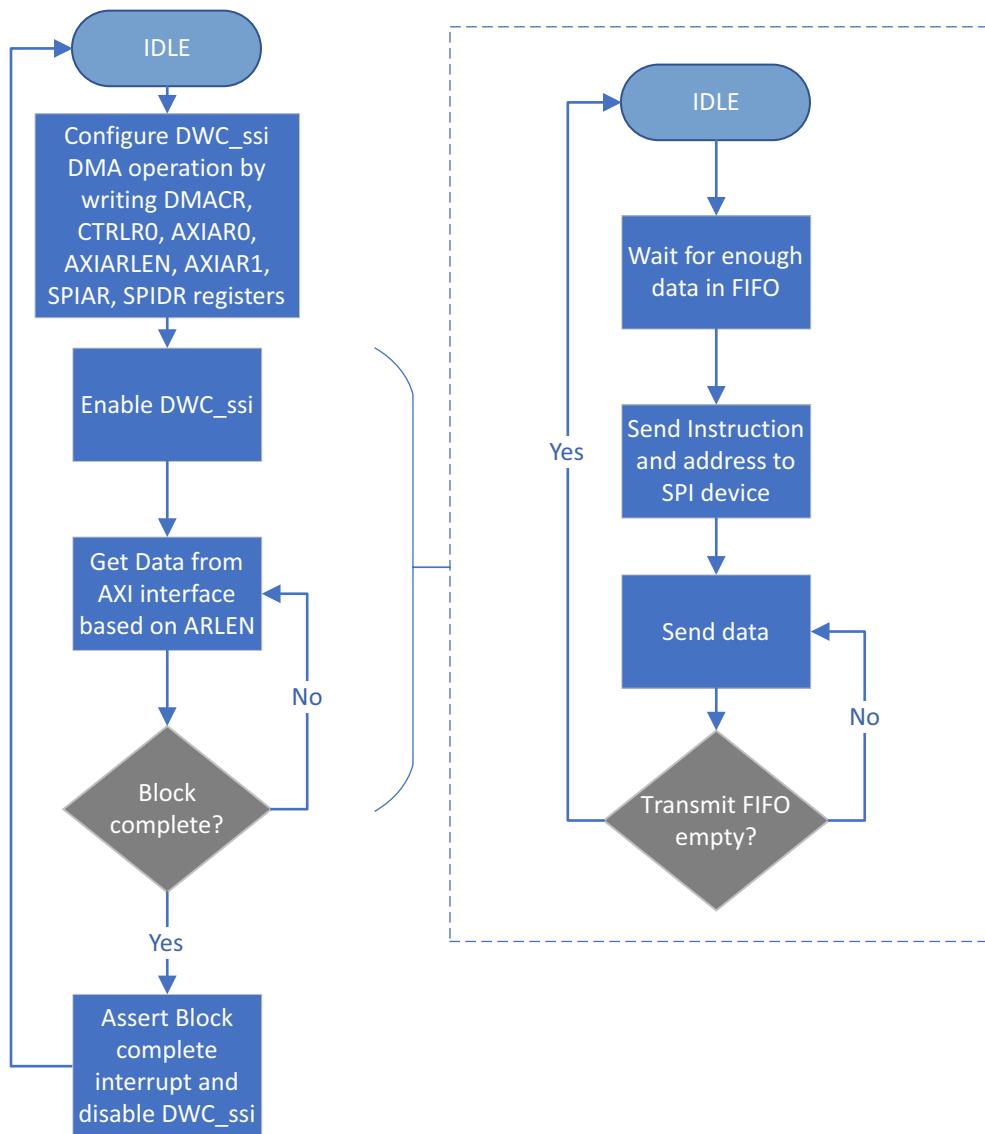
- If FIFO goes empty and clock stretching is not enabled, `DWC_ssi` de-selects the Serial Target device and asserts transmit FIFO underflow interrupt.
- If the clock stretching is enabled, `DWC_ssi` stretches the clock until the data is available.



**Note** In internal DMA mode, the `TXFTLR.TXFTHR` register field sets the minimum amount of data frames available in the FIFO before `DWC_ssi` starts the transfer.

---

After `DWC_ssi` transmits the required amount of data to the device, `DWC_ssi` clears the `SSIENR` register, and interrupt (`ssi_done_intr(_n)`) is provided to software.

**Figure 2-58 DMA Write Operation to SPI Device**

### 2.13.2.1 Examples for SPI Write Transfer

**Total amount of data to be transferred is a multiple of the amount of data per AXI burst**

In this example, the total amount of data to be fetched from AXI interface is a multiple of the amount of data fetched in a single burst transfer. The programming is as follows:

- CTRLR1.NDF = 11 (total 12 data frames to fetch)
- DFS = 31 (data frame size of 32)
- ATW = 10 (AXI transfer size = 4 bytes)
- ARLEN = 3 (4 beats per AXI burst)

In this scenario,

The amount of data per AXI burst = 12 bytes (ATW \* ARLEN)  
 Total amount of data to be transferred = 48 bytes (NDF\* DFS)

So, AXI interface completes this transfer by performing three bursts and SPI interface sends that data to the target device in a single transfer.

#### **Total amount of data to be transferred is not a multiple of the amount of data per AXI burst**

Consider the following values for this example:

- CTRLR1.NDF = 12 (total 13 data frames to fetch)
- DFS = 31 (data frame size of 32)
- ATW = 10 (AXI transfer size = 4 bytes)
- ARLEN = 3 (4 beats per AXI burst)

In this scenario,

Amount of data per AXI burst = 12 bytes (ATW \* ARLEN)  
 Total amount of data to be transferred = 52 bytes (NDF\* DFS)

This transfer is completed by performing three bursts of length four and one burst of length 1.

#### **Data Frame size is not a multiple of AXI burst size**

Consider the following programming for this example:

- CTRLR1.NDF = 24 (total 25 data frames to fetch)
- DFS = 15 (data frame size of 16)
- ATW = 10 (AXI transfer size = 4 bytes)
- ARLEN = 3 (4 beats per AXI burst)

In this scenario,

Amount of data per AXI burst = 12 bytes (ATW \* ARLEN)  
 Total amount of data to be transferred = 50 bytes (NDF\* DFS)

This transfer can be completed by performing three bursts of length four and one burst of length two, with arsize = 3'b000 (1 byte).



**Note** When the remaining data in the transfer is less than programmed AXI size (ARSIZE), then remaining bytes are fetched in a single transfer with ARSIZE = 3'b000.

### **2.13.3 SPI Read Operation**

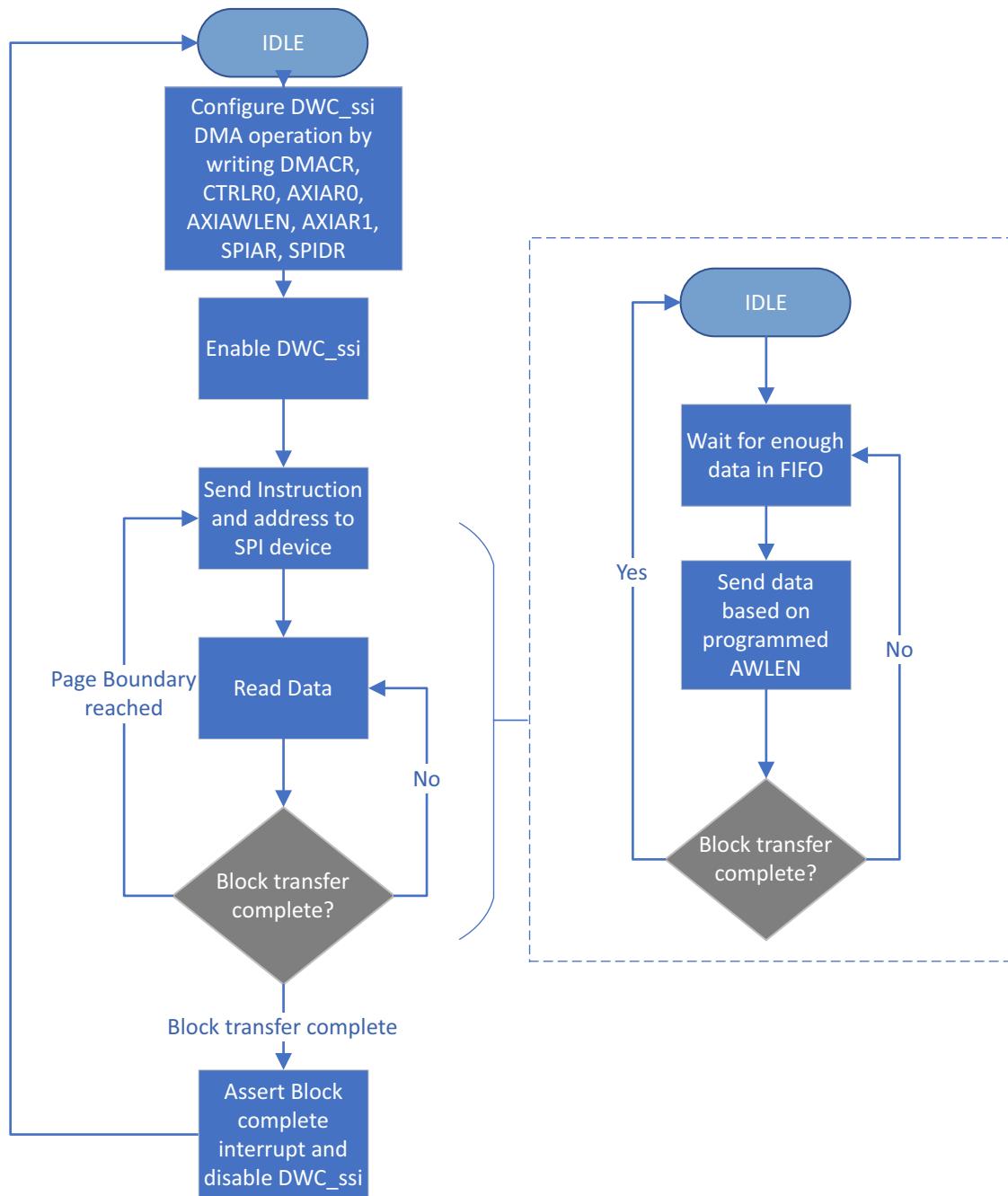
In SPI read operation, SPI device acts as source and internal memory acts as a destination device. DWC\_ssi fetches the data using the SPI interface, and transmits the data on the AXI interface. Following is the programming requirement for this operation:

1. Program CTRLR0 .TMOD = 2'b10 (SPI read).
2. Program the CTRLR0 .SPI\_FRF register field with the frame format for SPI transfer.
3. Set transfer characteristics for SPI operation in the CTRLR0, BAUDR and SPI\_CTRLR0 registers.
4. Program the CTRLR1 .NDF register field with the amount of data to be transmitted to the device.

5. Enable DMA transfers using the DMACR.IDMAE register field.
6. Set the source transfer width in the DMACR.ATW register field.
7. Set AXI transfer characteristics using the DMACR.AID, DMACR.AINC, DMACR.APROT, DMACR.ACACHE, and AXIAWLEN.AWLEN register fields.
8. Set source address in the AXIAR0, and AXIAR1 registers.
9. Set device destination address in the SPIAR registers.
10. Set instruction code in the SPIDR.SPI\_INST register field.
11. Enable DWC\_ssi using the SSIENR register.

After DWC\_ssi is enabled, DWC\_ssi starts SPI transfer using instruction code from the SPI\_DR.SPI\_INST register field, and address set in SPIAR.SDAR register field. This data is stored in the receive FIFO, and then transmitted to the AXI interface. After the receive FIFO receives the data greater than or equal to AXIAWLEN.AWLEN register field, DWC\_ssi starts the operation on the AXI interface. This process continues until the whole block of data (CTRLR1.NDF) is transmitted to the AXI interface. If DMACR.AINC register field is set to 0, then AXI address in AXIAR0 increments to the next address after every AXI burst request.

If Receive FIFO becomes full and clock stretching is not enabled, then DWC\_ssi de-selects the serial target device and starts a new read transfer on the next contiguous address. If the clock stretching is enabled, then DWC\_ssi stretches the clock and continues to receive until FIFO has enough space. After DWC\_ssi receives the required amount of data from the device, DWC\_ssi clears the SSIENR register and interrupt (ssi\_done\_intr(\_n)) is provided to software.

**Figure 2-59 DMA Read Operation to SPI Device**

### 2.13.3.1 Examples for SPI Read Transfer

**Total amount of data to be transferred is a multiple of the amount of data per AXI burst**

In this example, the total amount of data to be fetched from AXI interface is a multiple of the amount of data fetched in a single burst transfer. The programming is as follows:

- CTRLR1.NDF = 11 (total 12 data frames to fetch)
- DFS = 31 (data frame size of 32)
- ATW = 10 (AXI transfer size = 4 bytes)
- AWLEN = 3 (4 beats per AXI burst)

In this scenario,

```
Amount of data per AXI burst = 12 bytes (ATW * ARLEN)
Total Amount of data to be transferred = 48 bytes (NDF* DFS)
```

So, AXI interface completes this transfer by performing three bursts and SPI interface fetches that data to the target device using in a single transfer.

### Total amount of data to be transferred is not a multiple of the amount of data per AXI burst

Consider the following programming for this example:

- CTRLR1.NDF = 12 (total 13 data frames to fetch)
- DFS = 31 (data frame size of 32)
- ATW = 10 (AXI transfer size = 4 bytes)
- AWLEN = 3 (4 beats per AXI burst)

In this scenario,

```
Amount of data per AXI burst = 12 bytes (ATW * ARLEN)
Total amount of data to be transferred = 52 bytes (NDF* DFS)
```

This transfer is completed by performing three bursts of four beats and one burst of length one beat.

Data Frame size is not a multiple of AXI burst size Consider the following programming for this example:

- CTRLR1.NDF = 24 (total 25 data frames to fetch)
- DFS = 15 (data frame size of 16)
- ATW = 10 (AXI transfer size = 4 bytes)
- AWLEN = 3 (4 beats per AXI burst)

In this scenario, Amount of data per AXI burst = 12 bytes (ATW \* ARLEN) Total amount of data to be fetched = 50 bytes (NDF\* DFS) This transfer is completed by performing three bursts of four beats and one burst of two beats with awsize = 3'b000 (1 byte).



If the remaining data in the transfer is less than programmed AXI size (AWSIZE), then remaining bytes is sent in a single transfer with AWSIZE=3'b000.

### 2.13.4 AXI Manager Interface

DWC\_ssi uses AXI interface to perform internal DMA operation. AXI interface consists of all five channels, a separate clock (aclk), and a reset (aresetn) signal. AXI interface can be configured to support AXI3/AXI4 interface using the SSIC\_AXI\_INTF\_TYPE parameter.

All control signals for AXI interface are derived from the programming registers, the table below describes the mapping of each control signal with internal register field.

**Table 2-1 AXI Control Signals Mapping**

Signal Name	Register Field
<b>For Read transfers (CTRLR0.TMOD = 10)</b>	
awid/wid	DMACR. AID
awaddr[31:0]	AXIAR0 <b>Note:</b> The Address should be aligned with awsize address boundary.
awaddr[63:32]	AXIAR1
awszie	DMACR. ATW
awburst	DMACR.AINC If AINC=0; INCR burst If AINC=1; FIXED burst <b>Note:</b> WRAP burst type is not supported
awlock	awlock signal is set to 2'b00 as DWC_ssi does not support locked or exclusive accesses.
awcache	DMACR.ACACHE
awprot	DMACR.APROT
<b>For write transfers (CTRLR0.TMOD = 01)</b>	
arid	DMACR. AID
araddr[31:0]	AXIAR0 <b>Note:</b> The Address should be aligned with arsize address boundary.
araddr[63:32]	AXIAR1
arszie	DMACR. ATW
arburst	DMACR.AINC If AINC=0; INCR burst If AINC=1; FIXED burst <b>Note:</b> WRAP burst type is not supported
arlock	awlock signal is set to 2'b00 as DWC_ssi does not support locked or exclusive accesses.
arcache	DMACR.ACACHE
arprot	DMACR.APROT

AXI interface can receive error response from Write Response or Read data channel. In such situation DWC\_ssi terminates the transfer, immediately by de-asserting the chip select signal disables DWC\_ssi, and asserts the `ssi_axie_intr(_n)` signal. The data present in the internal FIFO is flushed and DWC\_ssi is disabled. Also, DWC\_ssi updates the SR.CMPLTD\_DF field with the amount of data which was successfully

transferred in the last DMA transfer. You can use this information to determine where exactly the error occurred, and the amount of data transferred successfully.



**Note** The CMPLTD\_DF of status register (SR) may not represent the correct value if the transfer is terminated by programming SSIENR bit to 0.

### 2.13.5 Signals Related to Internal DMA Feature

The following are the signals related to internal DMA feature:

- aresetn
- awid[SSIC\_AXI\_IDW-1:0]
- awaddr[SSIC\_AXI\_AW-1:0]
- awlen[SSIC\_AXI\_BLW-1:0]
- awsizer[2:0]
- awburst[1:0]
- awlock[1:0]
- awcache[3:0]
- awprot[2:0]
- awuser[SSIC\_AXI\_AWUW-1:0]
- awvalid
- Awready
- wid[SSIC\_AXI\_IDW-1:0]
- wdata[SSIC\_AXI\_DW-1:0]
- wstrb[SSIC\_AXI\_WSW-1:0]
- Wlast
- Wvalid
- Wready
- wuser[SSIC\_AXI\_WUW-1:0]
- bid[SSIC\_AXI\_IDW-1:0]
- bresp[1:0]
- bvalid
- buser[SSIC\_AXI\_BUW-1:0]
- bready
- arid[SSIC\_AXI\_IDW-1:0]
- araddr[SSIC\_AXI\_AW-1:0]
- arlen[SSIC\_AXI\_BLW-1:0]
- arsize[2:0]
- arbust[1:0]

- arlock[1:0]
- arcache[3:0]
- arprot[2:0]
- arvalid
- aruser[SSIC\_AXI\_ARUW-1:0]
- arready
- rid[SSIC\_AXI\_IDW-1:0]
- rdata[SSIC\_AXI\_DW-1:0]
- rresp[1:0]
- rlast
- rvalid
- ruser[SSIC\_AXI\_RUW-1:0]
- rready

For more information about these signals, see the *Signals Description* chapter.

### 2.13.6 Registers Related to Internal DMA Feature

The following are the registers related to internal DMA feature:

- CTRLR0
- CTRLR1
- BAUDR
- SSIENR
- SER
- DMACR
- AXIAWLEN
- AXIARLEN
- SPI\_CTRLR0
- SPIDR
- SPIAR
- AXIAR0
- AXIAR1

For more information about these registers, see the *Registers Descriptions* chapter.

### 2.13.7 Enabling Internal DMA Feature

To enable DMA for DWC\_ssi, choose the **true** option for the **Include DMA interface?** field using the **Top Level Parameters** tab during the Specify Configuration Activity in coreConsultant.

## 2.14 SSI Interrupts

DWC\_ssi supports combined and individual interrupt requests, each of which can be masked. The combined interrupt request is the ORed result of all other DWC\_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.



**Note** The Safety interrupts cannot be masked.

### 2.14.1 Description of SSI Interrupts

The DWC\_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 AHB access attempts to write into the transmit FIFO after it has been completely filled. When set, data written from the AHB is discarded. This interrupt remains set until you read the transmit FIFO error interrupt clear register (`TXEICR`).
- Transmit FIFO underflow interrupt (`ssi_txu_intr`): Present only when Internal DMA mode or SPI bridge mode or XIP write transfer is enabled. The interrupt is set when transmit FIFO pop happens from serial FSM and FIFO is empty. This interrupt remains set until you read the transmit FIFO error interrupt clear register (`TXEICR`).
- 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 AHB 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-Controller Contention interrupt (`ssi_mst_intr`): Present only when the DWC\_ssi component is configured as a serial-controller device. The interrupt is set when another serial controller on the serial bus selects the DWC\_ssi controller as a serial-target 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- interrupt clear register (`MSTICR`).
- SPI transmit error interrupt (`ssi_spite_intr`): Present only when the Dynamic wait state feature is enabled. The interrupt is set, if SPI Controller fails to get a READY status from the target until the amount of time defined in the `SPI_CTRLR1.MAX_WS` field. It stop the SPI transfer and the FIFO is flushed (in case of write operation). This interrupt remains set until you read the SPI transmit error interrupt clear register (`SPITECR`).

- SPI Controller error interrupt (`ssi_spime_intr`): Present only when SPI Bridge mode is enabled. The interrupt is set, When SPI Controller performs a transfer not expected by the SPI target. This interrupt remains set until you read the SPI Controller error interrupt clear register (`SPIMECR`).
- AHB error interrupt (`ssi_ahbe_intr`): Present only when SPI Bridge mode is enabled. The interrupt is set, when AHB Manager receives an error response. This interrupt remains set until you read the AHB error interrupt clear register (`AHBECR`).
- SSI done interrupt (`ssi_done_intr`): Present only when the Internal DMA mode is enabled. The interrupt is set, when internal DMA transfer completes. This interrupt remains set until you read the SSI done interrupt clear register (`DONECR`).
- AXI Error interrupt (`ssi_axie_intr`): Present only when the Internal DMA mode is enabled. The interrupt is set, when AXI Manager interface receives an error response. This interrupt remains set until you read the AXI Error interrupt clear register (`AXIECR`).
- Combined interrupt request (`ssi_intr`): OR'ed result of all the interrupt requests after masking. To mask this interrupt signal, you must mask all other `DWC_ssi` interrupt requests.

## 2.14.2 Description of Safety Interrupts

The Safety specific Interrupts of `DWC_ssi` are described as follows:

- SSI FSM one-hot encoding interrupt (`ssi_sfty_fp_intr`): Present only when SSI Safety Configuration 1 is enabled. The interrupt is set when any of the FSM protected using the FSM one-hot feature detects an error. The interrupt remains set until you write in the FSM one-hot Error Clear field (`FSMPEC`) in the SSI Safety Interrupt Clear register (`SAFETYICR`).
- SSI FSM Time-out interrupt (`ssi_sfty_fto_intr`): Present only when SSI Safety Configuration 1 is enabled. The interrupt is set when any of the FSM protected using the FSM Time-out feature detects an error. The interrupt remains set until you write in the SSI Clock Domain FSM Timeout Error Clear field (`SSIFSMTOC`) in the SSI Safety Interrupt Clear register (`SAFETYICR`).
- SSI AXI Interface Parity interrupt (`ssi_sfty_ap_intr`): Present only when SSI Safety Configuration 1 is enabled. The interrupt is set when any of the AXI interface detects a parity error. The interrupt remains set until you write in the AXI Parity Error Clear field (`AXIPEC`) in the SSI Safety Interrupt Clear register (`SAFETYICR`).
- SSI Data Path Parity interrupt (`ssi_sfty_dpe_intr`): Present only when SSI Safety Configuration 1 is enabled. The interrupt is set when any of the data path parity logic detects an error. The interrupt remains set until you write in the Data Path Parity Error Clear field (`DPPCC`) in the SSI Safety Interrupt Clear register (`SAFETYICR`).
- SSI Logic Duplication interrupt (`ssi_sfty_dup_intr`): Present only when SSI Safety Configuration 2 is enabled. The interrupt is set when any of the logic protected using duplication feature detects an error. The interrupt remains set until you write in the Logic Duplication Error Clear field (`LDXIPEC`) in the SSI Safety Interrupt Clear register (`SAFETYICR`).
- SSI Data Path ECC Uncorrectable interrupt (`ssi_sfty_dp_ecc_ue_intr`): Present only when SSI Safety Configuration 2 is enabled. The interrupt is set when Data path ECC protection feature detects an uncorrectable error. The interrupt remains set until you write in the Data path ECC Uncorrectable Error Clear field (`DPUEC`) in the SSI Safety Interrupt Clear register (`SAFETYICR`).
- SSI Data Path ECC Correctable interrupt (`ssi_sfty_dp_ecc_ce_intr`): Present only when SSI Safety Configuration 2 is enabled. The interrupt is set when Data path ECC protection feature detects a correctable error. The interrupt remains set until you write in the Data Path ECC Correctable Error Clear field (`DPCEC`) in the SSI Safety Interrupt Clear register (`SAFETYICR`).
- SSI TMR interrupt (`ssi_sfty_tmr_intr`): Present only when SSI Safety Configuration 2 is enabled. The interrupt is set when TMR Protection feature detects an error. The interrupt remains set until you

write in the TMR Error Status Clear field (TMRESC) in the SSI Safety Interrupt Clear register (SAFETYICR).

- SSI Register Space Protection Error interrupt (ssi\_sfty\_rs\_intr): Present only when SSI Safety Configuration 1/2 is enabled. The interrupt is set when Register Space Protection feature detects a error. The interrupt remains set until you write in the Register Space Error Clear field (REGERRC) in the SSI Safety Interrupt Clear register (SAFETYICR).
- SSI AHB Interface Address Parity interrupt (ssi\_sfty\_addr\_sp\_intr): Present only when AHB subordinate Interface Parity Protection feature is enabled for address bus. The interrupt is set when AHB Address Parity Protection feature detects an error. The interrupt remains set until you write in Register Bus Interface Address Parity Error Clear field (SLVIFAPEC) in the SSI Safety Interrupt Clear register (SAFETYICR).
- SSI AHB Interface Data Parity interrupt (ssi\_sfty\_data\_sp\_intr): Present only when AHB subordinate Interface Parity protection feature is enabled for data bus. The interrupt is set when AHB Data Parity Protection feature detects an error. The interrupt remains set until you write in the Register Bus Interface Data Parity Error Clear field (SLVIFDPEC) in the SSI Safety Interrupt Clear register (SAFETYICR).
- SSI AXI Interface Valid/Ready Protection error interrupt (ssi\_sfty\_vrp\_intr): Present only when AXI Valid/Ready protection feature is enabled. The interrupt is set when AXI Valid/Ready Protection feature detects an error. The interrupt remains set until you write in the AXI Valid/Ready Parity Error Clear field (AXIVRPEC) in the SSI Safety Interrupt Clear register (SAFETYICR).
- SSI AXI ECC Uncorrectable interrupt (ssi\_sfty\_axi\_ecc\_ue\_intr): Present only when AXI ECC feature is enabled. The interrupt is set when AXI interface ECC Protection feature detects an uncorrectable error. The interrupt remains set until you write in the AXI interface ECC Uncorrectable Error Clear field (AXIUEC) in the SSI Safety Interrupt Clear register (SAFETYICR).
- SSI AXI ECC Correctable interrupt (ssi\_sfty\_axi\_ecc\_ce\_intr): Present only when AXI ECC feature is enabled. The interrupt is set when AXI interface ECC protection feature detects a correctable error. The interrupt remains set until you write in the AXI Interface ECC Correctable Error Clear field (AXICEC) in the SSI Safety Interrupt Clear register (SAFETYICR).

DWC\_ssi also supports combined interrupt for the safety feature. The combined safety interrupts are carried through two signals:

- SSI combined safety unrecoverable error (ssi\_sfty\_ue\_intr): This signal is ORed version of all unrecoverable safety errors detected by the various safety mechanisms. This included following signals:
  - ssi\_sfty\_fp\_intr
  - ssi\_sfty\_fto\_intr
  - ssi\_sfty\_ap\_intr
  - ssi\_sfty\_dpe\_intr
  - ssi\_sfty\_ld\_intr
  - ssi\_sfty\_dp\_ecc\_ue\_intr
  - ssi\_sfty\_rs\_intr
  - ssi\_sfty\_vrp\_intr
  - ssi\_sfty\_axi\_ecc\_ue\_intr
- SSI combined recoverable error (ssi\_sfty\_re\_intr): This signal is ORed version of all recoverable safety errors detected by the various safety mechanisms. This included following signals:

- ❑ ssi\_sfty\_dp\_ecc\_ce\_intr
- ❑ ssi\_sfty\_axi\_ecc\_ce\_intr
- ❑ ssi\_sfty\_tmr\_intr
- ❑ ssi\_sfty\_addr\_sp\_intr
- ❑ ssi\_sfty\_data\_sp\_intr

### 2.14.3 Configuring Interrupt Pinout

To configure DWC\_ssi to include the individual or combined interrupts, choose the **Configure interrupt pinout** field using the **Top Level Parameters** tab during the Specify Configuration Activity in coreConsultant.

This field provides the following options:

- Combined Interrupt - Specifies that a single combined interrupt (the logical OR of all DWC\_ssi interrupt outputs) must appear as outputs of the design.
- Individual Interrupt - Specifies that each individual interrupt must appear as a separate output pin on the macrocell. When configured as a Serial Controller, there are six individual interrupts. When configured as a Serial Target, there are five individual interrupts. By default, the interrupt pinout is configured as an individual interrupt.

### 2.14.4 Signals Related to SSI Interrupts

The following signals are related to SSI Interrupts:

- ssi\_txe\_intr
- ssi\_txo\_intr
- ssi\_rxf\_intr
- ssi\_rxo\_intr
- ssi\_rxu\_intr
- ssi\_mst\_intr
- ssi\_intr

For more information about these signals, see the *Signals Description* chapter.

### 2.14.5 Registers Related to SSI Interrupts

The following are the registers related to SSI Interrupts:

- TXEICR
- RXOICR
- RXUICR
- MSTICR
- ISR RISR ICR
- IMR

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.15 Boot Mode

DWC\_ssi can be set to operate immediately after reset by setting the SSIC\_BOOT\_MODE\_EN parameter to 1. By setting the SSIC\_BOOT\_MODE\_EN parameter, the SSIENR.SSIC\_EN field is set to 1 on reset and DWC\_ssi can accept incoming write commands on the data register immediately after reset. This removes the requirement of programming DWC\_ssi after each reset.

### 2.15.1 Description of Boot Mode

The default values of DWC\_ssi control registers can be set using SSIC\_DFLT\_\* (Register Default Value) parameters listed in the *Parameter Description* chapter. Depending on the default values, the boot behavior of DWC\_ssi are set.

For example, if the requirement is to use DWC\_ssi to program a serial target attached to target port 0 using the Octal SPI mode, the following configuration must be set:

- SSIC\_BOOT\_MODE\_EN = 1 (DWC\_ssi reset with SSI\_EN bit set to 1)
- SSIC\_DFLT\_SLV = 16'h1 (Target 0 selected)
- SSIC\_DFLT\_SPI\_FRF = 2'b11 (SPI frame format set to Octal mode)
- SSIC\_DFLT\_TMOD = 2'b01 (Transmit mode set to Write)
- SSIC\_DFLT\_INST\_L = 2'b10 (Instruction length = 16 bits)
- SSIC\_DFLT\_ADDR\_L = 4'b0100 (Address length = 16 bits)
- SSIC\_DFLT\_TRANS\_TYPE = 2'b00 (Instruction and address to be sent on normal SPI mode)

By setting the mentioned parameters, software writes on the data register just after the reset. Once DWC\_ssi receives the required amount of data, DWC\_ssi starts a write transaction on target 0. This way, reset values of DWC\_ssi control registers can be set and the controller can work just after reset removing the requirement on any software intervention.

Similarly, an XIP transfer can also be initiated just after the reset by initiating an AHB read request with the xip\_en signal driven to 1.



**Note** When DWC\_ssi is used in coreAssembler with the Boot Mode enabled, the UVM RAL Subsystem Ping Test is not applicable because coreAssembler does not find a valid register to perform a ping test.

### 2.15.2 Enabling Boot Mode

To enable boot mode for DWC\_ssi, choose the **Yes** option for the **Enable boot mode for DWC\_ssi?** field using the **SPI Parameters** tab during the Specify Configuration Activity in coreConsultant.

## 2.16 Enhanced SPI Modes

DWC\_ssi supports the dual, quad, octal and dual octal modes of SPI using the `SSIC_SPI_MODE` configuration parameter. The possible values for this parameter are Standard, Dual SPI, Quad SPI, Octal SPI and Dual Octal SPI modes. When dual, quad, octal or dual octal mode is selected for this parameter, the width of `txd`, `rxd` and `ssi_oe_n` signals change to 2, 4, 8 or 16 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 39. Dual SPI, Quad, Octal or Dual 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 Enhanced SPI modes in detail:

### 2.16.1 Write Operation in Enhanced SPI Modes

Enhanced SPI write operations can be divided into three phases:

- Instruction phase
- Address phase
- Wait Cycle
- 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) - Specifies length of instruction, address, and data.
- `SPI_CTRLR0.INST_L` - Specifies length of an instruction (possible values for instruction length are 0, 4, 8, or 16 bits).
- `SPI_CTRLR0.ADDR_L` - Specifies address length.
- `CTRLR0.DFS` - 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`). DWC\_ssi waits until both have been programmed to start the write operation.



`SPI_CTRLR0.ADDR_L` and `SPI_CTRLR0.INST_L` can not be both programmed to 0 at the same time.

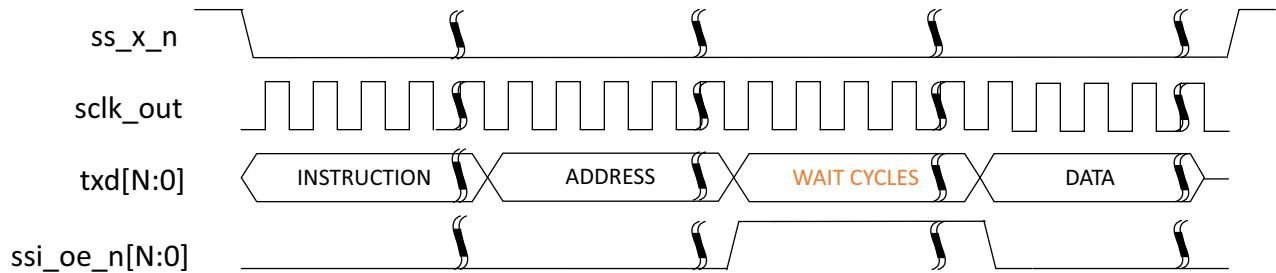
The instruction, address and data can be programmed to send in dual mode/quad mode/octal mode, which can be selected from the `SPI_CTRLR0.TRANS_TYPE` and `CTRLR0.SPI_FRF` fields.



- If `CTRLR0.SPI_FRF` is selected to be Standard SPI Format, all data 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.
- If you do not wish to insert wait cycles in the Write transfer, `SPI_CTRLR0.WAIT_CYCLES` field can be programmed to 00.

[Figure 2-60](#) shows a typical write operation in Dual, Quad, Octal or Dual Octal SPI Mode. The value of N is 7 if SSIC\_SPI\_MODE is set to Octal Mode, 3 if SSIC\_SPI\_MODE is set to Quad Mode, and 1 if SSIC\_SPI\_MODE is set to Dual Mode. 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 2-60 Typical Write Operation Enhanced SPI Mode**



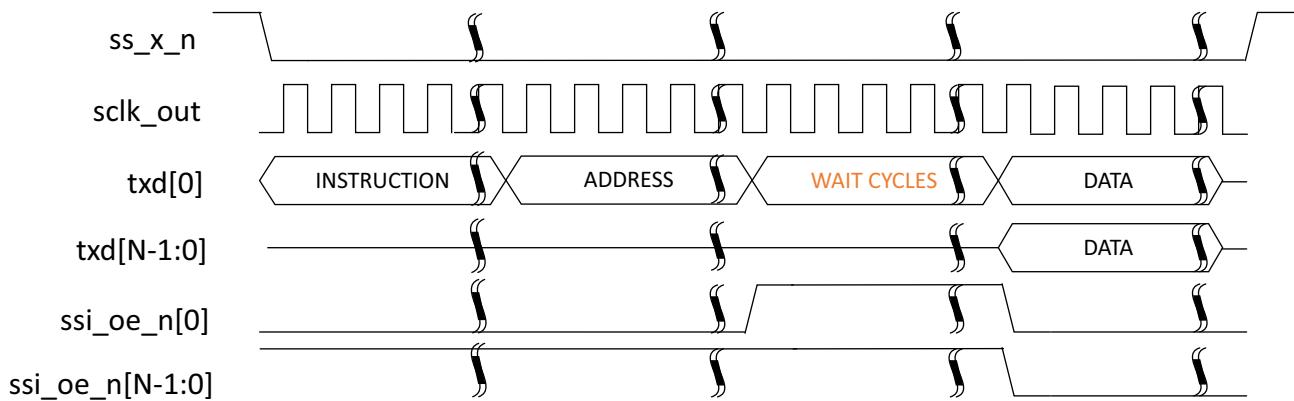
To initiate an Enhanced write operation, `CTRLR0.SPI_FRF` must be set to 01/10/11, respectively. This sets the transfer type, and for each write command, data is transferred in the format specified in `CTRLR0.SPI_FRF` field.

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 2-61](#) shows 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 2-61 Instruction and Address Transmitted in Standard SPI Format**

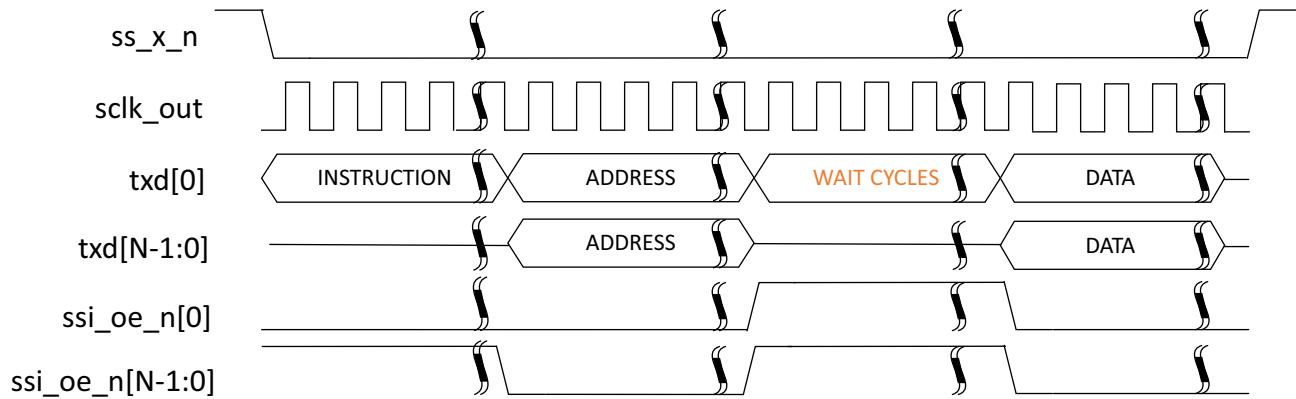


#### Case B: Instruction transmitted in standard and address transmitted in Enhanced SPI format

For this, `SPI_CTRLR0.TRANS_TYPE` field must be set to 01. [Figure 2-62](#) shows the timing diagram when an instruction is transmitted in standard format and address is transmitted in SPI format specified in

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.

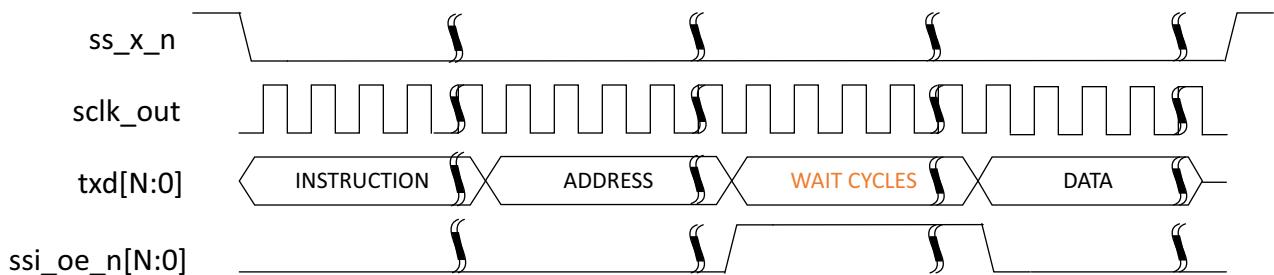
**Figure 2-62 Instruction Transmitted in Standard and Address Transmitted in Enhanced SPI Format**



#### Case C: Instruction and Address both transmitted in Enhanced SPI format

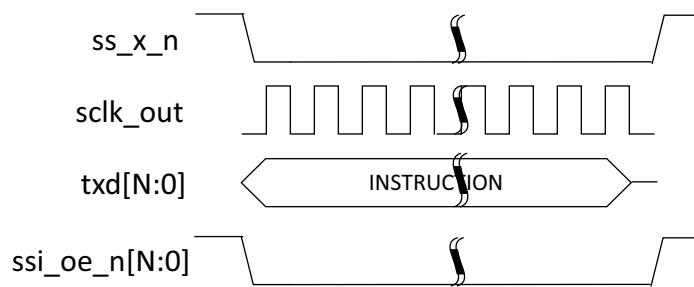
For this, SPI\_CTRLR0.TRANS\_TYPE field must be set to 10. [Figure 2-63](#) shows the timing diagram in which instruction and address are transmitted in SPI format specified in the CTRLR0.SPI\_FRF field. The value of N is: 7 if CTRLR0.SPI\_FRF is set to 11, 3 if CTRLR0.SPI\_FRF is set to 10, and 1 if CTRLR0.SPI\_FRF is set to 01.

**Figure 2-63 Instruction and Address Both Transmitted in Enhanced SPI Format**



#### Case D: Instruction only transfer in Enhanced SPI format

In this scenario, SPI\_CTRLR0.TRANS\_TYPE field must be set to 10. [Figure 2-64](#) 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 2-64 Instruction only Transfer in Enhanced SPI Format**

### Note

- Some of the devices require instruction and address only transfer (without data phase). This can be achieved by skipping the address phase, and transmitting the address as data to the device. For this, `SPI_CTRLR0.ADDR_L` should be programmed to 0, and `CTRLR0.DFS` should be programmed with the required length.
- Some of the devices require data only write transfer (without instruction and address phase). This can be achieved by skipping the instruction phase, and transmitting the first data as address and then continue with the data transfer to the device. For this, `SPI_CTRLR0.INST_L` needs to be programmed to 0 and `SPI_CTRLR0.ADDR_L` and `CTRLR0.DFS` needs to be programmed with the required frame size for the data.

## Case E: Dual Octal Write Operation

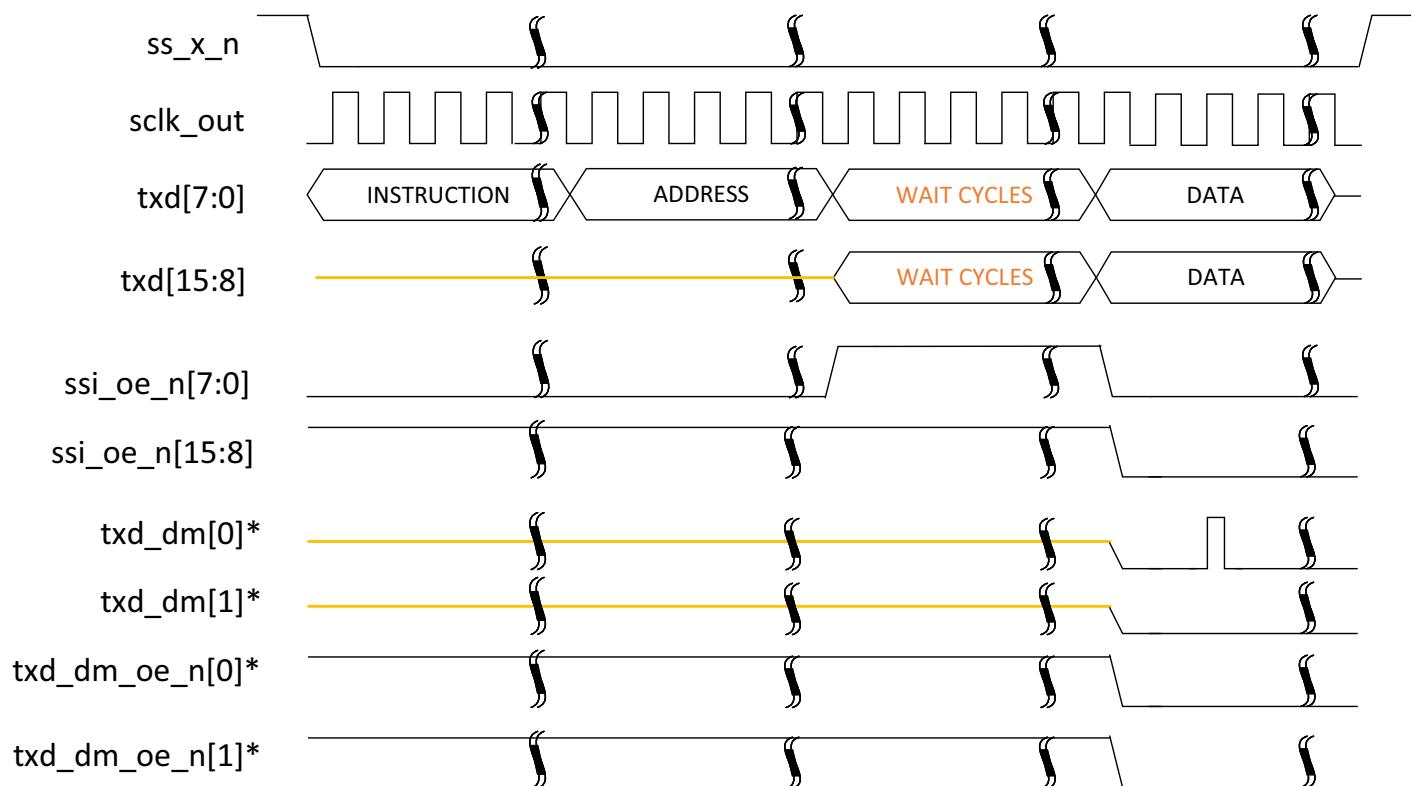
Set the following register fields for a dual-octal write operation:

- `CTRLR0.SPI_FRF`: Set to octal format (2'b11)
- `SPI_CTRLR0.INST_L`: Set as multiples of 8
- `SPI_CTRLR0.ADDR_L`: Set as multiples of 8
- Wait cycles can be programmed using `SPI_CTRLR0.WAIT_CYCLES` field
- `CTRLR0.DFS`: Set as multiples of 16
- `SPI_CTRLR0.TRANS_TYPE` : Set to 2'b11, which enables the dual-octal operation

The Dual octal protocol can be enabled only when the IP is programmed as specified earlier. The address and instruction are transmitted on the TXD [7:0] and the data phase is transmitted on all 16 lines (TXD[15:0]). The Dual Octal transfer can also be performed in the XIP mode with the following restriction:

- When `SPI_CTRLR0.DDR_EN` is set to 0, then the `HSIZE` value can only be 16/32.
- When `SPI_CTRLR0.DDR_EN` is set to 1, then `HSIZE` value can only be 32.
- If hardcoded DFS is enabled, then the hardcoded DFS value can be 16/32.
- If `SPI_CTRLR0.DDR_EN` is set to 0 and DFS value can only be 32 when `SPI_CTRLR0.DDR_EN` is set to 1.

Figure 2-65 shows an example SPI write transfer in Dual-Octal mode.

**Figure 2-65 Dual Octal Write Transfer**

\* Signals are only applicable for the DDR transfers

### 2.16.2 Read Operation in Enhanced SPI Modes

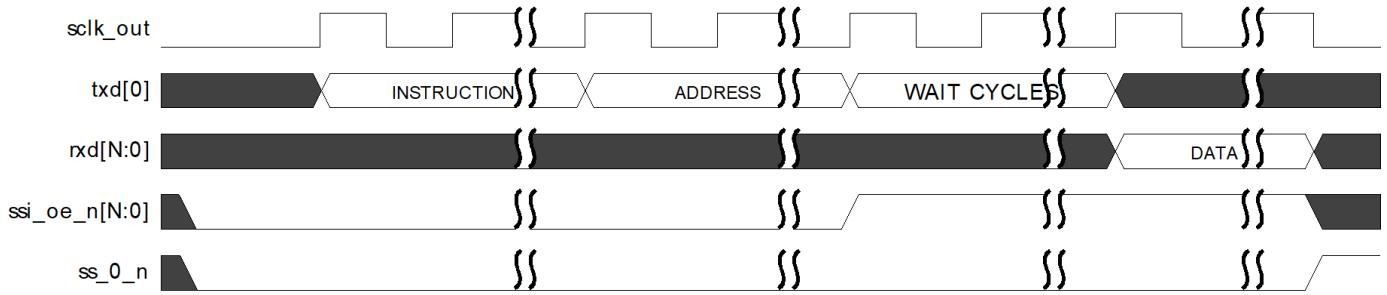
A Dual, Quad, Octal or Dual Octal SPI read operation can be divided into four phases:

- Instruction phase
- Address phase
- Wait cycles
- Data phase

Wait Cycles can be programmed using `SPIC_CTRLR0.WAIT_CYCLES` field. The wait cycles are introduced for target to change their mode from input to output and the wait cycles can vary for different devices.

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

Figure 2-66 shows a typical read operation in dual quad SPI mode. The value of N is: 7 if SSIC\_SPI\_MODE parameter is set to Octal mode, 3 if SSIC\_SPI\_MODE parameter is set to Quad mode, and 1 if SSIC\_SPI\_MODE parameter is set to Dual mode.

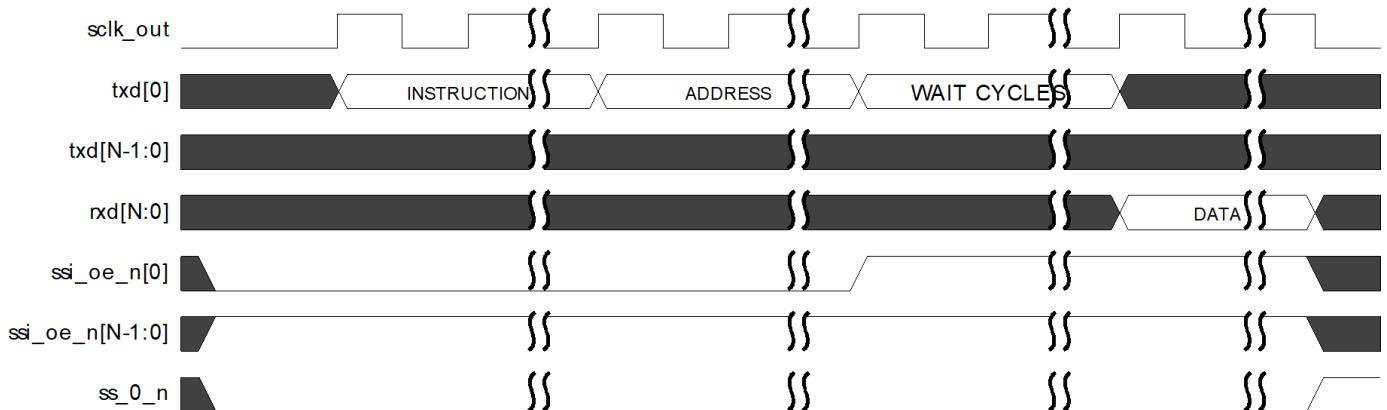
**Figure 2-66 Typical Read Operation in Enhanced SPI Mode**

To initiate an Enhanced 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 `CTRLR0.SPI_FRF` field.

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

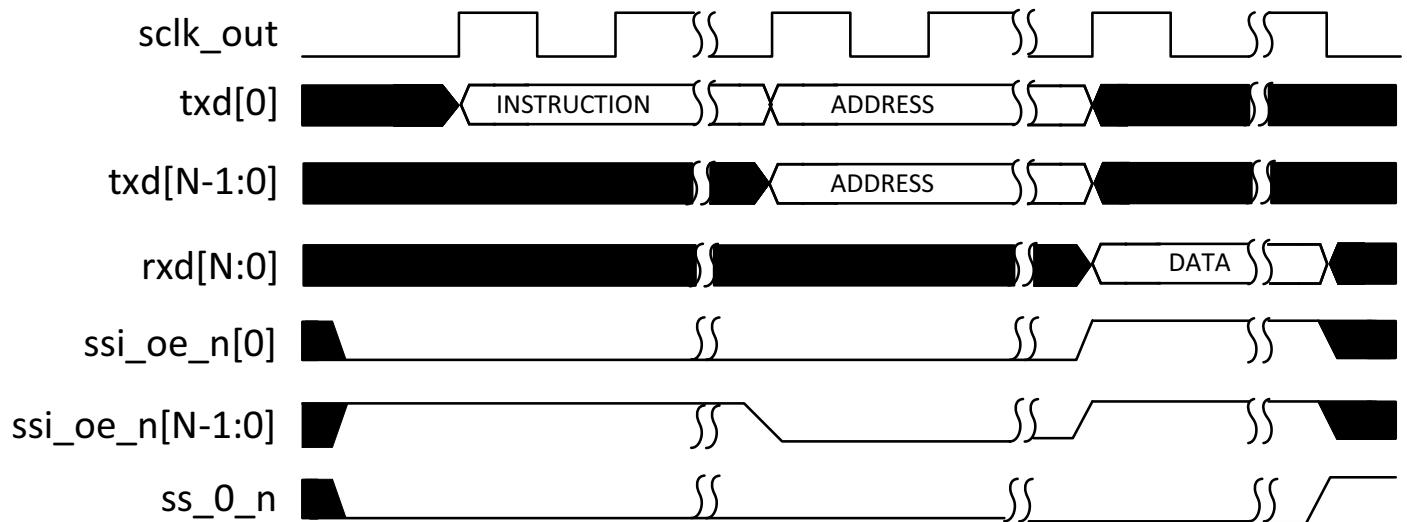
- Case A: Instruction and address both transmitted in standard SPI format

For this, `SPI_CTRLR0.TRANS_TYPE` field should be set to 00. [Figure 2-67](#) 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 2-67 Instruction and Address Transmitted in Standard SPI Format**

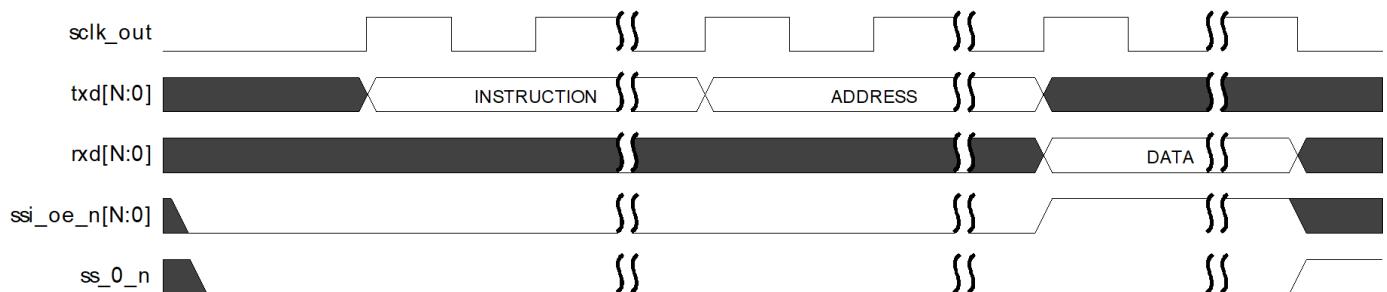
- Case B: Instruction transmitted in standard and address transmitted in Enhanced SPI format

For this, `SPI_CTRLR0.TRANS_TYPE` field should be set to 01. [Figure 2-68](#) shows the timing diagram in which instruction is transmitted in standard format and address is transmitted in Enhanced 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 2-68 Instruction Transmitted in Standard and Address Transmitted in Enhanced SPI Format**

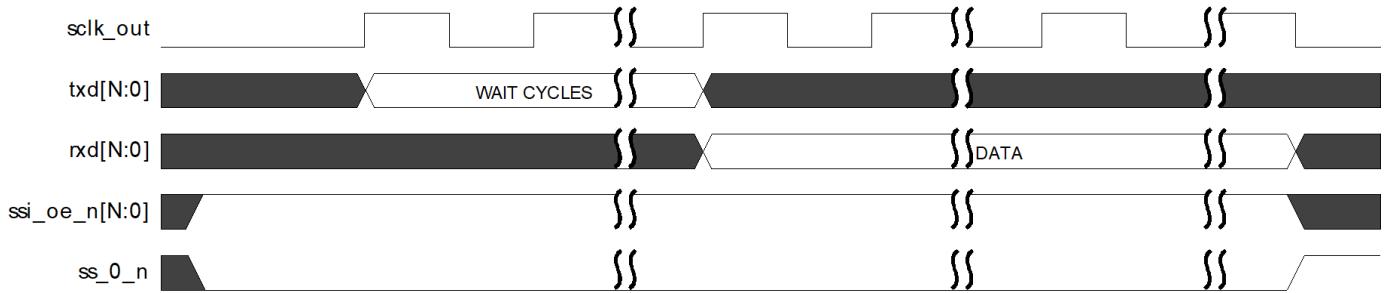
- Case C: Instruction and Address both transmitted in Enhanced SPI format

For this, `SPI_CTRLR0.TRANS_TYPE` field must be set to 10. [Figure 2-69](#) shows the timing diagram in which both instruction and address are transmitted in Enhanced 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 2-69 Instruction and Address Transmitted in Enhanced SPI Format**

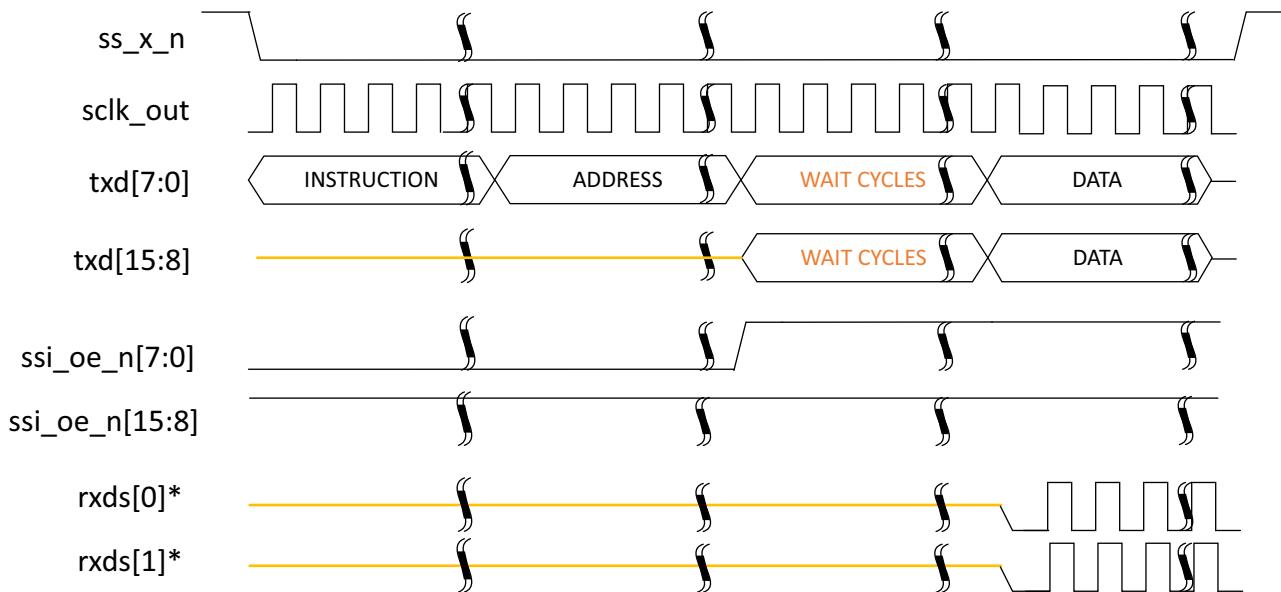
- Case D: No Instruction, No Address READ transfer

For this, `SPI_CTRLR0.ADDR_L` and `SPI_CTRLR0.INST_L` must be set to 0. [Figure 2-70](#) 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 temporary write in the data register (DR), DWC\_ssi waits for programmed wait cycles and then fetch the amount of data specified in NDF field.

**Figure 2-70 No Instruction and No Address READ Transfer**

- Case E: Dual Octal Read Operation

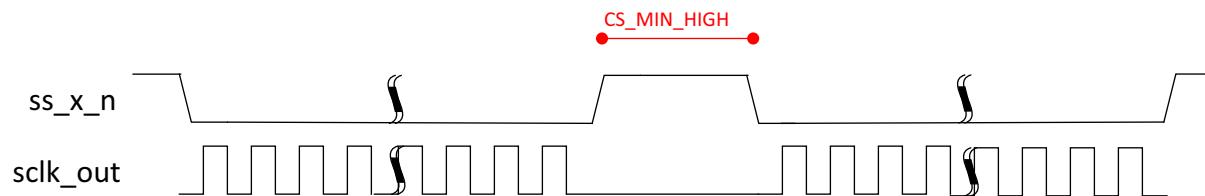
The read operation works in the same manner as the enhanced SPI protocol. The programming guidelines are described in “[Write Operation in Enhanced SPI Modes](#)” on page 102. Similar to the write operation, there are two data strobes for the read data one data strobe for rxd[7:0] and rxd[15:8].

**Figure 2-71 Dual Octal SPI Read Transfer**

\* Signals are only applicable for the DDR transfers

### 2.16.3 Variable Chip-select HIGH Period between back-to-back Transfers

DWC\_ssi supports a variable minimum delay between two back-to-back SPI transfers. You can set the `ssi_clk` period number of this delay in the `SPI_CTRLR1.CS_MIN_HIGH` register field.

**Figure 2-72 Delay for CS\_MIN\_HIGH**

**Note** The delay set in the `SPI_CTRLR1.CS_MIN_HIGH` field establishes the minimum delay between two back-to-back SPI transfers, the actual delay can be longer due to operational delays.

#### 2.16.4 Enabling the Required SPI Mode

To specify the SPI mode in which DWC\_ssi must work, select the required option in the **Select Controller SPI mode** field using the **SPI Parameter** tab during the Specify Configuration Activity in coreConsultant. The default mode is the Standard mode. You can change the mode to Dual, Quad, or Octal only if DWC\_ssi is configured as a Serial Controller.

This field provides the following options:

- Standard Mode: Configures DWC\_ssi in Standard Mode.
- Dual Mode: Configures DWC\_ssi in Dual Mode in which the width of the `txd` and `rxn` signals are 2-bits.
- Quad Mode: Configures DWC\_ssi in Quad Mode in which the width of `txd` and `rxn` signals are 4-bits.
- Octal Mode: Configures DWC\_ssi in Octal Mode in which the width of `txd` and `rxn` signals are 8-bits.
- Dual Octal Mode: Configures DWC\_ssi in Dual Octal Mode in which the width of `txd` and `rxn` signals are 16-bits.

#### 2.16.5 Registers Related to Enhanced SPI Modes

The following are the registers related to Enhanced SPI Modes:

- `CTRLR0`
- `SPI_CTRLR0`

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.17 Enhanced SPI Operation in Target Mode

DWC\_ssi can be configured to work in enhanced SPI mode in target configuration using `SSIC_SPI_MODE` 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 `SSIC_SPI_MODE` parameter, the signal width of `txd`, `rxd` and `ssi_oe_n` changes to 2, 4, and 8 respectively. Hence, the data is shifted in or out on more than one line, increasing the overall throughput. The mode of operation (write or read) can be selected using the `CTRLR0.TMOD` field.



**Note** Minimum clock ratio between `ssi_clk` and `sclk_in` should be 1:6 during enhanced SPI target operation.

The following sections describe the read and write operations in Enhanced SPI mode:

### 2.17.1 Write Operation

If `TMOD = 2'b01`, then the receive data is invalid and should not be stored in the receive FIFO. The data transfer occurs based on the selected frame format (`CTRLR0.SPI_FRF` field) as usual. Transmit data is fetched from the transmit FIFO, and sent through the `txd` line to the target device.

**Figure 2-73 Write Operation in Enhanced SPI Mode**

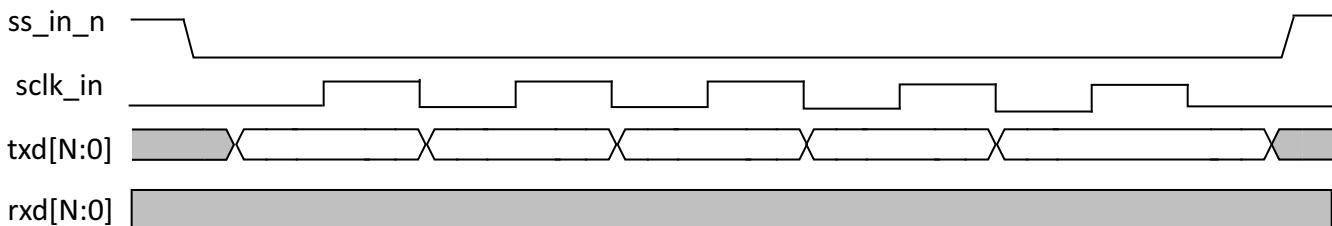


Figure 2-73 shows write operation in enhanced SPI mode.

- N = 7 if `CTRLR0.SPI_FRF` is set to 11
- N = 3 if `CTRLR0.SPI_FRF` is set to 10
- N = 1 if `CTRLR0.SPI_FRF` is set to 01

DWC\_ssi does not expect to read any data from SPI controller, unlike Enhanced memory devices, which expects Instruction and Address phase to read the data from them. Data transmission starts from the first clock edge itself. This improves the overall performance of the system.

### 2.17.2 Read Operation

When `TMOD = 2'b10`, the transmit data is invalid. When configured as a target, 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 per the selected frame format (`CTRLR0.SPI_FRF` field). 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.

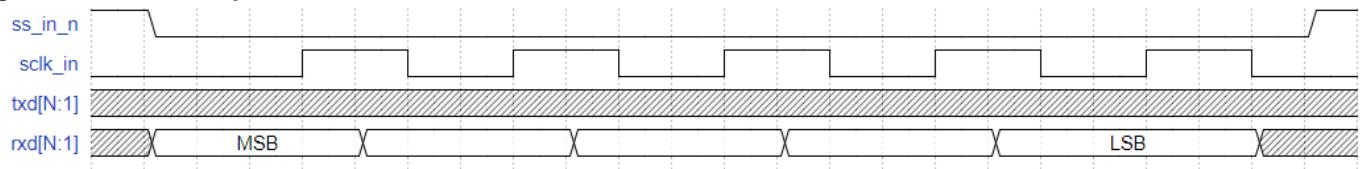
**Figure 2-74 Read Operation in Enhanced SPI Mode**

Figure 2-74 shows read operation in enhanced SPI mode.

- N = 7 if CTRLR0 .SPI\_FRF is set to 11
- N = 3 if CTRLR0 .SPI\_FRF is set to 10
- N = 1 if CTRLR0 .SPI\_FRF is set to 01

## 2.18 Clock Stretching in Enhanced SPI Transfers

DWC\_ssi includes clock stretching feature in enhanced SPI modes which can be used to prevent FIFO underflow and overflow conditions while transmitting or receiving the data respectively. If TX FIFO becomes empty before the transfer is complete, then DWC\_ssi serial controller masks the `sclk_out` and then resumes the clock when TX FIFO becomes non-empty. In case of receive transaction, when RX FIFO becomes full, then DWC\_ssi masks `sclk_out` until the data is read from receive FIFO (the data level goes below the RX threshold programmed in the RXFTLR register).

DWC\_ssi provides a programming bit (`SPI_CLOCK_STRETCH_EN`) in `SPI_CTRLR0` register to enable clock stretching feature.



**Note** For clock stretching applications, the DWC\_ssi masks `sclk_out` until the data is read from RX FIFO and the data level goes below the programmed RX threshold in the RXFTLR register. The value of the RXFTLR has to be 1 or more for clock stretching, as usage of the value of 0 is invalid for this clock stretching scenario.

### 2.18.1 Description of Clock Stretching in Enhanced SPI Transfers

While DWC\_ssi is transmitting or receiving the data from SPI device, the software may not be able to keep up with the transfer rate due to the bandwidth issues on the serial target interface. In such cases, TX FIFO becomes empty or RX FIFO overflows while transmitting and receiving the data.

To avoid data corruption, CPU must discard the current operation and start a new transfer. This process is time consuming and inefficient.

To handle such scenarios, clock stretching support has been added in DWC\_ssi.

- For write transactions, whenever transmit TX FIFO becomes empty, DWC\_ssi does not de-select the serial target. Instead, it masks the clock until the new data is pushed into the TX FIFO. Hence, the transfer is not broken, and CPU intervention is not required. The transfer is resumed after the transmit FIFO receives at least one FIFO entry.
- For read transactions, similar to the write transaction, whenever DWC\_ssi detects that RX FIFO is full, the clock is masked until the data is read from FIFO. The transfer is resumed when FIFO level goes below the set threshold level in the RXFTLR register.

If clock stretching feature is enabled for all the write transactions, you should program the number of data frames in the `CTRLR1` register. The clock stretching only happens in the data phase of enhanced SPI transfers. If you only program instruction and address in the data register, DWC\_ssi does not end the transfer after instruction and address are transmitted. To make sure that data phase is reached, ensure that at least one data (apart from instruction and address) is present in transmit FIFO. Set the minimum threshold value in the `TXFTRLR.TXFTHR` register to achieve that.

For Read transfers, If `RX_SAMPLE_DELAY` register is enabled, DWC\_ssi estimates the time when RX FIFO may become full and masks `sclk_out` accordingly. Clock stretching starts early in read operations to prevent FIFO overflow in the following conditions:

- When Baud rate != 2
  - If `SSIC_HAS_EXT_RAM` is set to 1, clock stretching starts when FIFO level reaches `SSIC_RX_FIFO_DEPTH-1`.
- When Baud Rate = 2

- ❑ If SSIC\_HAS\_EXT\_RAM is set to 1, clock stretching starts when FIFO level reaches to SSIC\_RX\_FIFO\_DEPTH-2.
- ❑ If SSIC\_HAS\_EXT\_RAM is set to 0, clock stretching starts when FIFO level reaches to SSIC\_RX\_FIFO\_DEPTH-1.

For the optimum operation, it is recommended to choose the Receive FIFO threshold level below than the clock stretching start values.



**Note** Clock stretching feature is applicable only for enhanced SPI transfers except XIP transfers.

## 2.18.2 Enabling the Clock Stretching in Enhanced SPI Transfers

To configure the clock stretching for DWC\_ssi, choose the **Yes** option for the **Enable Clock Stretching in Enhanced SPI Mode?** fields using the **Enhanced SPI Parameters** tab during the Specify Configuration Activity in coreConsultant.

## 2.18.3 Registers Related to Clock Stretching in Enhanced SPI Transfers

The following are the registers related to Clock Stretching in Enhanced SPI Transfers:

- SPI\_CTRLR0
- TXFTLR
- CTRLR1
- RXFTLR

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.19 SPI Dynamic Wait States Feature

Legacy SPI devices do not allow any flow control mechanism on the SPI interface. The SPI Dynamic wait state feature defines a protocol under which SPI target can control the data flow during the transfer. This feature allows SPI target to insert wait states dynamically after the transaction is requested by SPI controller.

### 2.19.1 Dynamic Wait States in Legacy SPI Mode

In the normal mode of operation in SPI, for Read operation:

- the command and address are driven by the SPI controller on TXD line
- the serial target responds back with the data on first driving edge of after address phase

With Dynamic Wait State feature, SPI target can communicate the data availability using the rxd pin. The status is sampled along with the last address bit.

- If the status indicates that the data is available, then the operation continues as in the normal SPI mode.
- If the status is set to 0, then DWC\_ssi serial controller introduces wait states programmed in the DYN\_WS field of the SPI\_CTRLR1 register.



**Note** Dynamic wait state feature can only be enabled when SSIC\_SPI\_MODE !=0.

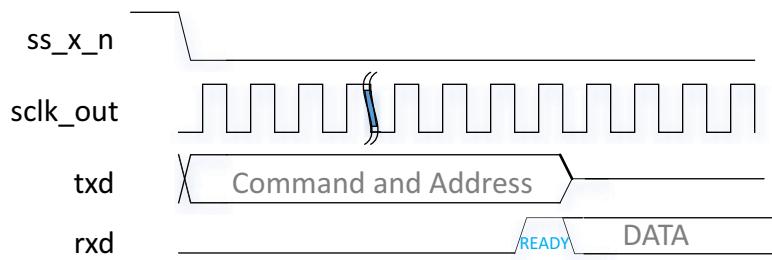
The status is checked again when the transfer is resumed after the wait states are over. This is illustrated in [Figure 2-75](#).

You can configure the number of times a serial target can provide a NON-READY response, which can be programmed in the MAX\_WS field of the SPI\_CTRLR1 register.

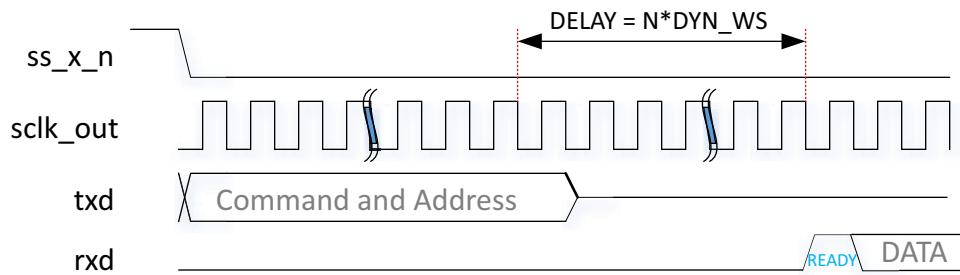
For example, if MAX\_WS field of the SPI\_CTRLR1 register field is set to 2 then serial target can provide a NON\_READY response for maximum two times. The maximum programmable value of MAX\_WS field of the SPI\_CTRLR1 register is 15, so the maximum wait cycles can be  $15 * (\text{SPI\_CTRLR1}.\text{DYN\_WS})$ . If the serial controller fails to get a READY status from the serial target within this time, then it stops the SPI transfer and the FIFO is flushed (in case of write operation). This is indicated by SPI transmit error interrupt `ssi_spi_te(_n)`.

If MAX\_WS field of the SPI\_CTRLR1 register is set to 0, then DWC\_ssi waits until the serial target responds with the READY status. In this case, the transaction can end only when the serial target provides a READY response; or when DWC\_ssi is disabled by writing into the SSIENR register.

The software must take care of the deadlock situation by implementing appropriate mitigation mechanism.

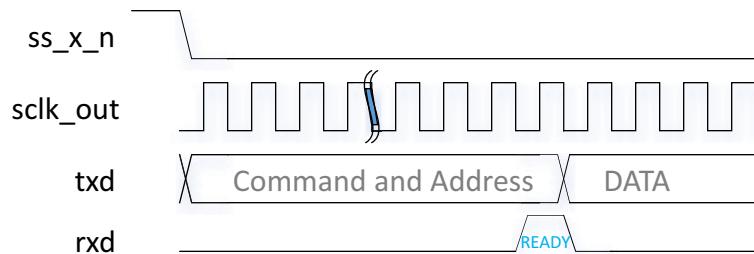
**Figure 2-75 SPI Read Transaction With 0 Dynamic Wait States**

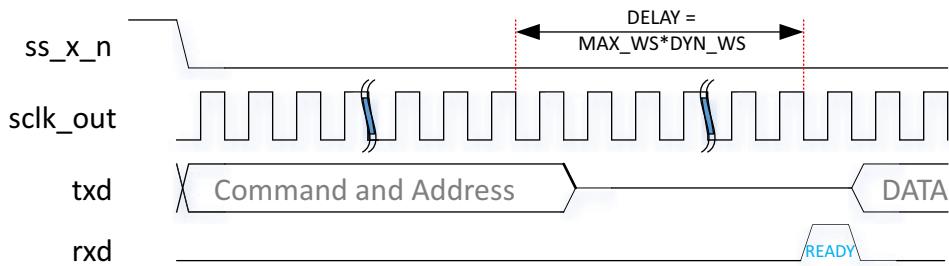
[Figure 2-76](#) shows the example for a read operation with dynamic wait states. To perform a read operation, set the TMOD field of the CTRLR0 register to 2.

**Figure 2-76 SPI Read Transaction With Dynamic Wait States**

$N$  = value of the MAX\_WS field of the SPI\_CTRLR1 register. (if the MAX\_WS field of the SPI\_CTRLR1 register is not set to 0).

Similar operation use case applies for the write operation as well. The wait is indicated by the `rxd` pin and wait states are inserted if the serial target is not ready to accept the data from the serial controller. To perform a write operation, set the TMOD field of the CTRLR0 register to 1.

**Figure 2-77 SPI Write Transaction With 0 Wait State**

**Figure 2-78 SPI Write Transaction With Dynamic Wait States**

When the SPI\_DWS\_EN field of the CTRLR0 register is set to 1, then command and address length can be set using the SPI\_CTRLR0.INST\_L and SPI\_CTRLR0.ADDR\_L register fields respectively. DWC\_ssi uses these values to send command and instruction to in standard SPI format, WAIT\_CYCLES field of the SPI\_CTRLR0 register is ignored in this mode. The delay can be inserted only by serial target device using rxd (MISO) pin.

### 2.19.2 Dynamic Wait States in Enhanced SPI Mode

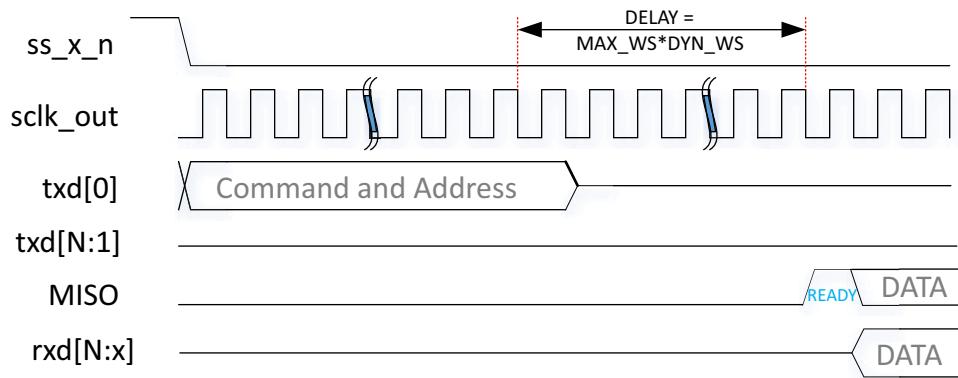
In Enhanced SPI mode (Dual/Quad/Octal/Dual Octal), data pins are bi-directional. The wait status indication is given on the rxd (MISO) pin in standard SPI operation. The read and write operations both can have wait cycles after completion of address phase (defined in the WAIT\_CYCLES field of the SPI\_CTRLR0 register). After the wait cycles are complete, DWC\_ssi Controller checks for the READY status in rxd[1] pin, and then resumes the transfer.



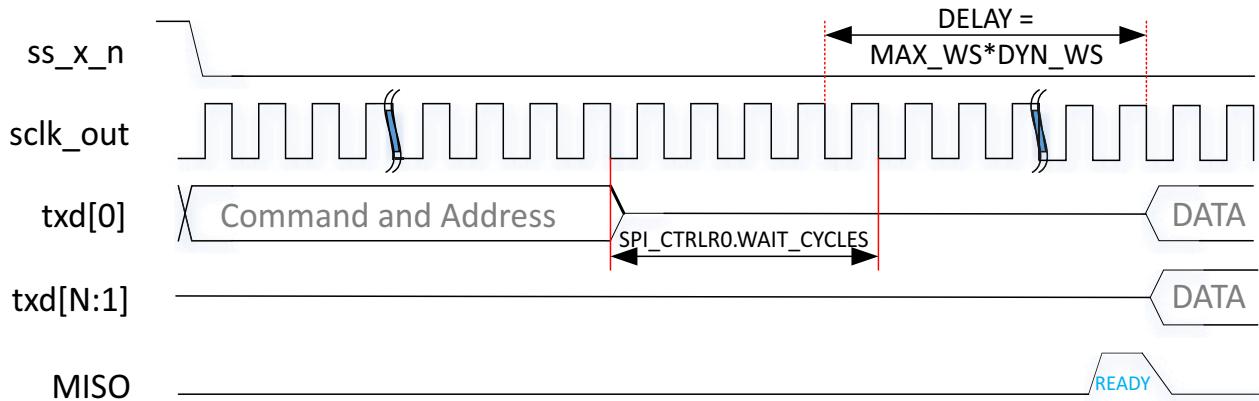
- When address is transmitted on multiple lanes, the WAIT\_CYCLES field of the SPI\_CTRLR0 register cannot be set to 0 because the rxd (MISO) pin cannot be sampled during the last cycle of the address phase.
- Dynamic Wait States in Enhanced SPI Mode feature is not applicable for HyperBus transfers and XIP transfers.
- For write operation the wait cycles (SPI\_CTRLR0.WAIT\_CYCLES) are applicable only when address is transmitted on multiple lanes (SPI\_CTRLR0.TRANS\_TYPE > 0).

From [Figure 2-79](#) to [Figure 2-82](#) to explain dynamic wait state behavior in enhanced SPI operation. In the following figures N= 1 for Dual mode; 3 for Quad mode and 7 for Octal mode of operation.

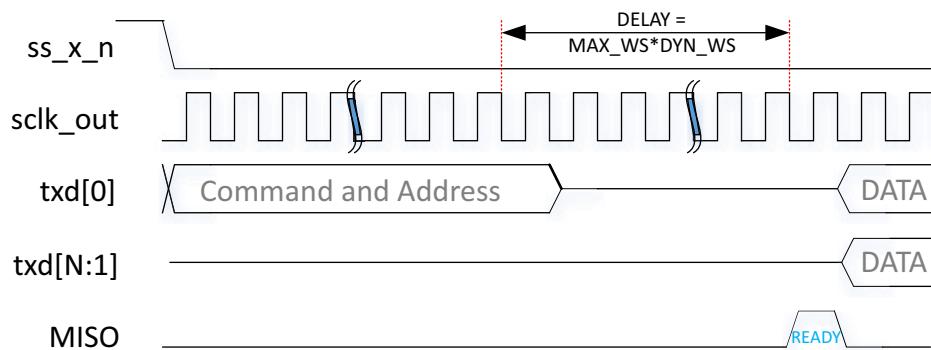
**Figure 2-79 Read Operation When Address is Transmitted on Single Lane with SPI\_CTRLR0.WAIT\_CYCLES=0**

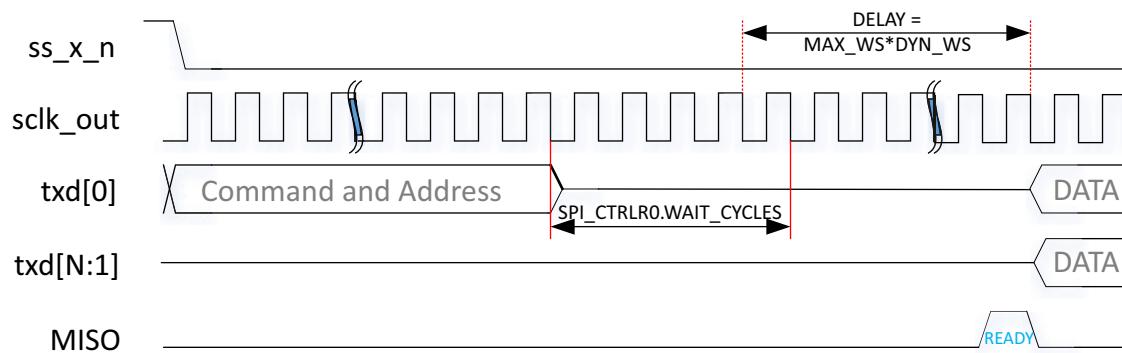


**Figure 2-80 Read Operation When Address is Transmitted on Multiple Lane**



**Figure 2-81 Write Operation When Address is Transmitted on Single Lane**



**Figure 2-82 Write Operation When Address is Transmitted on Multiple Lane**

### 2.19.3 Enabling the Dynamic Wait States Feature

To enable the Dynamic Wait State feature for SPI in DWC\_ssi, choose the **Yes** option for the **Enable Dynamic Wait State feature for SPI?** field using the **SPI Parameters** tab during the Specify Configuration Activity in coreConsultant.

### 2.19.4 Registers Related to Dynamic Wait States Feature

The following are the registers related to the Dynamic Wait States feature:

- **CTRLR0**
- **SPI\_CTRLR1**

For more information about these registers, see the Registers Descriptions chapter.

## 2.20 Dual Data-Rate (DDR) Support

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 the SPI protocol:

- Mode 0 – When the default serial clock phase and default serial clock polarity are not enabled ( $\text{SCPH} = 0 \ \& \ \text{SCPOL} = 0$ )
- Mode 3 – When the default serial clock phase and default serial clock polarity are enabled ( $\text{SCPH} = 1 \ \& \ \text{SCPOL} = 1$ )

### 2.20.1 Description of Dual Data-Rate (DDR) Support

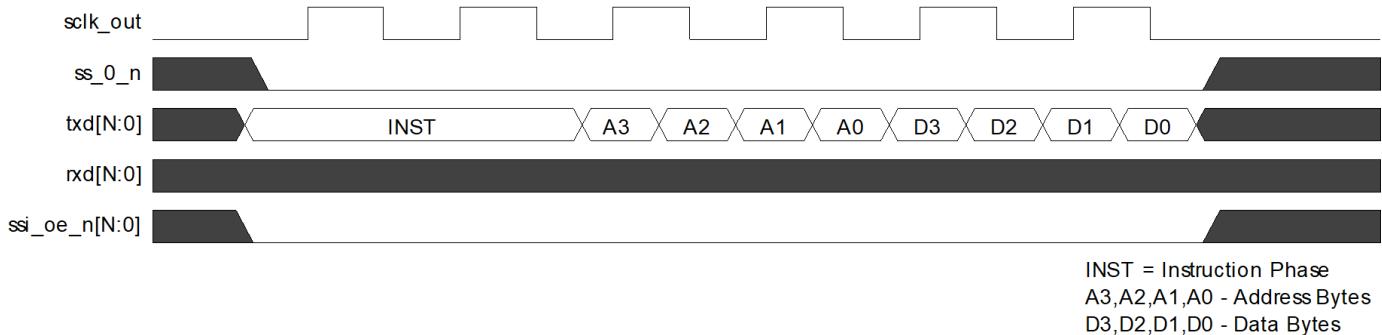
DDR commands enable data to be transferred on both edges of clock. Following are the different types of DDR commands:

- 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.

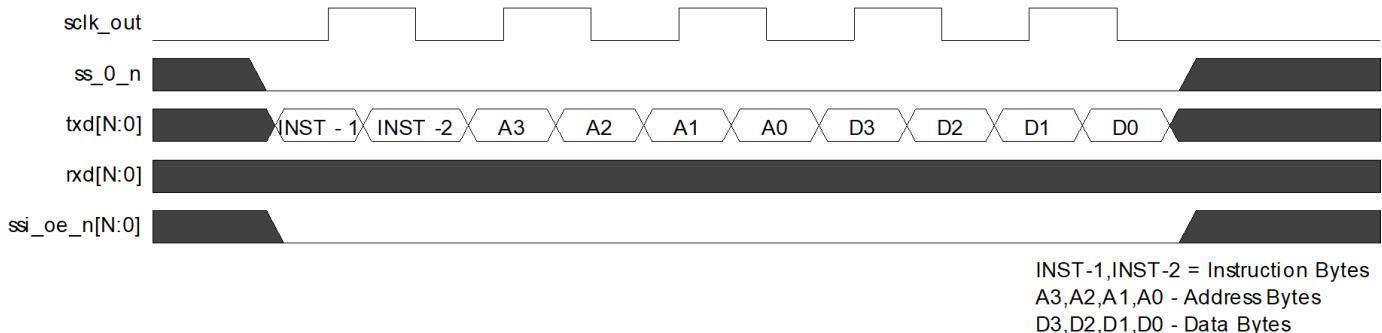
The `SPI_DDR_EN` (`SPI_CTRLR0[16]`) bit is used to determine if the Address and data have to be transferred in DDR mode and `INST_DDR_EN` (`SPI_CTRLR0[17]`) bit is used to determine if Instruction must be transferred in DDR format. These bits are only valid when the `CTRLR0.SPI_FRF` bit is set to be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

[Figure 2-83](#) describes a DDR write transfer where instructions are continued to be transmitted in standard format. In [Figure 2-83](#), 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 2-83 DDR Transfer with  $\text{SCPH} = 0$  and  $\text{SCPOL} = 0$**



[Figure 2-84](#) describes a DDR write transfer where instruction, address and data all are transferred in DDR format.

**Figure 2-84 DDR Transfer with Instruction, Address and Data Transmitted in DDR Format**

In the DDR transfer, address and instruction cannot be programmed to a value of 0. The number of wait cycles cannot be 0 in a DDR read operation.

## 2.20.2 Transmitting Data in DDR Mode

In DDR mode, data is transmitted on both edges so that it is difficult to sample data correctly. DWC\_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. DWC\_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** register is equal to  $\lceil (\text{BAUDR} / 2) - 1 \rceil$ . Data is driven before  $(\text{BAUDR} / 2) - \text{TXD\_DRIVE\_EDGE}$  **ssi\_clk** before the sampling edge. 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** register is set to 1 then the data is transmitted two **ssi\_clk** before the edge of **sclk\_out**.

[Figure 2-85](#), [Figure 2-86](#), and [Figure 2-87](#) 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 6. In [Figure 2-85](#), transmit edge and driving edge of the data are the same. This is default behavior in DDR mode.

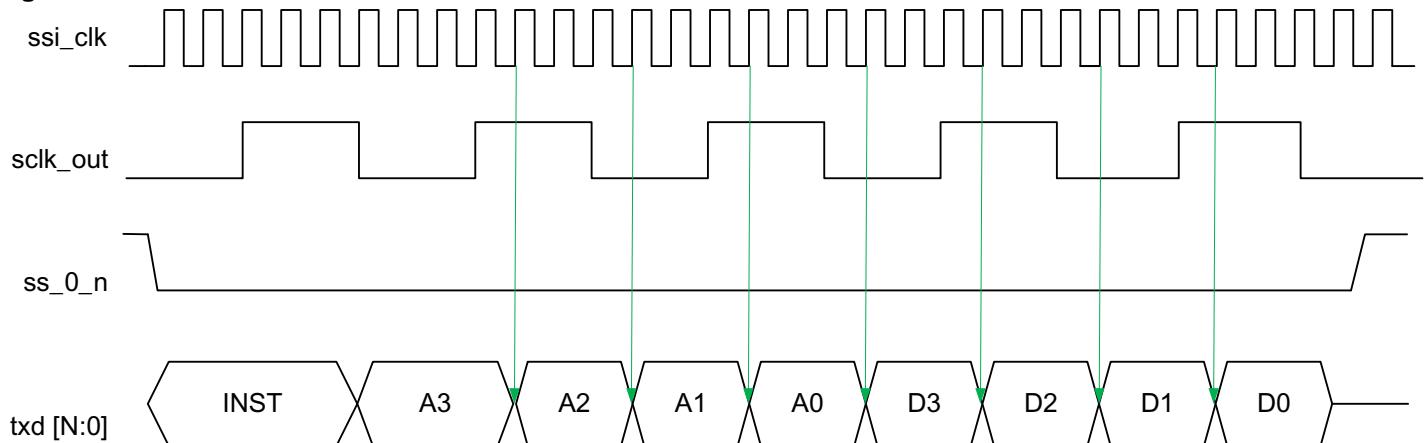
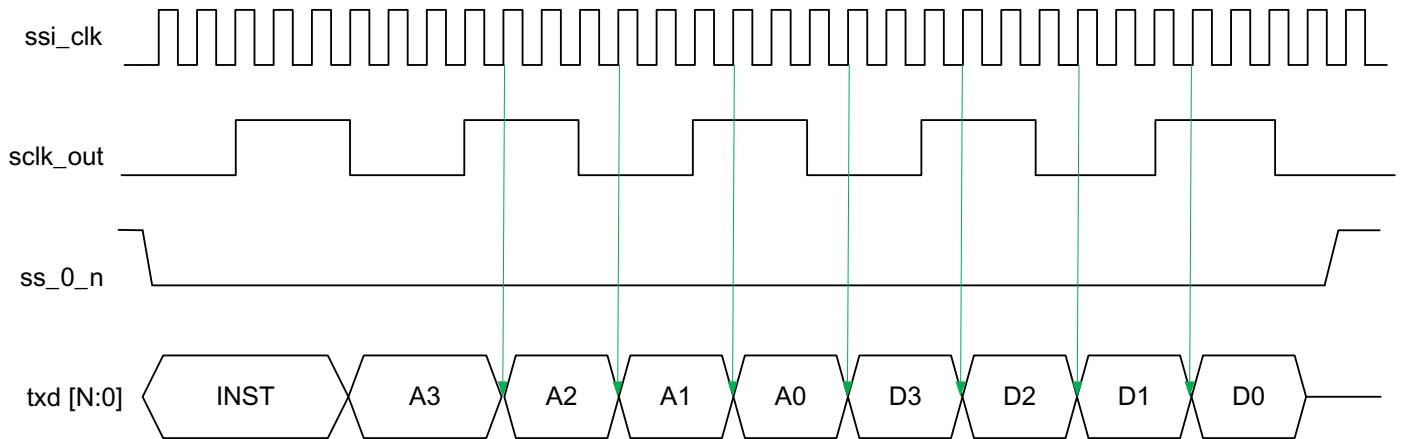
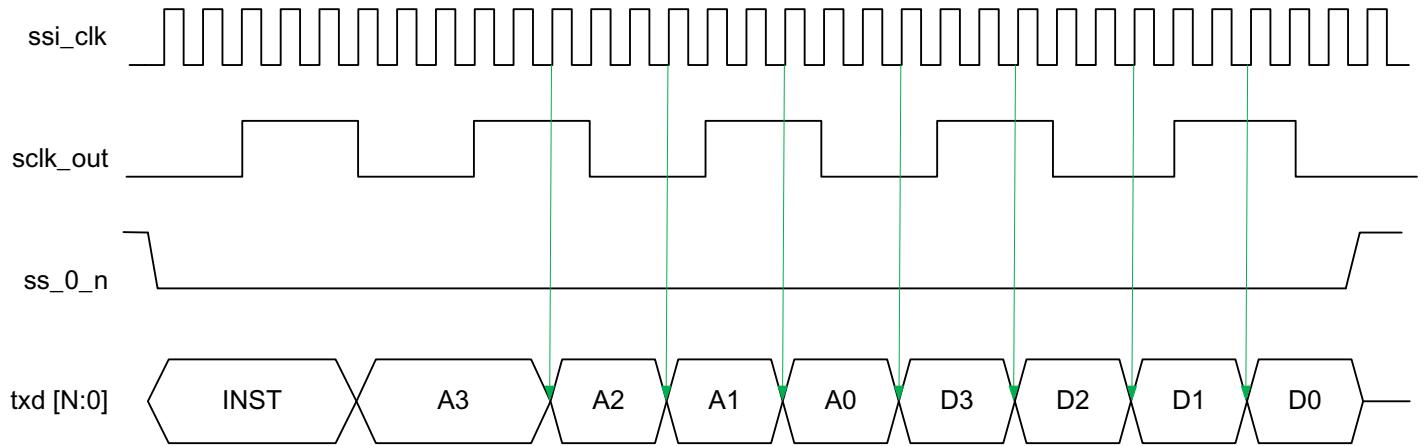
**Figure 2-85 Transmit Data With DDR\_DRIVE\_EDGE = 0**

Figure 2-85 shows the default behavior in which the transmit and driving edge of the data is the same.

**Figure 2-86 Transmit Data With DDR\_DRIVE\_EDGE = 1**



**Figure 2-87 Transmit Data With DDR\_DRIVE\_EDGE = 2**



### 2.20.3 Enabling DDR Transfer in SPI Frame Format

To enable DDR data transfer in SPI frame format in DWC\_ssi, choose the **Yes** option for the **Include DDR transfers in SPI frame format?** field using the **SPI Parameters** tab during the Specify Configuration Activity in coreConsultant. You can enable this feature only when DWC\_ssi is configured as a serial controller.



Baud rate value in DDR mode should be multiple of 4.

### 2.20.4 Registers Related to Dual Data-Rate (DDR)

The following are the registers related to Dual Data-Rate (DDR):

- CTRLR0
- SPI\_CTRLR0
- DDR\_DRIVE\_EDGE

- BAUDR

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.21 Read Data Strobe Signal Support

In order to achieve higher frequencies, data strobe signaling is used in read operations. The device gives a data strobe (known as read data strobe- rxds) signal along with read data to determine the output data valid window.

DWC\_ssi provides a configurable parameter SSIC\_HAS\_RXDS to include data strobe signal in the design. After setting the SSIC\_HAS\_RXDS parameter to 1, program the SPI\_CTRLR0.SPI\_RXDS\_EN register to 1 in order to sample incoming data for read data strobe signal as shown in [Figure 2-88](#). If you do not set the SPI\_CTRLR0.SPI\_RXDS\_EN signal to 1 the data is sampled based on existing logic.

It is expected that the device uses the incoming clock, and feeds it back to the controller as data strobe. DWC\_ssi transmits sufficient amount of clock cycles to complete the transfer. If the number of data frames expected is 4 the controller transmits 4 clocks only.

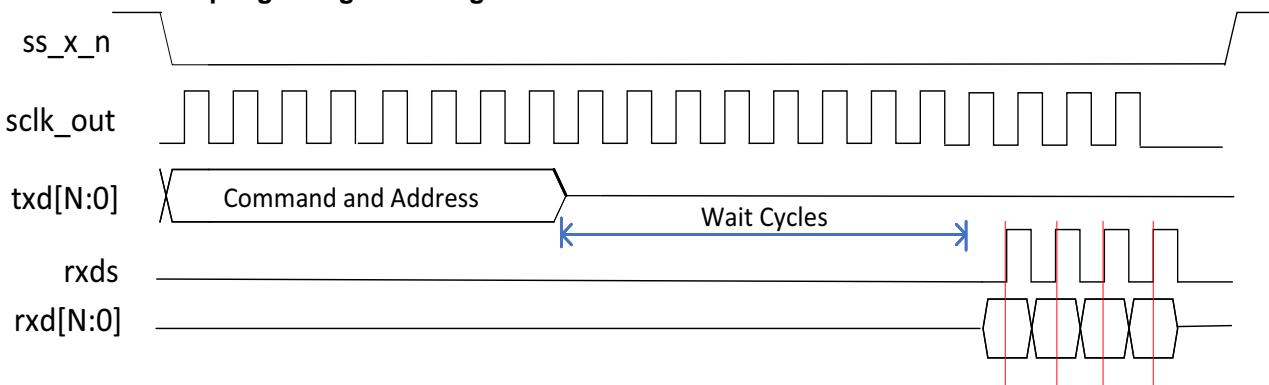


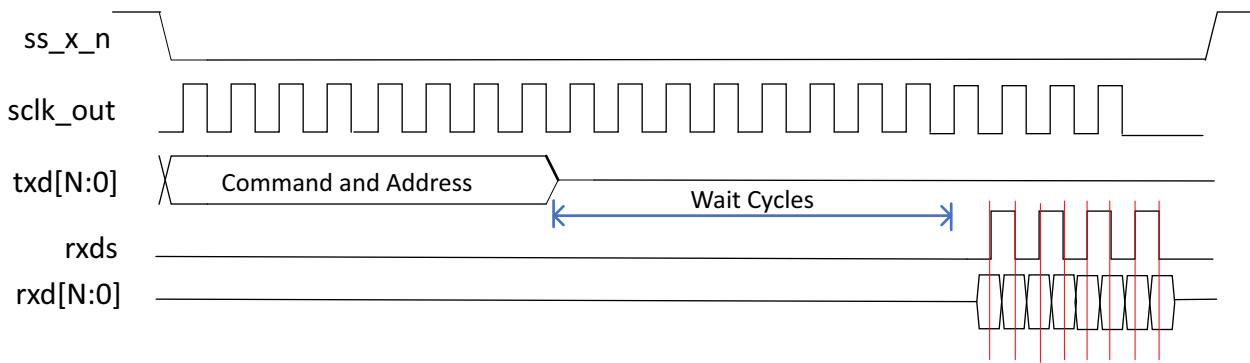
- It is expected that data is stable around clock edge of data strobe signal while sampling on both edges.
- Only SPI mode 0 is supported in this mode, that is SCPH = 0 and SCPO = 0.
- The data strobe signaling is available for enhanced SPI transfers (CTRLR0.SPI\_FRF != 0) only.

For more information, see RXDS Signal Alignment in the DWC\_ssi User Guide.

Read data strobe signaling is supported for both DDR and SDR transfer modes. The transfer mode (SDR and DDR) is decided using the SPI\_CTRLR0.SPI\_DDR\_EN register field. [Figure 2-88](#) and [Figure 2-89](#) show the examples for data sampling in SDR and DDR mode respectively.

**Figure 2-88 Data Sampling Using RXDS Signal in SDR Mode**

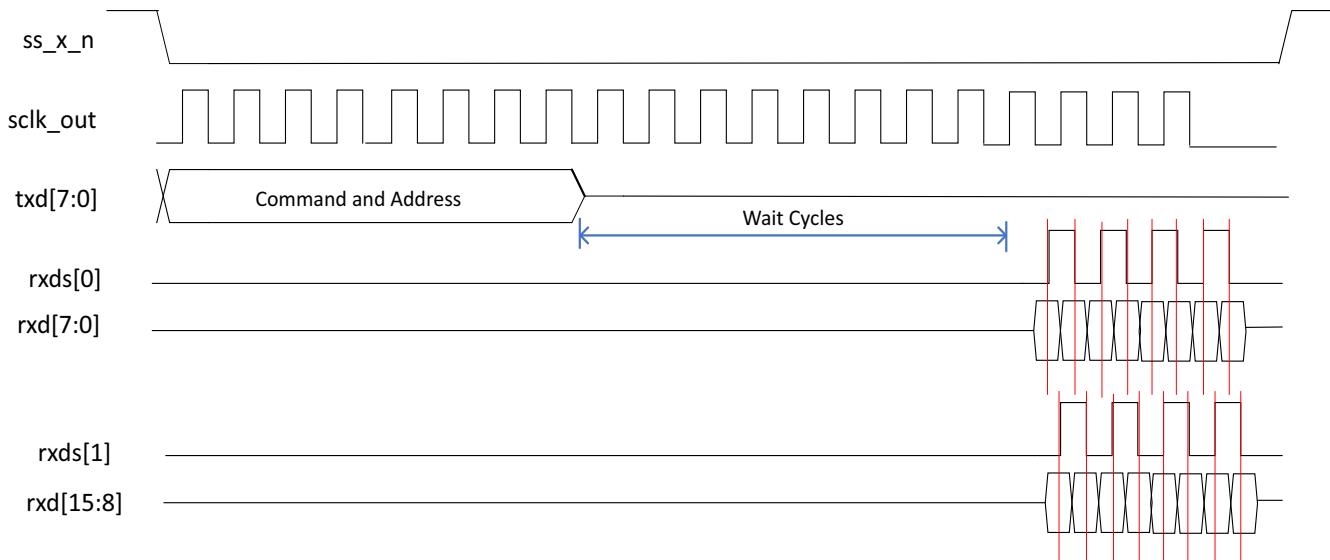


**Figure 2-89 Data Sampling Using RXDS signal in DDR mode**

You can enable the Dual Octal transfers by setting `SPI_CTRLR0 . TRANS_TYPE` to `2'b11` when `SSIC_SPI_MODE` is set to 4. In this mode, the `rxds` signal is 2 bits wide and the `rxd` signal is 16 bits wide. The command and address are transmitted on `txd[7:0]` and data is received on the `rxd[15:0]`. The `rxd[0]` signal is used to sample `rxd[7:0]` and `rxds[1]` is used to sample `rxd[15:8]`.



**Note** Dual Octal transfer mode is supported only for DDR RXDS transfers.

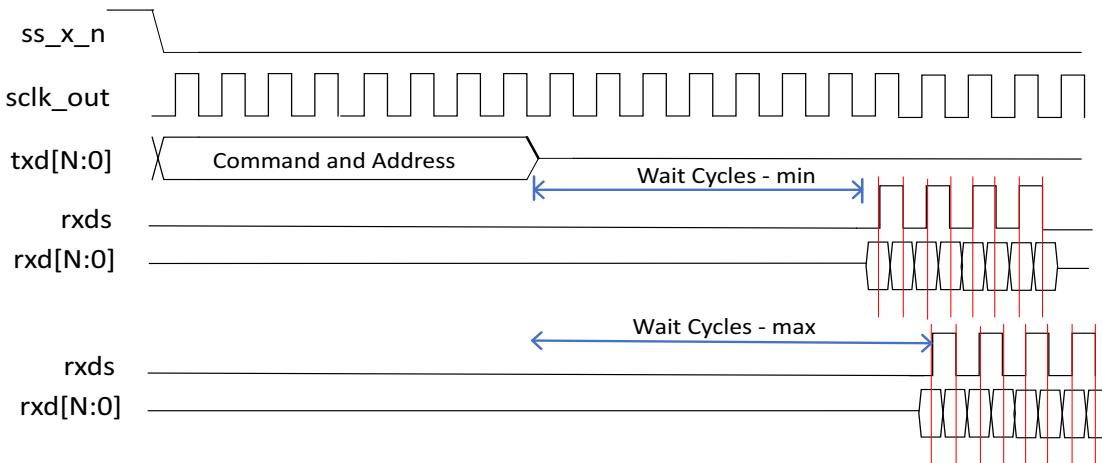
**Figure 2-90 Dual Octal RXDS DDR Transfer**

### 2.21.1 Variable Latency Support

DWC\_ssi provides a programming option `SPI_CTRLR0 . RXDS_VL_EN` or `XIP_CTRLR . RXDS_VL_EN` (for Concurrent XIP operation) to enable extra clock edges during the RXDS transfer. In this mode the IP sends clocks until the required data is received by the controller. Once that is done DWC\_ssi de-asserts the chip-select and the rest of the clocks sent by the device (if any) are ignored. You can use this feature for the devices which used variable wait cycles (or variable latency) for read data strobe transfers. [Figure 2-91](#)

shows an example of a transfer where the device can incur a variable delay to send the Read data strobe -rxds signal.

**Figure 2-91 Variable Wait Cycle Transfer**



### Note

- This feature is only applicable for the RXDS Read transfers.
- The extra data from the SPI target (if any) is ignored by the DWC\_ssi.
- You should enable this feature only when the end device supports variable latency or the Row boundary crossing features.
- For the variable latency, the wait cycles for the RXDS transfers should be set as minimum latency period described by the device.
- The Variable Latency feature is not applicable for XIP Prefetch and continuous transfers.
- The Variable Latency feature is not applicable for transfers where clock stretching is enabled.

#### 2.21.1.1 Design for Test

When the SSIC\_HAS\_RXDS parameter is set to 1, then the rxds signal is used to capture data on both the edges. One of the internal shifter used to negative edge of the rxds signal and to capture the data on rxd line. During the scan testing, these flip-flops might remain uncovered. Therefore, you must connect the scan\_mode to the chip-level scan mode. During scan mode (`scan_mode = 1`), the clock input for these flip-flops are connected to the `ssi_clk` signal, rather than the `rxds` signal. This makes the register testable, and subsequent downstream points controllable.

#### 2.21.2 Enabling Read Data Strobe Signal

To enable data strobe signal on the RXD line in DWC\_ssi, choose the **Yes** option for the **Include data strobe signal for rxd line?** field using the **SPI Parameters** tab during the Specify Configuration Activity in coreConsultant. You can enable this feature only when DDR data transfer is enabled in SPI format and when DWC\_ssi is configured as a serial controller.

### 2.21.3 Signals Related to Read Data Strobe Signal

The following signals are related to Read Data Strobe Signal:

- rxds
- scan\_mode

For more information about these signals, see the *Signals Description* chapter.

### 2.21.4 Registers Related to Read Data Strobe Signal

The following are the registers related to Read Data Strobe Signal:

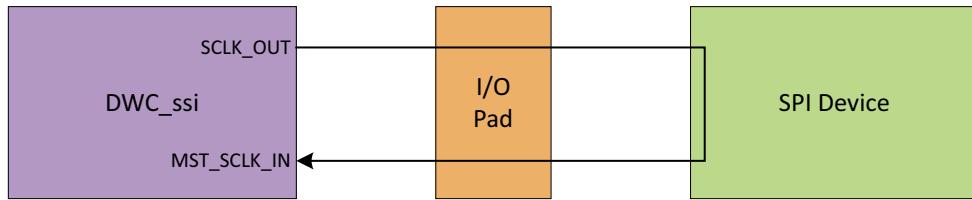
- CTRLR0
- SPI\_CTRLR0
- XIP\_CTRL

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.22 Clock Loop Back Support

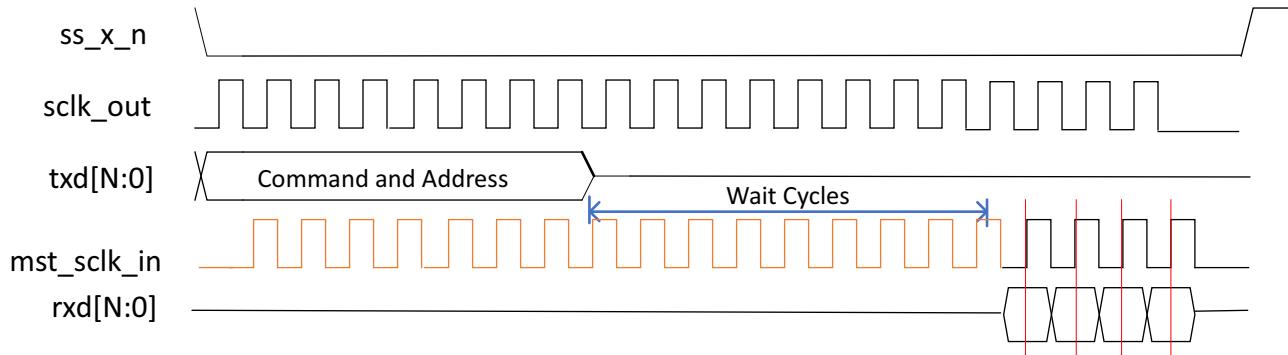
Some devices do not support data strobe signaling, which makes it difficult to achieve higher bandwidths in the SDR support. DWC\_ssi supports the clock loop back feature for such devices. In this mode, the clock out from DWC\_ssi loops back in a way that incurs the same delay as the data signal. The loop back clock driven on the `mst_sclk_in` signal shown in [Figure 2-92](#) is an example on how the loop back clock can be implemented during integration. Apart from the propagation delays, the internal device delay from the device must be managed external to the IP through the Delay line to match the timings with the `rxn` signal.

**Figure 2-92 Clock Loop Back Example**



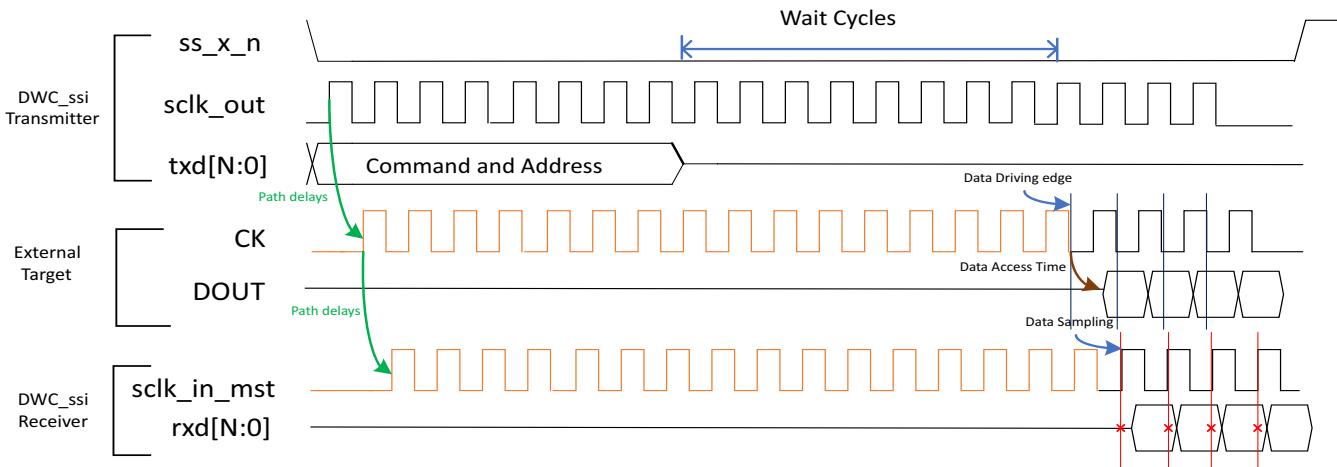
The data is sampled on the positive edge of the loop back clock after removing the command, instruction, and wait cycles from the transfer. [Figure 2-93](#) shows an example of data sampling with clock loop back.

**Figure 2-93 Clock Loop Back Sampling for Data Sampling**



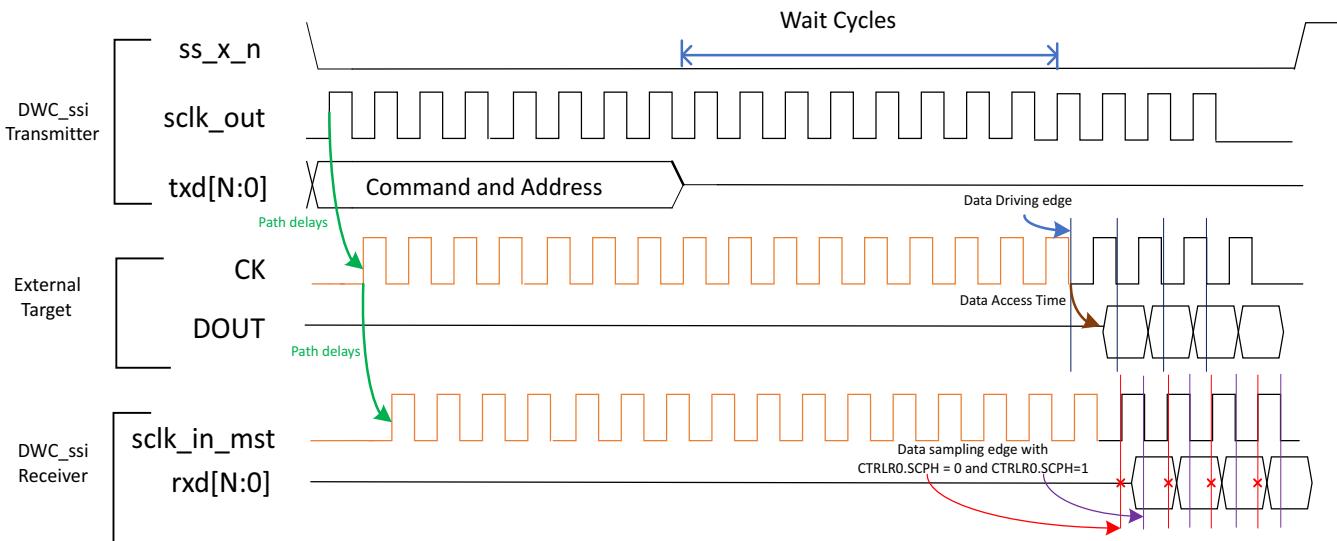
- Only SPI mode 0 is supported in this mode that is `SCPH` = 0 and `SCPOL` = 0.
- Clock Loop back is applicable for all Enhanced SPI and Standard SPI transfers (with Instruction or address phase) Enhanced SPI and Standard SPI transfers (with instruction or address phase)
- Clock Loop back is not applicable for SPI DDR transfers.
- `RX_SAMPLE_DELAY` is not applicable when clock loop back mode is enabled.
- The Clock loop back and read data strobing cannot be enabled at once by the software. The result might be indeterministic.
- Clock Loop mode is not applicable when `CTRLR0.SSTE` is set to 1.
- Clock Loop mode is not applicable when `CTRLR0.SPI_DWS_EN` is set to 1.

There are some target devices that have an access time of more than half of the `sclk_out` period. In such cases, you might not get an appropriate sampling window to sample the data on the positive edge of the incoming looped-back clock. [Figure 2-94](#) shows an example of such a case.

**Figure 2-94 Clock Loop Back With Higher Access Time**

For such cases, DWC\_ssi provides an option to change the sampling edge of the data which enables correct sampling. [Figure 2-95](#) shows an example of this case.

To enable this feature, you need to modify `CTRLR0 . SCPH=1`. As observed in the waveform below, DWC\_ssi shifts the sampling edge (when `CTRLR0 . SCPH=1`) for the incoming data to facilitate the cases where the data access time from the target device is more than 50% of the `sclk_out` period.

**Figure 2-95 Data Sampling Variation With CTRLR0.SCPH Programming**

The data is still transmitted in the SPI 0 format, only the sampling edge changes when `CTRLR0 . SCPH=1` and `CTRLR0 . CLK_LOOP_EN=1`.

## 2.22.1 Design for Test

When SSIC\_HAS\_CLK\_LOOP\_BACK is set to 1 then `mst_sclk_in` is used to capture data in `rxrd`. Since the `mst_sclk_in` is not a free running clock, during the scan testing these flops may remain uncovered. Therefore, you must connect `scan_mode` to the chip-level scan mode.

During scan mode (when `scan_mode = 1`), the clock input for these flip-flops is connected to `ssi_clk`, rather than `mst_sclk_in`, which leaves the register testable and subsequent downstream points controllable.

## 2.23 Data Mask

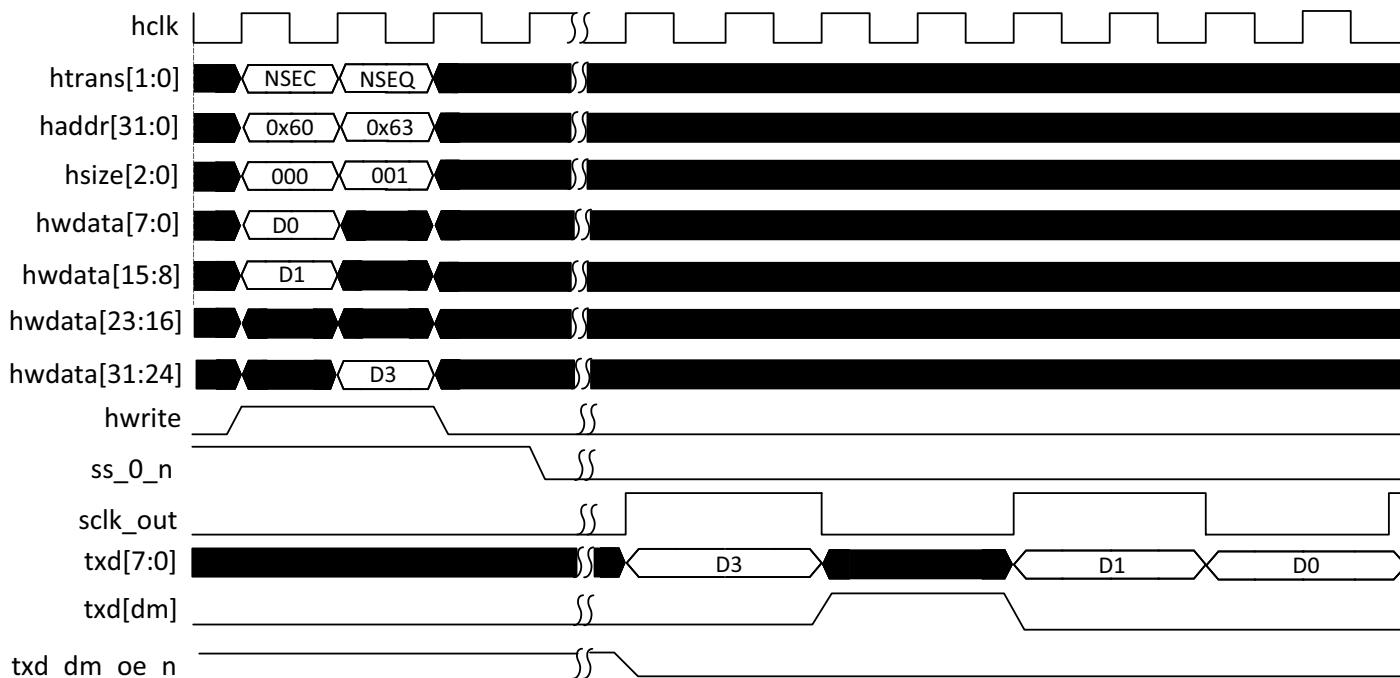
DWC\_ssi supports the data mask feature that is used to mask the write data on an SPI interface. DWC\_ssi supports this feature on enhanced SPI modes. You can use the SSIC\_SPI\_DM\_EN parameter to enable this feature. When configured in data mask mode, DWC\_ssi provides the txd\_dm signal to mask the data on the txd line. This signal is used only during WRITE transactions in enhanced SPI DDR operations.

### 2.23.1 Description of Data Mask

You can set the SPI\_CTRLR0 field of the SPI\_DM\_EN register to 1 to enable the data mask feature. The value of data mask is derived from incoming transactions on the AHB interface to data register. For example, if you want to transmit 4 bytes {D3, D2, D1, D0} (with D3 as MSB) by masking the D2 byte, you must divide the transfer into two transfers such that 16-bit transfer containing {D1, D0} is transferred and then, one 8-bit transfer with D3 being written in the data register. DWC\_ssi enables the data mask signal while transmitting the third byte in transfer.

Figure 2-96 shows a data mask used for Octal SPI write transfer. In Figure 2-96, the data mask signal is active while transmitting the third byte of the data that is not written in the SPI device. The data mask signal txd\_dm is qualified by txd\_dm\_oe\_n signal.

**Figure 2-96 Octal SPI Write Transfer with Data Mask Enabled**





- Each FIFO location in the transmit data FIFO represents a data frame size, which can go from 4 to 32 bits. In case where the data frame size is 32, all the bits of FIFO data are valid and are sent to the SPI interface. But, if the data frame size is smaller than 32, for example 16, then, only 16 bits of each FIFO location are valid. Since the data register is 32 bits wide, all the 32 bits must be written. Among these 32 bits only 16 are valid and the data mask processing happens only for those 16 bits. The reserved 16 bits must also be written and must not be skipped.
- The data mask is only valid for the data phase of the enhanced SPI transfers. The address and instruction phases do not use the data mask. Since the address/command and the data register use the same location, you must write the address/command into the data register before starting to write the data.
- For the last data to be written into the FIFO, the most significant bit must be valid. Otherwise, the partial data is not written into data register.

## 2.23.2 Enabling Data Mask

To enable Data Mask support in DWC\_ssi, choose the **Yes** option for the **Include Data Mask Signal for data transfers in SPI mode?** field using the **SPI Parameters** tab during the Specify Configuration Activity in coreConsultant.

## 2.23.3 Signals Related to Data Mask

The following signals are related to Data Mask:

- txd\_dm\_oe\_n
- txd\_dm

For more information about these signals, see the *Signals Description* chapter.

## 2.23.4 Registers Related to Data Mask

The following are the registers related to Data Mask:

- SPI\_CTRLR0
- XIP\_CTRL

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.24 HyperBus Protocol Support

HyperBus is an interface that supports both parallel and serial interface memories, while enhancing system performance, ease of design, and system cost reduction. A read or write transaction on a HyperBus interface consists of a sequential series of 16-bit, one-clock cycle, data transfers through two corresponding 8-bit width, one-half-clock cycle data transfers, one on each single-ended clock edge or differential clock crossing.

DWC\_ssi supports the HyperBus protocol that works on the principle of the SPI DDR interface. The DDR protocol transfers two data bytes per clock cycle on the input/output (I/O) signals. You can enable the HyperBus feature in DWC\_ssi using the SSIC\_HYPERBUS\_EN configuration parameter.

### 2.24.1 Description of HyperBus Protocol Support

In the HyperBus protocol, read and write transactions transfer complete 16-bit words of data. Read data words always contain two valid bytes. Write data words may have one or both bytes masked to prevent writing of individual bytes within a write burst.

Command, address, and data information are transferred over the eight HyperBus (txd[7:0]/rx[7:0]) signals. The HyperBus target device uses the clock to capture information while receiving the command, address, or data on the DQ signals. The Command or Address values are center aligned with clock transitions. The HyperBus uses 48-bit command-address (CA) to convey the instruction and address information. [Table 2-2](#) shows the CA frame format.

**Table 2-2 Command-Address Bit Assignments**

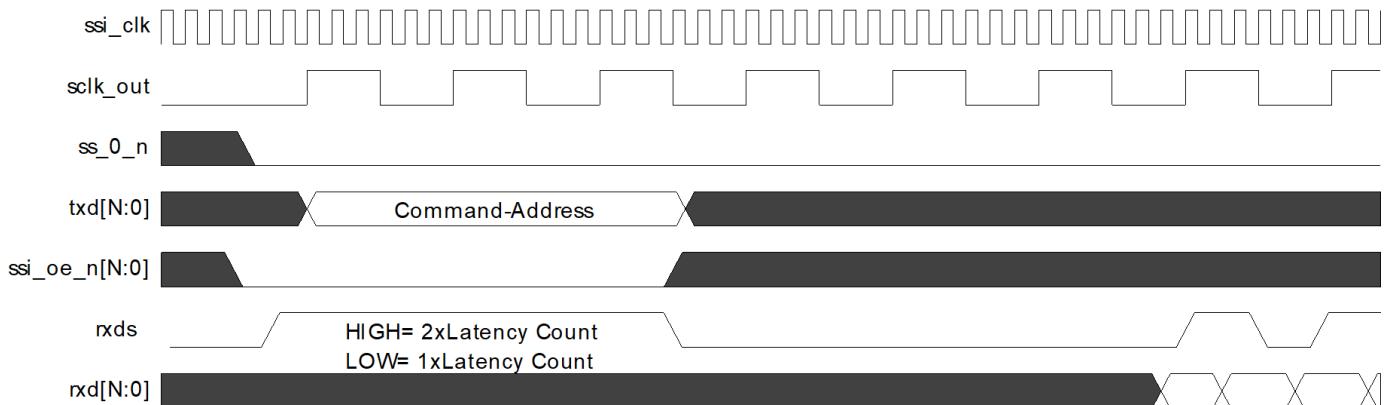
CA Bit	Bit Name	Bit Function
47	R/W#	<p>Identifies the transaction as a read or write.</p> <ul style="list-style-type: none"> <li>■ 1 – Indicates Read transaction.</li> <li>■ 0 – Indicates a Write Transaction.</li> </ul>
46	Address Space (AS)	<p>Indicates whether the read or write transaction accesses the memory or register space.</p> <ul style="list-style-type: none"> <li>■ 0 – Indicates memory space.</li> <li>■ 1 – Indicates the register space.</li> </ul> <p>The register space is used to access device ID and configuration registers.</p>
45	Burst Type	<p>Indicates whether the burst is linear or wrapped.</p> <ul style="list-style-type: none"> <li>■ 0 – Indicates a wrapped burst.</li> <li>■ 1 – Indicates a linear burst.</li> </ul>
44–16	Row & Upper Column Address	<p>Row and Upper Column component of the target address: System word address bits A31-A3</p> <p>If a particular device density does not use upper Row address bits, the Host controller serial controller interface must set it to 0. The size of Rows and therefore the address bit boundary between Row and Column address is dependent on the serial target device.</p>
15–3	Reserved	<p>Reserved for future column address expansion.</p> <p>Reserved bits are don't care in current HyperBus devices but the Host controller must be set to for future compatibility.</p>

CA Bit	Bit Name	Bit Function
2–0	Lower column address	Lower Column component of the target address: System word address bits A2–0 selecting the starting word within a half-page.

## 2.24.2 Read Transactions

The HyperBus serial controller begins a transaction by driving the `ss_x_n` signal to Low while the clock is idle. Then, the clock starts toggling while CA words are transferred. The HyperBus serial controller then continues clocking for a number of cycles defined by the latency count setting in a configuration register. If the Read-Write Data Strobe (RWDS) is low during the CA cycles, one latency count is inserted. If RWDS is high during the CA cycles, an additional latency count is inserted. After these latency clocks are completed, the memory transitions the RWDS and outputs the target data. [Figure 2-97](#) represents a typical HyperBus read command.

**Figure 2-97 HyperBus Read Command**



**Note** Certain HyperBus devices can remove read data strobe signal in between transfers if page boundary is reached. In such a situation, the controller must send the clock continuously until the transfer resumes. DWC\_ssi does not support such transfer and the software must ensure that such situations do not occur before initiating the transfer by DWC\_ssi.

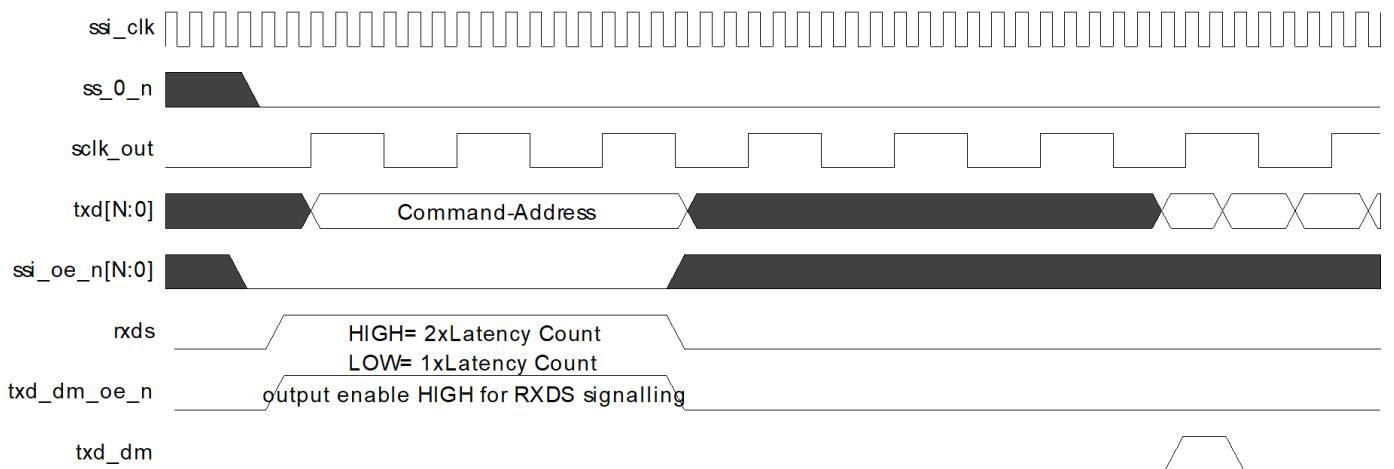
## 2.24.3 Write Transactions

The HyperBus controller begins a transaction by driving the `ss_x_n` signal to Low while the clock is idle. Then, the clock starts toggling while CA words are transferred. Further, the HyperBus controller continues clocking for a number of cycles defined by the latency count setting in a configuration register. If RWDS is Low during the CA cycles, one latency count is inserted. If RWDS is high during the CA cycles, an additional latency count is inserted. After these latency clocks are completed, the HyperBus controller starts to output the target data. The Write data is center aligned with the clock edges. The first byte of data in each word is captured by the memory on the rising edge of `sclk_out` signal and the second byte is captured on the falling edge of `sclk_out` signal. The latency count can also be zero for a write transaction that is device dependent.

For a HyperBus device, the read data strobe signal is driven by the device during the Command-Address (CA) phase and the serial controller uses this signal as data mask during the write data phase. DWC\_ssi provides a separate data mask signal that you can use with the rxds signal to form a bi-directional bus to connect with the serial target. DWC\_ssi provides the data mask output enable signal to create a bi-direction pin to connect to the serial target. [Figure 2-98](#) represents a typical HyperBus Write command.

For more information on data mask support, see “[Data Mask](#)” on page [131](#).

**Figure 2-98 HyperBus Write Command**



## 2.24.4 Setting the CTRLR0 Register to Enable the HyperBus Feature

When the HyperBus feature is enabled (`SSIC_HYPERBUS_EN = 1`) in `DWC_ssi`, the `CTRLR0.SPI_HYPERBUS_EN` register bit is set to program `DWC_ssi` as a HyperBus controller. As the HyperBus interface has a fixed frame format, the following fields are fixed when `DWC_ssi` is programmed into HyperBus mode:

- `CTRLR0.SPI_FRF` – 2'b11 (Octal Frame format)
- `SPI_CTRLR0.TRANS_TYPE` – 2'b10 (Address to be sent in Octal format)
- `SPI_CTRLR0.ADDR_L` – 4'b1100 (Address length of 48 bits)
- `SPI_CTRLR0.INST_L` – 2'b00 (No Instruction phase)
- `SPI_CTRLR0.SPI_DDR_EN` – 1 (Address to be sent on Octal-DDR format)
- `SPI_CTRLR0.SPI_RXDS_EN` – 1 (Enable read data strobe to sample incoming data)

You can configure Wait cycles using the `SPI_CTRLR0.WAIT_CYCLES` field, depending on memory requirement. As the HyperBus protocol counts the Wait cycles from the third CA cycle, `DWC_ssi` sends `SPI_CTRLR0.WAIT_CYCLES-1` wait cycles after the CA phase is complete. The remaining fields in the `SPI_CTRLR0` register are not used for a HyperBus transfer.

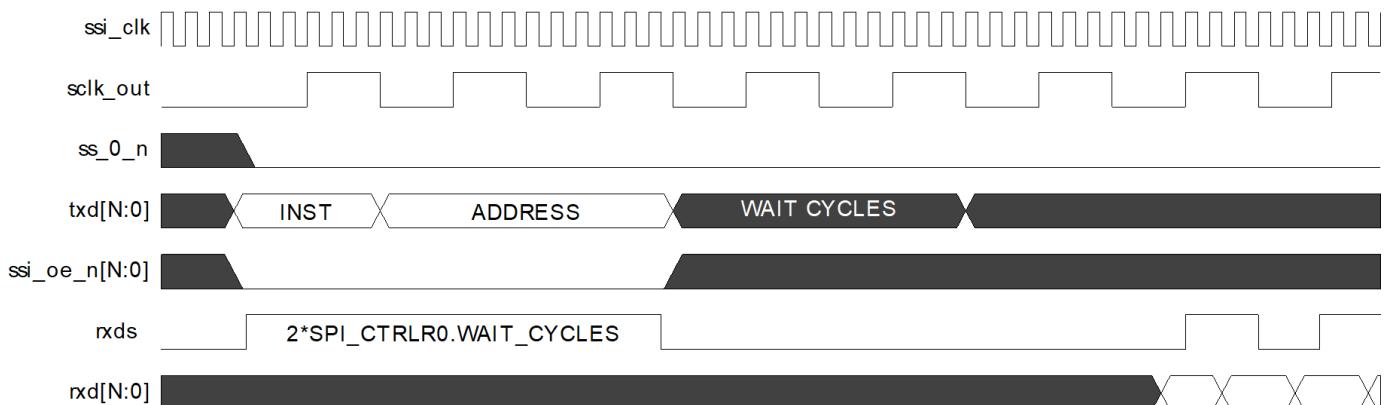
You can set the type of transaction (read/write) and data frame size in `CTRLR0.TMOD` and `CTRLR0.DFS` fields, respectively. The read back value of fixed fields is same as the register value but internally the hardware ignores those when `CTRLR0.SPI_HYPERBUS_EN` bit is set to 1.

After `CTRLR0.SPI_HYPERBUS_EN` is set to 1, `DWC_ssi` expects a 48-bit address to start the transaction. The 48-bit data must be packed according to the HyperBus specification as described in [Table 2-2](#).

## 2.24.5 RXDS Signaling Support in Command-Address Phase

DWC\_ssi provides an option for RXDS signaling in Command-Address (CA) phase as required by the HyperBus protocol. You can enable this feature by programming the SPI\_CTRLR0.RXDS\_SIG\_EN bit to 1. In this mode, if the rxds signal is detected as HIGH during command and address cycles, then DWC\_ssi sends  $(2 * \text{SPI\_CTRLR0.WAIT\_CYCLES} - 1)$  Wait cycles after the address phase is complete. [Figure 2-99](#) shows an example of rxds signaling during CA phase. If the SPI\_CTRLR0.RXDS\_SIG\_EN bit is set to 0 or if the rxds level is 0 during CA phase, the number of Wait cycles is fixed to the programmed value in the SPI\_CTRLR0.WAIT\_CYCLES bit.

**Figure 2-99 RXDS Signaling in Command- Address Phase**



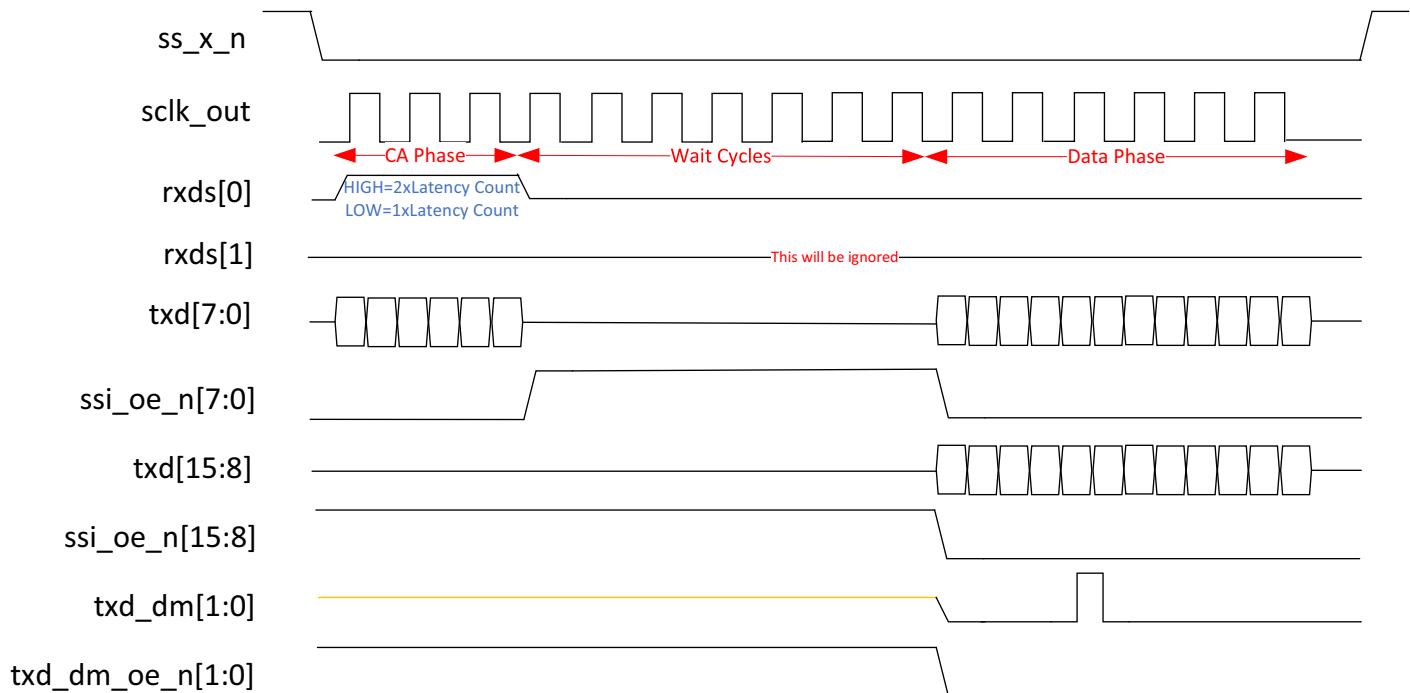
## 2.24.6 HyperBus 2.0 and 2.0e Support

HyperBus 2.0 and 2.0e protocol enables the data transfer on 16 data lines. This section describes the protocol defined for 16 data lines on SPI interface. Enable the SSIC\_SPI\_MODE configuration parameter to configure the IP to support this protocol.

### 2.24.6.1 HyperBus x16 Write Operation

To enable HyperBus transfer for 16 lanes, set SPI\_CTRLR0.TRANS\_TYPE to 2'b11. The HyperBus transfer on 16 lanes works in the same manner. The HyperBus controller transmits the Command and Address (CA Phase) information on 8 lanes. This will be followed by wait cycles and then the data will be sent on all 16 lanes. DWC\_ssi uses RXDS[0] (RWDS[0] as per HyperBus Specification) to increase the latency count as per the requirement. RXDS[1] is not used during the address phase.

[Figure 2-100](#) shows an example SPI x16 Hyperbus write transfer.

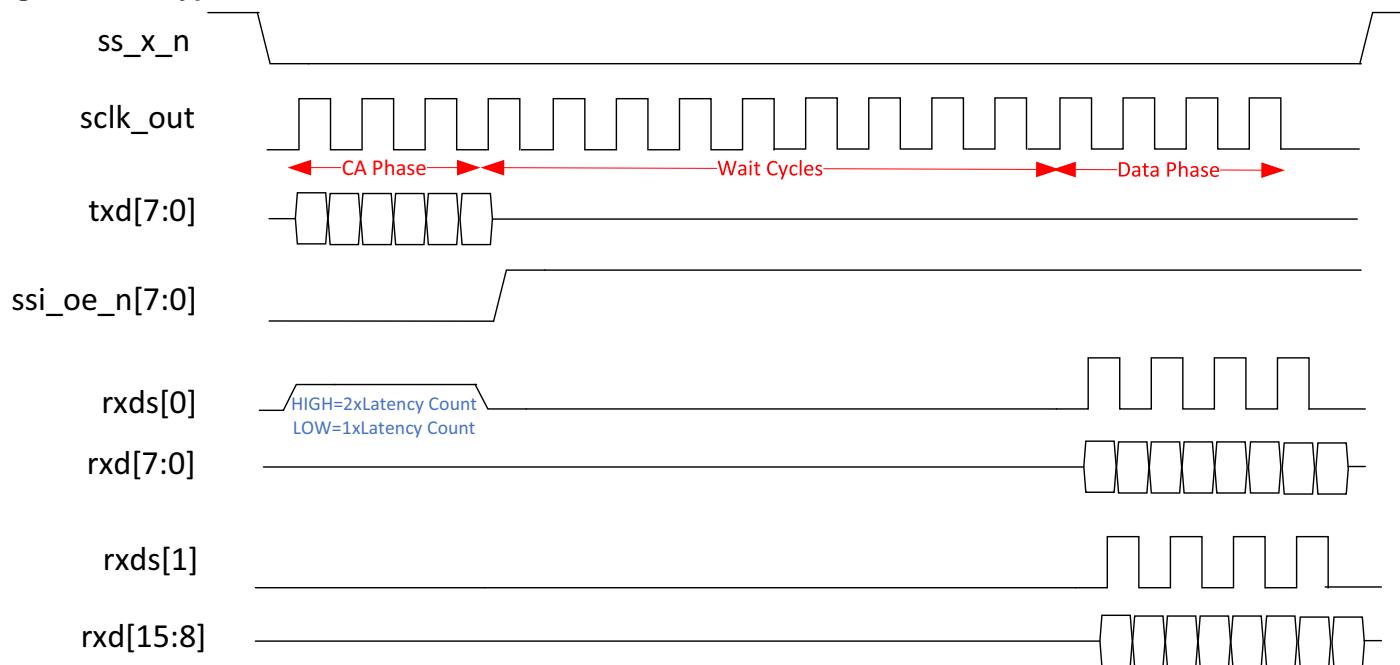
**Figure 2-100 HyperBus x16 Write Transfer**

### 2.24.6.2 HyperBus x16 Read Operation

Similar to the write transfer, the read transfer also sends the CA information on 8 lanes and then expects the data on the 16 lanes. The RXDS[0] is used to sample the data on RXD[7:0]; and RXDS[1] is used to sample the data on RXD[15:0].

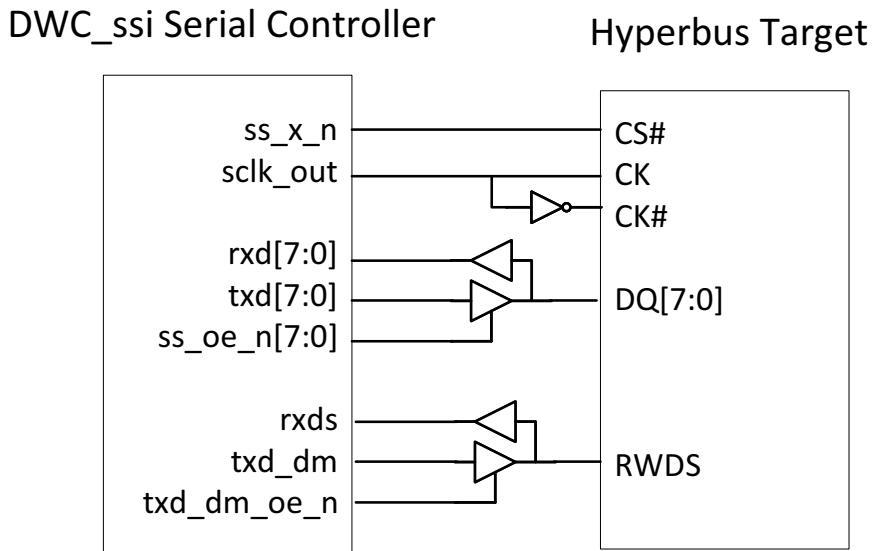
The programming guidelines are described in “[Set the following register fields for a dual-octal write operation:](#)” on page [105](#). Similar to the write operation, there are two data strobes for the read data, one data strobe for RXD[7:0] and RXD[15:8], which will be used to sample the incoming data.

[Figure 2-101](#) shows an example Read transfer in x16 Hyperbus mode.

**Figure 2-101 HyperBus x16 Read Transfer**

## 2.24.7 HyperBus Connections

Figure 2-102 shows signal connections from DWC\_ssi serial controller with a typical HyperBus target device.

**Figure 2-102 HyperBus Connections**

DWC\_ssi does not support optional signals (RESET# [Hardware Reset] and RSTO# [Reset Output]) or to transfer an interrupt notification to controller [INT#]). If used, they should be connected via glue logic.

## 2.24.8 Enabling the HyperBus Feature

To enable the HyperBus feature in DWC\_ssi, choose the **Yes** option for the **Enable HyperBus Transfer mode?** field using the SPI Parameters tab during the Specify Configuration Activity in coreConsultant.

## 2.24.9 Registers Related to HyperBus Feature

The following are the registers related to HyperBus Protocol:

- CTRLR0
- SPI\_CTRLR0
- XIP\_CTRL

For more information about these registers, see the *Registers Descriptions* chapter.

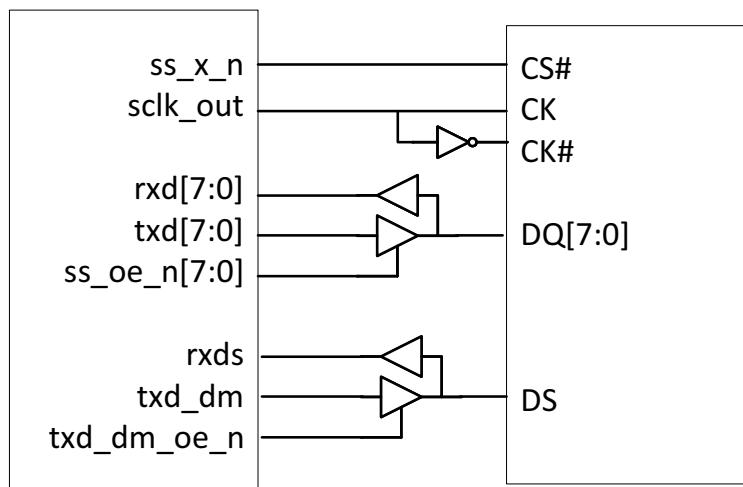
## 2.25 JEDEC xSPI Protocol Support

eXpanded Serial Peripheral Interface (xSPI) is a standard for Non-volatile Memory devices, which provides high data throughput, with low signal count. It is primarily used in computing, automotive, Internet of Things (IoT), embedded systems, and mobile systems between host processing and peripheral devices. The xSPI electrical interface can deliver up to 400 Megabytes per second raw data throughput. DWC\_ssi supports all the command formats described in JEDEC xSPI version 1.0.

[Figure 2-103](#) shows signal connections from DWC\_ssi serial controller with a typical xSPI target device.

**Figure 2-103 Signal Connections from DWC\_ssi Serial Controller with xSPI Target**

DWC\_ssi Serial Controller                            xSPI Target



DWC\_ssi does not support the optional signals (RESET# [Hardware Reset] and RSTO# [Reset Output]) or to transfer an interrupt notification to serial controller[INT#]). If used, they should be connected via a glue logic.

xSPI supports two command modes:

- 1S-1S-1S: One IO signal used during command transfer, command modifier transfer, and data transfer. All phases are SDR.
- 8D-8D-8D: Eight IO signals used during command transfer, command modifier transfer, and data transfer. All phases are DDR.

To support both the profiles, following parameters should be enabled:

- SSIC\_SPI\_MODE = 3 (Octal SPI Support)
- SSIC\_HAS\_DDR = 1 (Dual Data Rate Support)
- SSIC\_HAS\_RXDS =11 (Read Data Strobe Support)
- SSIC\_SPI\_DM\_EN = 1 (Data Mask feature for DDR transfers)
- SSIC\_HYPERBUS\_EN = 1 (RXDS Signaling support)

[Table 2-3](#) shows the support for transaction formats in 1S-1S-1S protocol mode:

**Table 2-3 Transaction Formats in 1S-1S-1S Protocol Mode**

<b>Command Name</b>	<b>DWC_ssi Support</b>
Format 0.A: Command only	Standard SPI Transmit only mode
Format 0.B Command and Read Data	
Format 0.C Command, 3-byte Address, Read Data	
Format 0.D Command, 4-byte Address, Read Data	
Format 0.E Command, 3-byte Address, Initial Access Latency (Temporary Cycles), Read Data	Standard SPI EEPROM mode
Format 0.F Command, 4-byte Address, Initial Access Latency (Temporary Cycles), Read Data	
Format 0.G Command and Write Data	
Format 0.H Command and 3-byte Address	
Format 0.I Command and 4-byte Address	Standard SPI Transmit only mode
Format 0.J Command, 3-byte Address, Write Data	
Format 0.K Command, 4-byte Address, Write Data	

Table 2-4 shows the support for transaction formats in 8D-8D-8D protocol mode:

**Table 2-4 Transaction Formats in 8D-8D-8D Protocol Mode**

<b>Command Name</b>	<b>DWC_ssi Support</b>
Format 1.A: Command and Command Extension	Octal DDR Write with Command only
Format 1.B: Command, Command Extension, 4-byte Address, 'n' Latency Cycles, and Read Data	Octal DDR read with data strobe
Format 1.C: Command, Command Extension and 4-byte Address	Octal DDR Write with Command and address only transfer
Format 1.D: Command, Command Extension, 4-byte Address, and Write Data	Octal DDR Write transfer
Format 2.A: Command, 5-byte Address, Initial Access Latency	HyperBus Read/Write command without RXDS signaling
Format 2.B: Command, 5-byte Address, Initial Access Latency	HyperBus Read/Write command with RXDS signaling

Following are the relevant programming fields to achieve all transaction formats:

- `CTRLR0.DFS`: Data frame size for a SPI transfer
- `CTRLR0.FRF`: Frame format for serial transfer (set to 0 for all SPI transfers)
- `CTRLR0.SPI_FRF`: SPI Frame format (set to 0x0 for 1S-1S-1S and 0x3 for 8D-8D-8D)

- SPI\_CTRLR0.TRANS\_TYPE: Address and instruction transfer format (set to 0x2 for 8D-8D-8D)
- CTRLR0.SPI\_HYPERBUS\_EN: HyperBus protocol enable
- SPI\_CTRLR0.INST\_L: Instruction length for Octal transfers
- SPI\_CTRLR0.ADDR\_L: Address length for Octal transfers
- SPI\_CTRLR0.WAIT\_CYCLES: Wait cycles length for Octal transfers
- SPI\_CTRLR0.SPI\_DM\_EN: Data masking enable for Octal DDR write transfers
- SPI\_CTRLR0.SPI\_RXDS\_SIG\_EN: RXDS signaling enable during command and address phase for HyperBus transfers

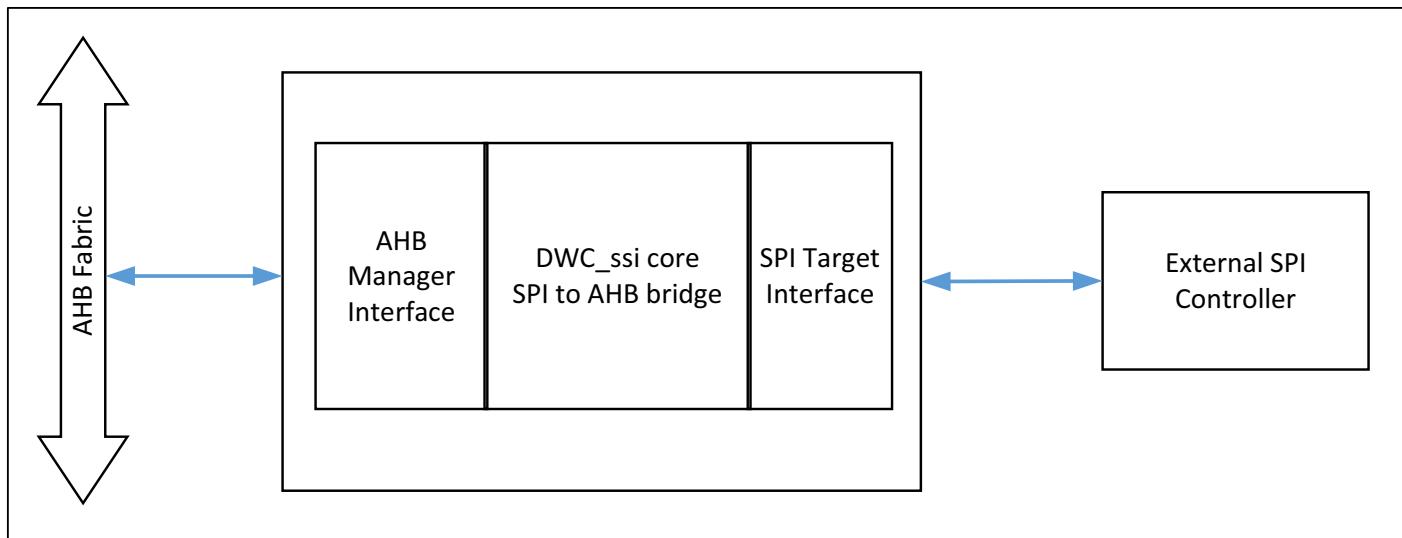
## 2.26 SPI Bridge Feature

When DWC\_ssi is configured for SPI Bridge operation (`SSIC_SPI_BRIDGE = 1`), DWC\_ssi converts all the transfers from SPI to AHB transactions. External SPI controller sends address to DWC\_ssi. DWC\_ssi takes the address, and converts it into an AHB transfer. This enables external controller to access the internal memory directly without any CPU intervention. DWC\_ssi can be enabled through the boot mode feature on the Reset and Load boot program from external controller if required. [Figure 2-104](#) shows the SPI target interface and AHB manager interface in bridge subsystem. In this example, SPI target interface is a primary interface and AHB Manager is a secondary interface.



The bridge mode of operation is supported only when the DWC-SSI-SPI-SLV-BRIDGE plus license exists.

**Figure 2-104 SPI Target to AHB Manager Bridge Subsystem**



SPI Bridge configuration parameter `SSIC_SPI_BRIDGE` is enabled only if DWC\_ssi is in enhanced SPI target mode (`SSIC_SLV_SPI_MODE != 0`).



When `SSIC_SPI_BRIDGE` parameter is enabled (`SSIC_SPI_BRIDGE = 1`), DWC\_ssi can only be used as bridge between SPI controller interface and AHB subsystem. Legacy operations, such as Microwire, SSP Protocol support in DWC\_ssi serial Target are not available.

All the SPI transfers have following phases:

1. Instruction Phase
2. Address Phase
3. Data Phase

## 2.26.1 Instruction Phase

Every transfer from SPI controller starts with an 8-bit instruction, which determines the transfer type and process for the rest of the transfer. [Table 2-5](#) shows the instruction field decode.

**Table 2-5 Instruction Field Decode**

Bit	Bit Name	Description
7	Read/Write	<p>Defines whether the transfer is read or write.</p> <ul style="list-style-type: none"> <li>■ 0: Read</li> <li>■ 1: Write</li> </ul>
6:5	Transfer Type	<p>Defines the transfer type.</p> <ul style="list-style-type: none"> <li>■ 00: Read/Write Data</li> <li>■ 01: Read Request</li> <li>■ 10: Read/Write Status</li> <li>■ 11: Read Data with Temporary</li> </ul>
4:3	Wait Cycles	<p>Defines the Wait Cycles for Read Transactions. Selects the number of wait cycles for a read transfer. This field is only applicable for the Read data transfers. When the transfer type is 00, the field decode is as follows:</p> <ul style="list-style-type: none"> <li>■ 00: 0 Wait Cycle</li> <li>■ 01: 1 Wait Cycle</li> <li>■ 10: 2 Wait Cycles</li> <li>■ 11: Reserved</li> </ul> <p>When the transfer type is 11, the field decode is as follows:</p> <ul style="list-style-type: none"> <li>■ 00: 8 Wait Cycles</li> <li>■ 01: 16 Wait Cycles</li> <li>■ 10: 24 Wait Cycles</li> <li>■ 11: 32 Wait Cycles</li> </ul> <p>Note: Wait Cycles are not applicable for Write Data operation.</p>
2:0	Data Length	<p>Defines the data to be transmitted in the current transfer.</p> <p>3'b000 = 1      3'b001 = 4      3'b010 = 8      3'b011 = 16      3'b100 = 32      3'b101 = 64      3'b110 = Reserved      3'b111 = Reserved</p> <p>Note: Each Frame size is of 32 bits (4 bytes)</p>

## 2.26.2 Address Phase

Address phase determines the address from which the data should be read or written on a secondary interface (AHB). Address could be received in standard/dual/quad/octal/dual octal frame format, which is defined by `spi_mode_slv` input signal.

AHB interface address width is fixed to 32 bits. SPI interface expects to receive 24-bit address, which is sent to AHB interface. Since the data is always read/written in frame of 32 bits the address is always word aligned. The 32-bit address on the AHB interface is derived using the following formula:

`SSIC_AHB_ADDR_OFFSET + {{SPI_ADDR}}, 2'b00}`

`SPI_ADDR` = 24-bits address received on SPI Interface

## 2.26.3 Data Phase

Data is received or transmitted in frames of 32 bits. The amount of data to be transmitted or received is defined by the Data Length field of Instruction phase.

## 2.26.4 Bridge Operation

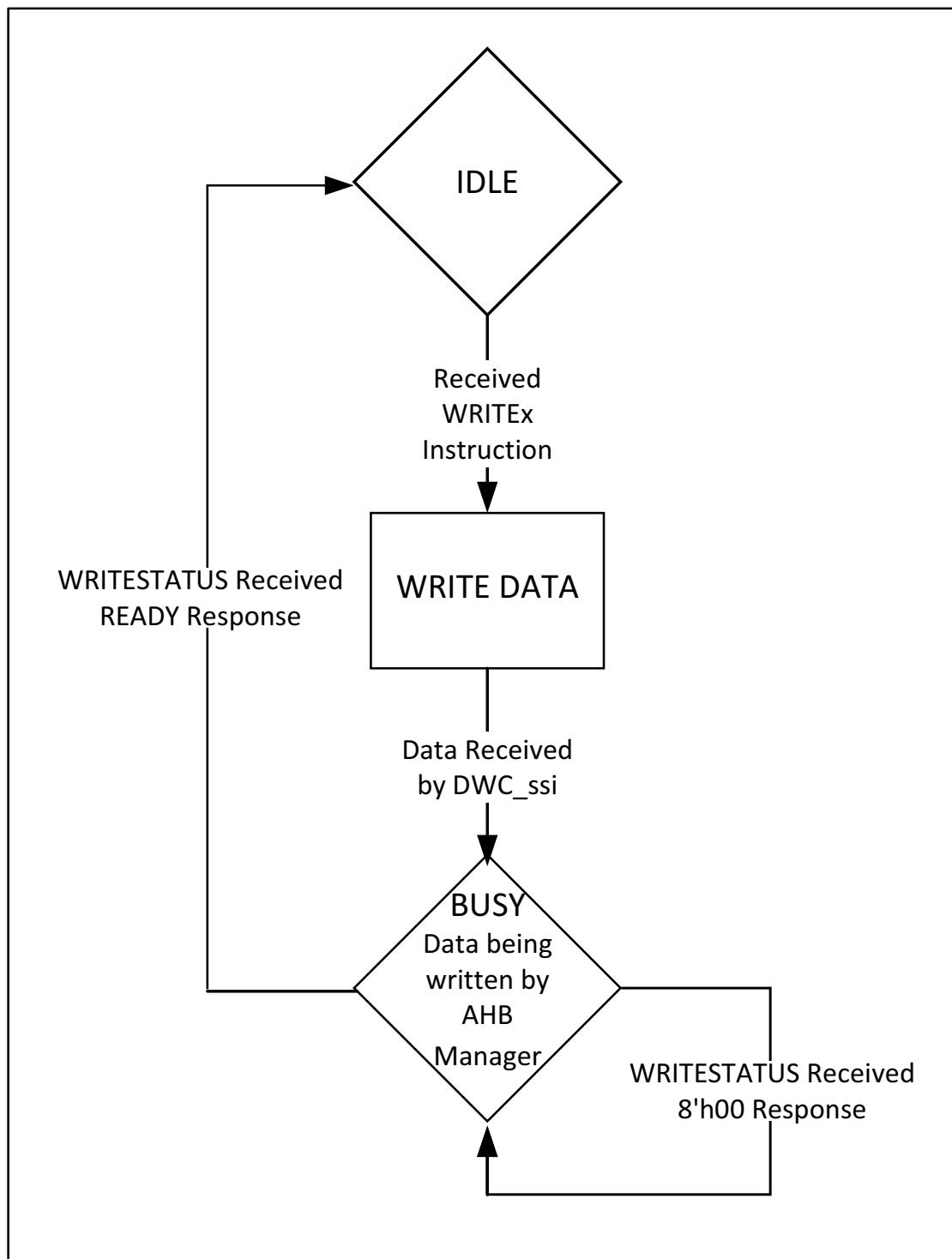
DWC\_ssi performs as a bridge between SPI controller, and AHB subsystem as shown in [Figure 2-104](#). The transfers from SPI target are directly converted in AHB read/write operations. The 8-bit instruction defines the characteristics of the transfer and direction of data flow, and `spi_mode_slv` input pin defines SPI operating mode. Following sections describe the read and write operations.

### 2.26.4.1 Write Operation

Following are the three phases applicable for write operation:

1. Instruction Phase
2. Address Phase
3. Data Phase

[Figure 2-105](#) captures how DWC\_ssi performs write operation.

**Figure 2-105 Write Operation Flow**

### Instruction phase

Determines the amount of data to be transmitted during write operation. The SPI frame format is derived from `spi_mode_slv` input pin. SPI target supports the following instruction opcodes:

**Table 2-6 Instruction Set for Write Data Operations**

<b>Instruction</b>	<b>Instruction Name</b>	<b>Description</b>
8'h8x x=0/1/2/3/4/5	WRITEn	Write operation X=0/1/2/3/4/5 n=1/4/8/16/32/64 Lower 3 bits of instruction determines the amount of data to be received by SPI target. For field decode, see <a href="#">Table 2-5</a> .
8'hC0	WRITESTATUS0	Write Data Status instruction
8'hC8	WRITESTATUS1	Write Data Status instruction with 1 wait cycle
8'hD0	WRITESTATUS2	Write Data Status instruction with 2 wait cycles

## Address Phase

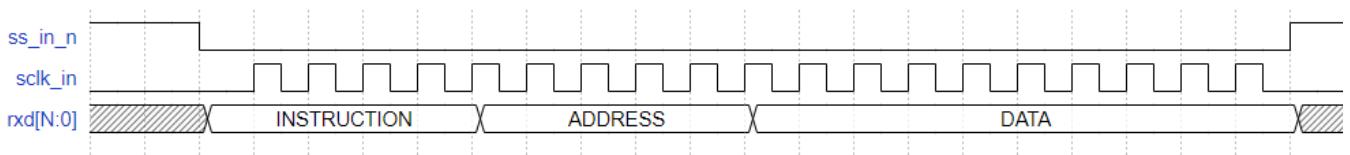
Address from SPI target is directly sent to the AHB interface as per the rules described in “[Address Phase](#)” on page [145](#).

## Data Phase

The data on AHB interface is transmitted in bursts, DWC\_ssi waits for the defined amount of data to initiate a burst on AHB interface, the amount of data in each burst is specified in “[Data Phase](#)” on page [145](#).

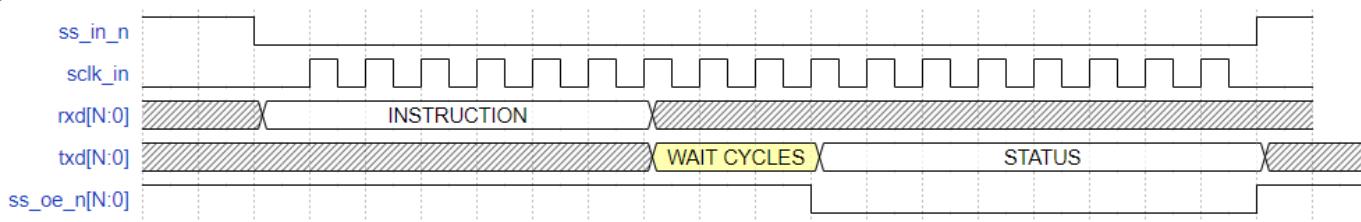
### WRITE<sub>x</sub>:

Data Write transaction is always performed in the format specified by spi\_mode\_slv input pin. The instruction, data, and address phase are sent in single/dual/quad or octal format.

**Figure 2-106 WRITE<sub>x</sub> Instruction**

- N = 0 for standard SPI Mode
- N = 1 for Dual SPI Mode
- N = 3 for Quad SPI Mode
- N = 7 for Octal SPI Mode

WRITESTATUS<sub>x</sub>: After the write operation is completed by the controller, it should poll for status of the write operation using instruction opcode 8'hC0/8'hC8/8'hD0. Only when the controller receives a READY response, it should start a new transfer.

**Figure 2-107 Write Status Instruction**

After SPI controller receives a READY response from DWC\_ssi, it can move on to the next operation. The AHB interface may get an ERROR response for the write transfer. If AHB interface gets an ERROR response, the WRITESTATUSx instruction indicates the same by sending a response of 8'hC0. SPI controller can then issue a new WRITE transfer, or retry the same operation.



**Note** WRITESTATUS is an optional command for SPI controller. If the system can make sure that the previous write transfer completes on AHB interface before issuing a new request by SPI controller, then this command can be skipped.

**Table 2-7 Response Field Decode**

Bit	Name	Description
7	READY	Indicates whether the data is ready to be fetched from DWC_ssi. <ul style="list-style-type: none"> <li>■ 0: Not Ready</li> <li>■ 1: Ready</li> </ul>
6	ERROR	Indicates whether AHB transaction ended with an ERROR response or not. <ul style="list-style-type: none"> <li>■ 0: No Error</li> <li>■ 1: Error</li> </ul>
[5:0]	Reserved	Reserved and driven to 0 always.

#### 2.26.4.2 Read Operation

In enhanced SPI (Dual/Quad/Octal/Dual Octal) modes all the signals are bi-directional; hence, controller and target should know when to sample or drive the data on the line. The read instruction set defines the protocol under which the read transactions should occur. There are two ways in which SPI Controller can read data through SPI bridge:

- Direct read Instruction
- Read Data in three stages (this requires poling)

##### Direct Read Instruction

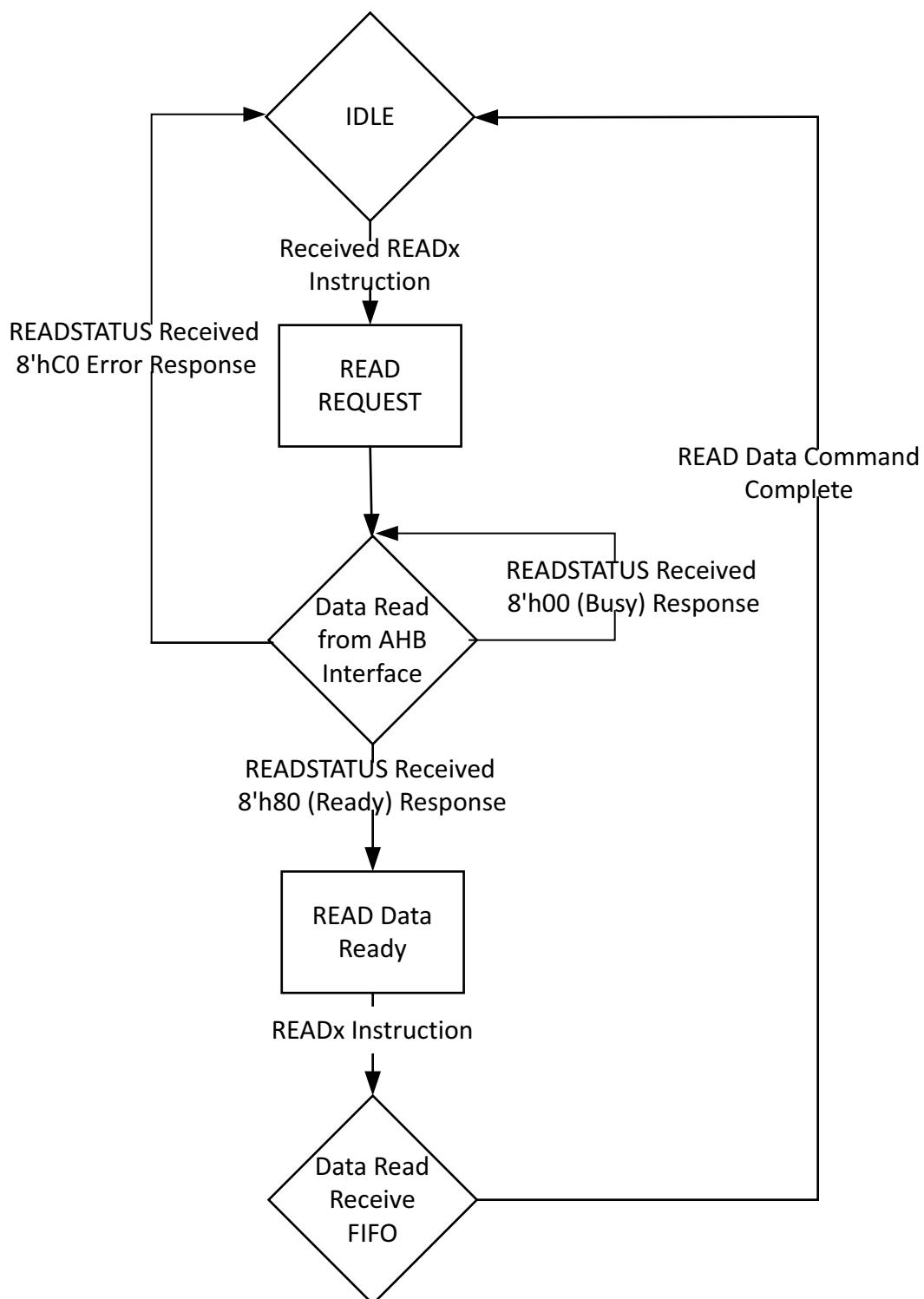
In Direct Read Instruction method SPI Controller initiates a read command and expects the data to be returned in the same command. This way the data is read in the same instruction. You need to make sure that the data is read on the AHB manager interface within the specified cycles (TEMPORARY cycles).

## Read Data in Three Stages

In some cases, the delay involved during the read operation is such that the read data availability cannot be guaranteed in terms of SPI clock cycles. To make sure that the data is read in the expected way during the read operation, DWC\_ssi provides an alternate approach to divide the read operation into three parts:

1. Read Request phase
2. Wait State
3. Read Data phase

[Figure 2-108](#) shows the flow for read operation.

**Figure 2-108 Read Operation Flow for Three Stage Operation**

SPI controller requests the read data from SPI target by sending only the instruction and address during the read request phase. The length of data (1/4/8/16/32/64) to be fetched is also encoded into the instruction. This is converted into an AHB transfer, and the data is fetched and stored in the internal FIFO. After the Read request phase, SPI controller checks for availability of data by polling through READSTATUS instruction. If SPI controller receives a positive response for that instruction, then it reads the data using READDx instruction. DWC\_ssi then sends all the data received from AHB interface to SPI controller.

[Table 2-8](#) shows all the instructions applicable for READ operation.

**Table 2-8 Instruction set for READ Operations**

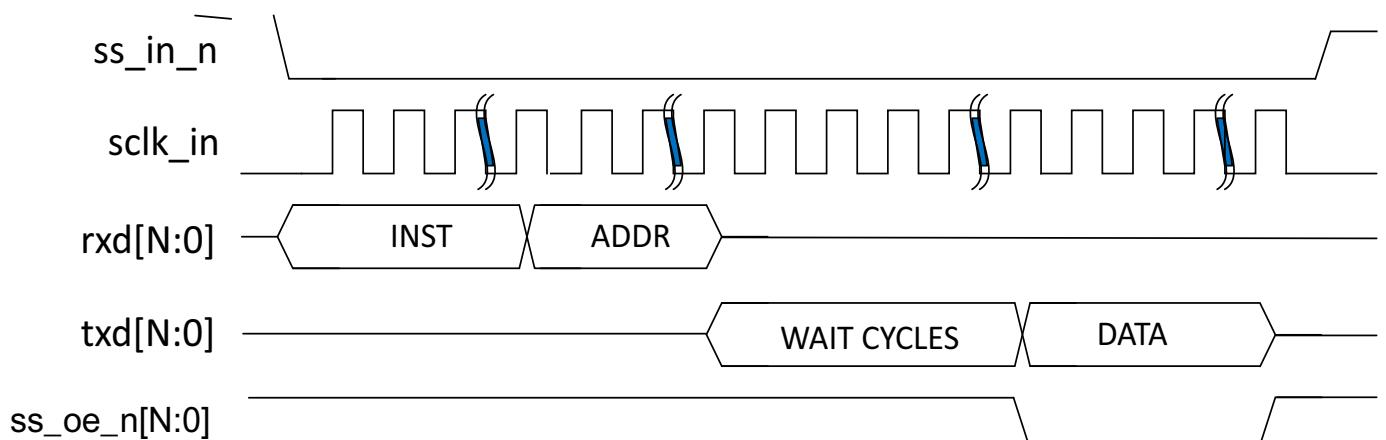
Instruction	Instruction Name	Description
8'h2x x=0/1/2/3/4/5	READREQn	<p>Read request. n=1/4/8/16/32/64</p> <p>Lower 3 bits of instruction determines amount of data to be fetched by AHB manager. For field decode, see <a href="#">Table 2-5</a>.</p> <p>Note: This instruction is applicable for three stage read approach.</p>
8'h0x x=0/1/2/3/4/5	READ0n	<p>Read Data Instruction (without wait cycles). n=1/4/8/16/32/64</p> <p>Lower 3 bits of instruction determines amount of data to be fetched by AHB manager. For field decode, see <a href="#">Table 2-5</a>.</p> <p>Note: This instruction is applicable for three stage read approach.</p>
8'h0x x=8/9/A/B/C/D	READ1n	<p>Read data with 1 Wait Cycle. n=1/4/8/16/32/64</p> <p>Lower 3 bits of instruction determines amount of data to be fetched by AHB manager. For field decode, see <a href="#">Table 2-5</a>.</p> <p>Note: This instruction is applicable for three stage read approach.</p>
8'h1x x=0/1/2/3/4/5	READ2n	<p>Read data with 2 Wait Cycles. n=1/4/8/16/32/64</p> <p>Lower 3 bits of instruction determines amount of data to be fetched by AHB manager. For field decode, see <a href="#">Table 2-5</a>.</p> <p>Note: This instruction is applicable for three stage read approach.</p>
8'h6x x=0/1/2/3/4/5	READ8n	<p>Read Data with 8 wait cycles. n=1/4/8/16/32/64</p> <p>Lower 3 bits of instruction determines amount of data to be fetched by AHB manager. For field decode, see <a href="#">Table 2-5</a>.</p> <p>Note: This instruction is applicable for direct read approach.</p>

Instruction	Instruction Name	Description
8'h6x x=8/9/A/B/C/D	READ16n	<p>Read data with 16 Wait Cycles. n=1/4/8/16/32/64</p> <p>Lower 3 bits of instruction determines amount of data to be fetched by AHB manager. For field decode, see <a href="#">Table 2-5</a>.</p> <p>Note: This instruction is applicable for direct read approach.</p>
8'h7x x=0/1/2/3/4/5	READ24n	<p>Read data with 24 Wait Cycles. n=1/4/8/16/32/64</p> <p>Lower 3 bits of instruction determines amount of data to be fetched by AHB manager. For field decode, see <a href="#">Table 2-5</a>.</p> <p>Note: This instruction is applicable for direct read approach.</p>
8'h7x x=8/9/A/B/C/D	READ32n	<p>Read data with 32 Wait Cycles. n=1/4/8/16/32/64</p> <p>Lower 3 bits of instruction determines amount of data to be fetched by AHB manager. For field decode, see <a href="#">Table 2-5</a>.</p> <p>Note: This instruction is applicable for direct read approach.</p>
8'h40	READSTATUS0	<p>Read Data ready instruction without any wait cycles.</p> <p>Note: This instruction is applicable for three stage read approach.</p>
8'h48	READSTATUS1	<p>Read Data ready instruction with 1 wait cycle.</p> <p>Note: This instruction is applicable for three stage read approach.</p>
8'h50	READSTATUS2	<p>Read data ready instruction with 2 wait cycles.</p> <p>Note: This instruction is applicable for three stage read approach.</p>

## Direct Read Approach

In direct read approach, only single instruction is used to fetch the data using AHB interface. After DWC\_ssi receives a command with more than or equal to 8 wait cycles (see [Table 2-8](#) for details), DWC\_ssi initiates a command on AHB interface to fetch the required amount of data. After the data is received from AHB interface, it is transmitted to SPI interface after required amount of wait cycles.

[Figure 2-109](#) describes the waveform of the direct read transfer.

**Figure 2-109 Direct Read Data Instruction**

- N= 0 for Standard SPI mode
- N = 1 for Dual SPI Mode
- N = 3 for Quad SPI mode
- N = 7 for Octal SPI mode

If DWC\_ssi is unable to fetch the data during the specific wait cycles, then DWC\_ssi indicates this by setting TXE bit in the Status Register (SR) and Transmit FIFO underflow interrupt signal is asserted.

### Three Stage Read Approach

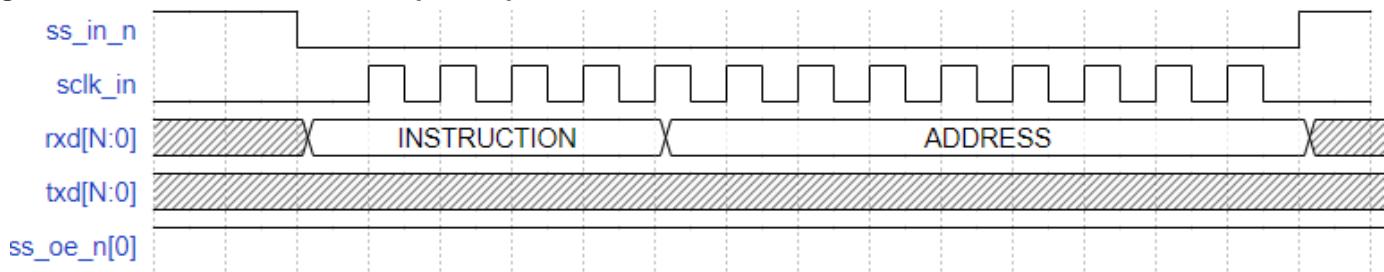
When DWC\_ssi is IDLE (The receive/transmit FIFO is empty), and READREQn instruction is received, it is treated as read request, and it starts fetching the data from AHB manager interface. After the data is available, the READxxx instruction returns the read data to SPI controller. The SPI controller ensures that the data is available before issuing another READxx instruction by polling for the data availability.

The AHB interface may receive an ERROR response for the read transfer, then the READSTATUS instruction indicates it in the response, and SPI controller takes the required actions.

Following are the waveforms for all read instructions:

### READREQx: Read Request Instruction

Read transaction is always performed in the format specified by `spi_mode_s1v` input pin. This instruction does not require any data transfer, SPI target expects only instruction and address.

**Figure 2-110 READREQx Read Request Operation**

- N= 0 for standard SPI mode

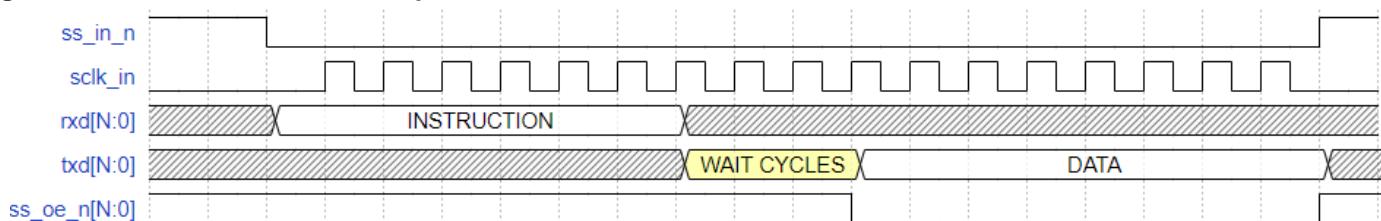
- N = 1 for Dual SPI mode
- N = 3 for Quad SPI mode
- N = 7 for Octal SPI mode

### READxx: Read Data Instruction

After the data is available, READxx instruction is used to read back the data from DWC\_ssi. This time DWC\_ssi does not expect any address to be transmitted by the SPI controller. After the instruction is received, DWC\_ssi I/Os change direction and revert with the data after some wait cycles. Wait cycles can be sent by SPI target depending on the instruction (Table 2-8).

Figure 2-111 shows the read data operation using READxx instruction.

**Figure 2-111 READxx Read Data Operation**



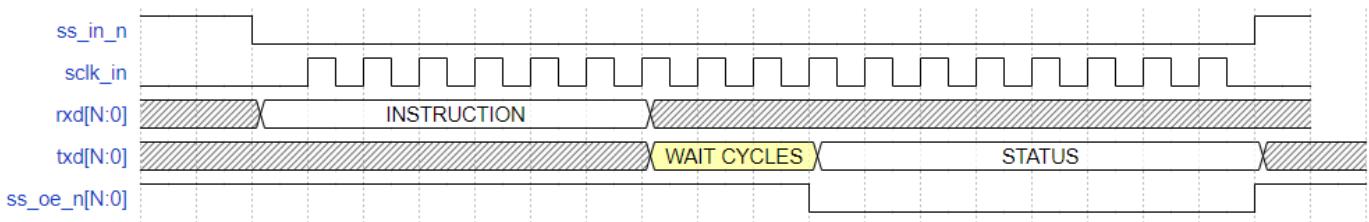
**Note** The I/O transition for the external Controller occurs during wait cycles.

### READSTATUSx: Read Status Instruction

Read status instruction is used while DWC\_ssi is fetching the data from AHB interface. SPI controller should always poll for data before performing read data operation, to ensure that the data is available for the read operation. DWC\_ssi responds with READSTATUSx command with 8-bit status. The MSB of the status bit indicates whether the data is ready or not.

- Response 8'h00 indicates that DWC\_ssi is not ready with the data
- Response 8'h80 indicates that DWC\_ssi is ready with the data

**Figure 2-112 READSTATUSx Command**



**Note** The I/O transition for the external Controller device happens during wait cycles.

After SPI controller receives READY response from DWC\_ssi, it can perform the read data operation to read the data from DWC\_ssi. The AHB interface may get ERROR response for the read transfer, in that situation

the READSTATUSx instruction indicates the same by sending a response of 8'hC0. SPI controller can then issue a new READ transfer or retry the same operation.



**Note** If SSIC\_ENH\_CLK\_RATIO parameter is set to 1, then status read or read data operation should be performed using at least one wait cycle to allow DWC\_ssi serial target device to change the I/O directions.

**Table 2-9 Response Field Decode**

Bit	Name	Description
7	READY	Indicate whether the data is ready to be fetched from DWC_ssi. ■ 0: Not Ready ■ 1: Ready
6	RDE	Indicates whether AHB transaction ended with an ERROR response or not. ■ 0: No Error ■ 1: Error
[5:0]	Reserved	Reserved and driven to 0 always.

## 2.26.5 External SPI Controller Expectations

The external SPI controller should adhere to the flow specified in write and read operations. Any out of order operations by SPI controller can move DWC\_ssi into an undefined state. If SPI controller behaves abnormally, then `ssi_spime_intr` interrupt is sent by DWC\_ssi. In such situations, reset DWC\_ssi and restart the new transfer.

Following conditions may lead to an error state and SPIMER bit in raw interrupt status (RISR) register is set for notification.

- SPI controller requests for read/write transfer when DWC\_ssi is busy with AHB transfer.  
For example,
  - a read is requested by the SPI controller and is followed by write data transfer while read transfer is still ongoing on AHB interface.
  - a write transfer is pending on AHB interface but another read/write data transfer is received by SPI target.
- SPI controller reads less amount of data than requested during the read request phase.
- If the SPI controller has requested for 16 data frames but it reads only 12 data frames, then SPIMER bit in raw interrupt status register is set and the rest of the data is flushed from the FIFO by performing soft reset.
- SPI controller writes less amount of data than specified in the transfer length.
- If the SPI controller has requested for 16 data frames but it writes only 12 data frames and de-selects the target early, then SPIMER bit in raw interrupt status (RISR) register is set and any of the data left in the FIFO is flushed by performing soft reset.
- SPI controller sends more data in case of write, or SPI controller read more data than requested.

- If AHB read transfer receives an error response from the target, then no further transfer is performed for the current request, and the data in the FIFO is flushed immediately. If read request transfer receives an error response on AHB interface, DWC\_ssi does not expect read data transfer after that because there is no read data available in FIFO. If AHB manager still performs a read data transfer this results in SPI controller error.
- Successful read request transfer is followed by another read request from SPI controller.
- A successful read request transfer (which does not receive an error response on AHB interface) should be followed by read data or read status transfer. If a successful read request transfer is followed by another read request from SPI controller, this results in SPI controller error.

## 2.26.6 AHB Manager Interface

AHB Manager interface is used as a secondary interface to read or write data for every SPI transfer. The AHB interface address and data width is fixed to 32. Following sections describe how the SPI transfers are converted to AHB transactions by DWC\_ssi.

### 2.26.6.1 Write Operation

When SPI target receives a write request (WRITE<sub>x</sub> command) from SPI target DWC\_ssi saves the address per the rules specified in “[Address Phase](#)” on page [145](#). After the data is received it is transmitted as a burst transfer defined by RXFBTR register. on AHB interface. This continues until the target is de-selected by the controller. Each data frame consists of 32 bit of data, which is transmitted on the AHB interface as a 32-bit transfer. The address increments automatically with each transfer.

If AHB write transfer receives and ERROR response, it is indicated by `ssi_ahbe_intr` interrupt of SSI status register. As soon as the error is received on the AHB interface, the rest of the data from SPI interface is discarded and no further transfers are performed for the current operation.

### 2.26.6.2 Read Operation

When SPI Target receives a read request (READ<sub>xx</sub> command) from SPI Controller DWC\_ssi saves the address per the rules specified in “[Address Phase](#)” on page [145](#). DWC\_ssi then waits for a burst to complete, and the data to be received completely. The data length is specified in the Data length field of instruction. For data length of 1, 4, 8, or 16, corresponding burst type is used to fetch the data. For the burst length of 32 or more, INCR burst type is used to fetch the data.

If any read transfer receives an ERROR response, DWC\_ssi terminates the rest of transfer and indicates the same in SSI status register. When SPI interface performs READSTATUS instruction for data ready status, this is indicated to SPI Controller with error response of 8'hC0.

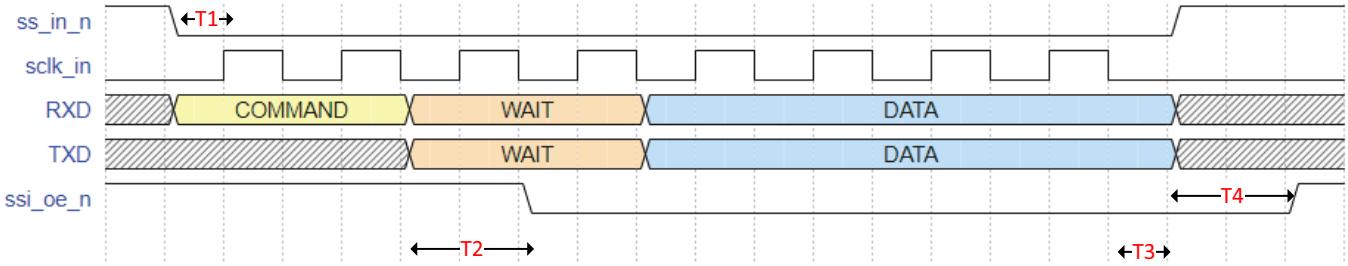


- is fixed to 3'b010 (32 bit) for all transfers.
- If AHB manager loses the grant in the middle of the read or write operation, DWC\_ssi restarts the transfer from last address.
- DWC\_ssi sends the first stores all the read data before sending it to SPI controller, so the minimum receive FIFO depth should be equal to the maximum burst length desired by the user.
- AHB Manager does not support SPLIT or RETRY responses.

## 2.26.7 SPI Interface Timings in Bridge Mode

This section describes the timing details of the SPI interface in the Bridge mode of operation.

**Figure 2-113 SPI Interface Timing Diagram**



**T1:** Period from the chip select de-assertion signal (`ss_in_n`) to the first clock edge (`sclk_in`)

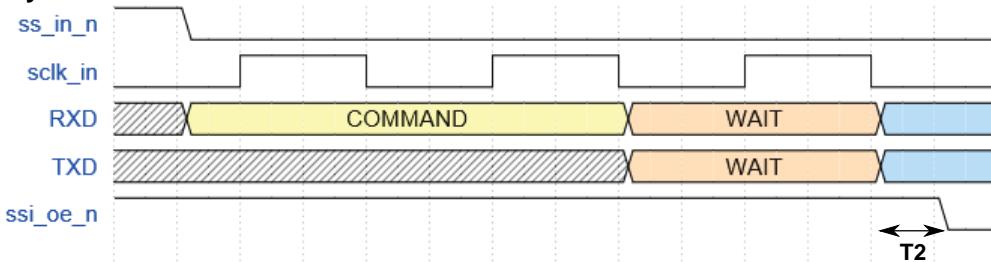
$T1 > 2 \text{ ssi\_clk cycles if } \text{SSIC\_ENH\_CLK\_RATIO} = 1$

$T1 > 4 \text{ ssi\_clk cycles if } \text{SSIC\_ENH\_CLK\_RATIO} = 0$

**T2:** Period from command phase completion to output enable (`ssi_oe_n`) assertion in case of read transactions

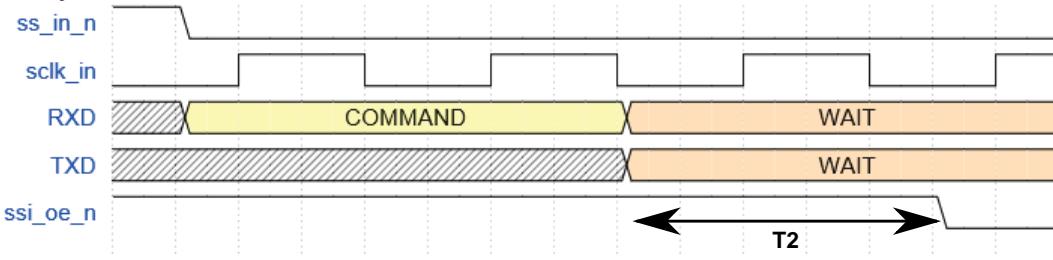
Case A: `SSIC_ENH_CLK_RATIO` parameter is set to 1

**Figure 2-114 Wait Cycle = 1**



If wait cycle is equal to 1, then  $T_{\text{ssi\_clk}} < T2 < 2 * T_{\text{ssi\_clk}}$

**Figure 2-115 Wait Cycles = 2**



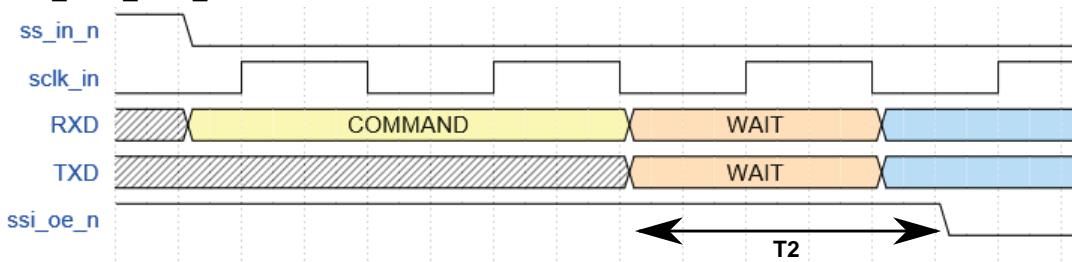
If wait cycles are equal to 2, then  $T_{\text{sclk\_in}} + T_{\text{ssi\_clk}} < T2 < T_{\text{sclk\_in}} + 2 * T_{\text{ssi\_clk}}$

Because the direct read wait cycles are 8 or more, the following equation provides a generic guidance:

$$T_{\text{sclk\_in}} * (\text{Number of wait cycles} - 1) + T_{\text{ssi\_clk}} < T2 < T_{\text{sclk\_in}} * (\text{Number of wait cycles} - 1) + 2 * T_{\text{ssi\_clk}}$$

### Case B: SSIC\_ENH\_CLK\_RATIO is set to 0

**Figure 2-116 SSIC\_ENH\_CLK\_RATIO = 0**



In this case the output enable signal asserts on the last edge of the wait cycle. If there is no wait cycle, then the output enable signal asserts after the last sampling edge of the command phase.

$$\{Tsclkin * (\text{Number of wait cycles})\} + Tssi\_clk < T2 < \{Tsclkin * (\text{Number of wait cycles})\} + 2*Tssi\_clk$$



**Note** The wait cycle starts from the last edge of the command phase, which includes both instruction and address.

### T3: Period from the last clock edge till the internal transfer done

After the read transfer is complete, internal state of the machine changes to Done in which it waits for chip select de-assertion to complete the transfer with the following constraint.

$$Tssi\_clk < T3 < 2 * Tssi\_clk$$



**Note** The last clock edge refers to the last valid edge of **sclk\_in** signal. It could be a rising edge or a falling edge depending on SCPH and SCPOL values.

### T4: Period from chip select de-assertion to output enable de-assertion

After chip select de-assertion, the output enable signal changes to the default state with the following constraint.

$$3*Tssi\_clk < T4 < 4 * Tssi\_clk$$

The internal state machine is in Done state. So, maximum cycles from the last clock edge of the transfer to the output enable de-assertion are six **ssi\_clk** cycles.

## 2.27 eXecute In Place (XIP) Mode

DWC\_ssi provides a function to directly perform memory read operation from AHB transaction. This is called execute in place mode, in which DWC\_ssi acts as memory mapped interface to an SPI memory. The XIP mode can be enabled in DWC\_ssi by selecting the configuration parameter `SSIC_XIP_EN` or `SSIC_XIP_WRITE_EN` (for write). This includes an extra sideband signal `xip_en` on an AHB interface. This signal level decides if the AHB transfers are register read-write or XIP transfers.

If `xip_en` signal is driven to 1, then DWC\_ssi expects a read request to be made on the AHB interface. This request is translated to SPI read on the serial interface. As soon as the data is received, it is returned to the AHB interface in the same transaction. The `haddr` signal is used to derive the address to be sent on the SPI interface.

### 2.27.1 XIP Read Usage Model

XIP operations are supported only in Dual, Quad, or Octal enhanced SPI modes of operation, and so, the `CTRLR0.SPI_FRF` bit must not be programmed to 0. Typically, an XIP operation consists of an address phase and a data phase. The programming flow to set-up an XIP transfer is as follows:

1. Set the SPI frame format value in `CTRLR0` register.
  2. Set the Address length, Wait cycles, and transaction type in `SPI_CTRLR0`.
- Note that, the maximum address length is 32.



If DWC\_ssi is configured for Concurrent mode, use `XIP_CTRL/XIP_SER` in place of `SPI_CTRLR0/SER` respectively.

The instruction phase can also be included in XIP transfer by using `SPI_CTRLR0` field of the `XIP_INST_EN` register. In this case, the following registers must be set:

1. Set length of instruction in the `SPI_CTRLR0` register.
2. Write the instruction opcodes in the `XIP_INCR_INST` and `XIP_WRAP_INST` registers.

After the programming is complete, you can initiate a read transaction through the AHB interface that is transferred to the SPI peripheral using programmed values.

When `SPI_CTRLR0` field of the `XIP_DFS_HC` register is set to 0, AHB control signals are used to derive the values of the data frame size and the number of data frames to be fetched by the device. `hsize` signal is used to get the value of data frame size for the transfer. [Table 2-10](#) shows the data frame size mapping from the `HSIZE` value. The data frame size can also be fixed for all the transfers (for more information on XIP Transfer, see “[Hardcode Data Frame Size for XIP Read Transfer](#)” on page [166](#).)

**Table 2-10 hsize to Data Frame Size Decode**

hsizes	Data frame Size (DFS)
3'b000	8
3'b001	16
>=3'b010	32

The number of data frames to be fetched is derived from the HBURST signal. [Table 2-11](#) shows the decode for the number of data frames decoded from the HBURST value.

**Table 2-11 HBURST to Number of Data Frames (XIP\_DFS\_HC = 0)**

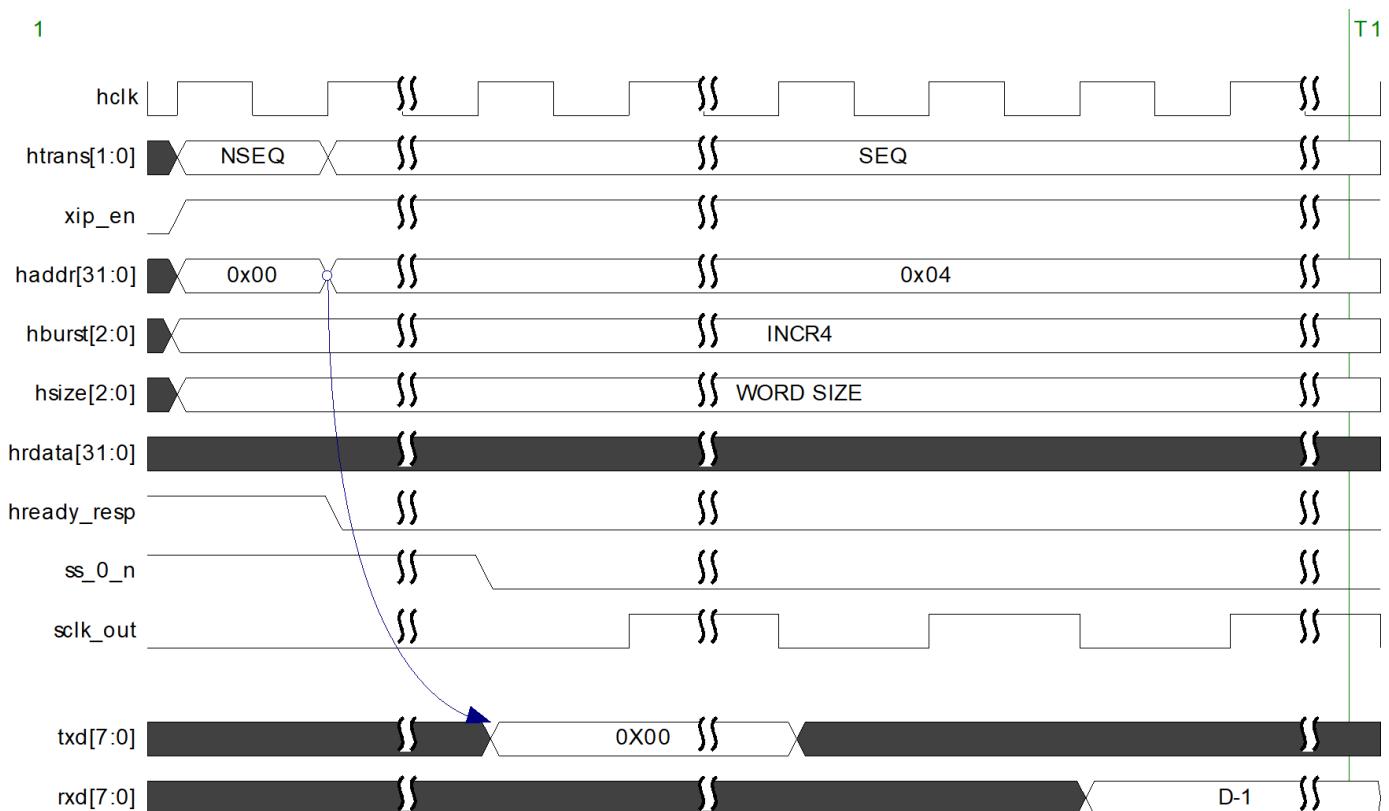
HBURST	Type	Number of Data Frames (NDF)
000	Single	1
001	INCR	Data is fetched until the burst completes
010	WRAP4	4
011	INCR4	4
100	WRAP8	8
101	INCR8	8
110	WRAP16	16
111	INCR16	16

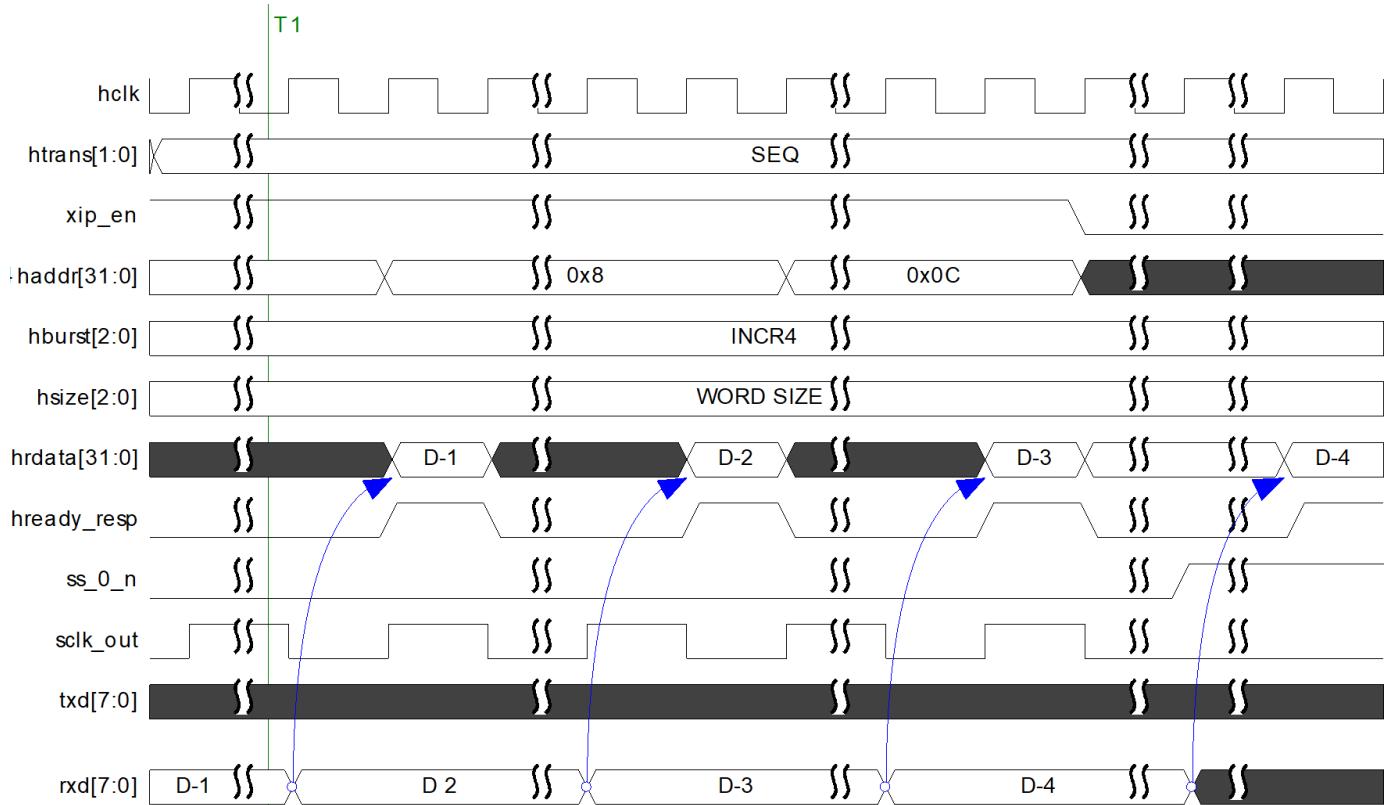
## 2.27.2 XIP Transfers

Once the CTRLR0 and SPI\_CTRLR0 registers are programmed with correct values for transfer, then XIP transfer can be initiated by driving the xip\_en signal to 1 and by performing a read operation on the AHB bus. [Figure 2-117](#) shows an example of an XIP transfer on an AHB interface. The transfer is of 4 beats starting from address 0x0. In [Figure 2-117](#), note the following:

- DWC\_ssi is working in Octal SPI mode.
- The length of data frame is derived from AHB signal hsize that is 32-bit.
- The xip\_en signal must be active for the address phase of the burst.
- The address sent on AHB SPI line is same as the address sent by the AHB interface. The number of bits sent on SPI line is defined by ADDR\_L field of the SPI\_CTRLR0 register.

DWC\_ssi first receives all the data and then transmits on the AHB interface. Once the required amount of data is fetched for a particular burst, the serial target device is de-selected. For each new burst request, address is sent again on AHB interface.

**Figure 2-117 XIP Transfer**



There are two types for AHB transfers that are handled differently by DWC\_ssi. The following sections describe the behavior of DWC\_ssi for each AHB transfer.

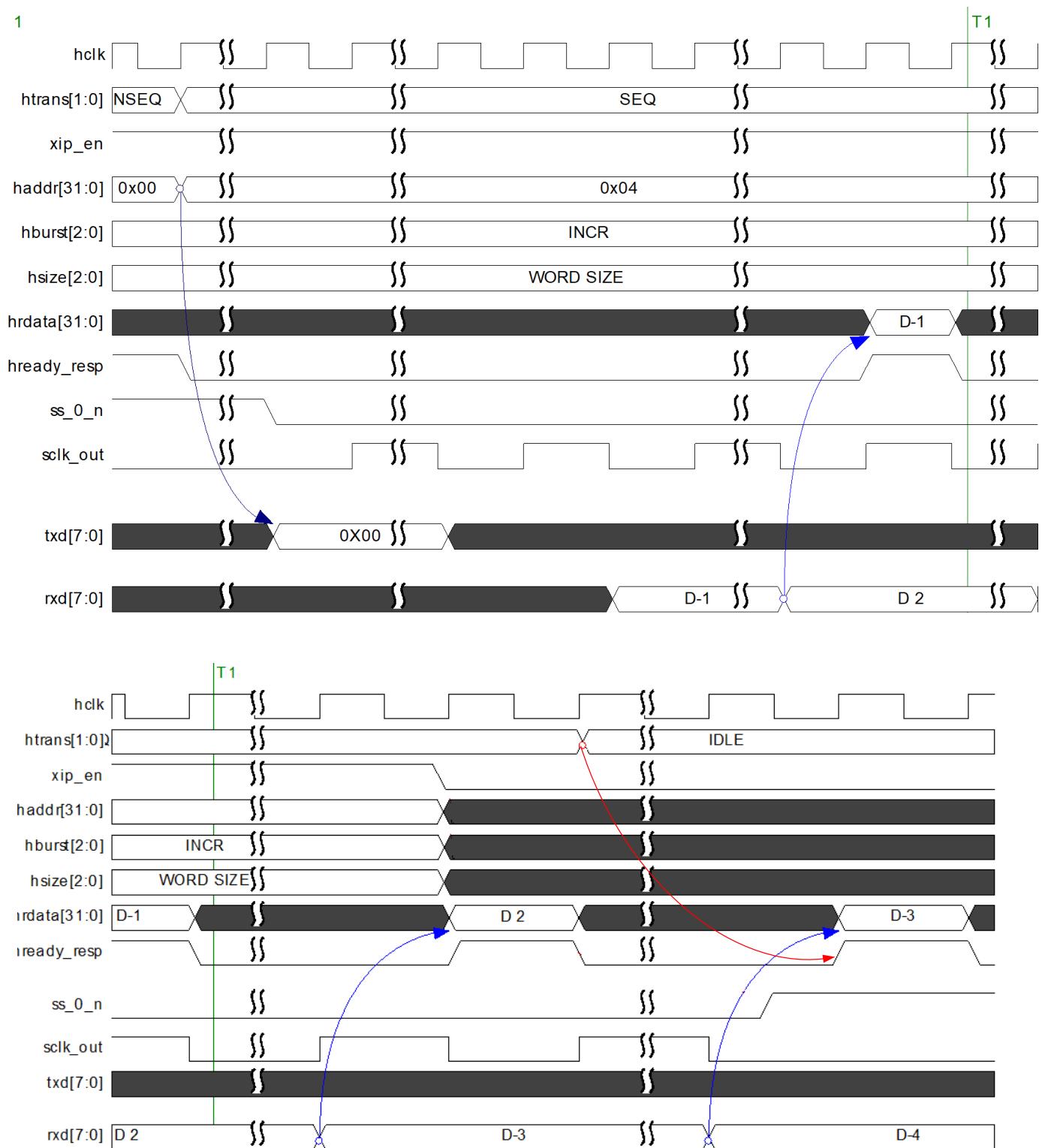
#### Case A: Fixed INCR/WRAP Burst Transfer

When DWC\_ssi receives a FIXED burst request, it fetches only fixed amount of data from SPI device. The number of data frames (NDF) field is determined by HBURST and data frame size is derived from the hsize signal. [Figure 2-117](#) shows an example of fixed incremental burst INCR4.

For the WRAP request, the device must send the correct data and DWC\_ssi forwards the data on the AHB interface.

#### Case B: Undefined Incrementing Burst (INCR)

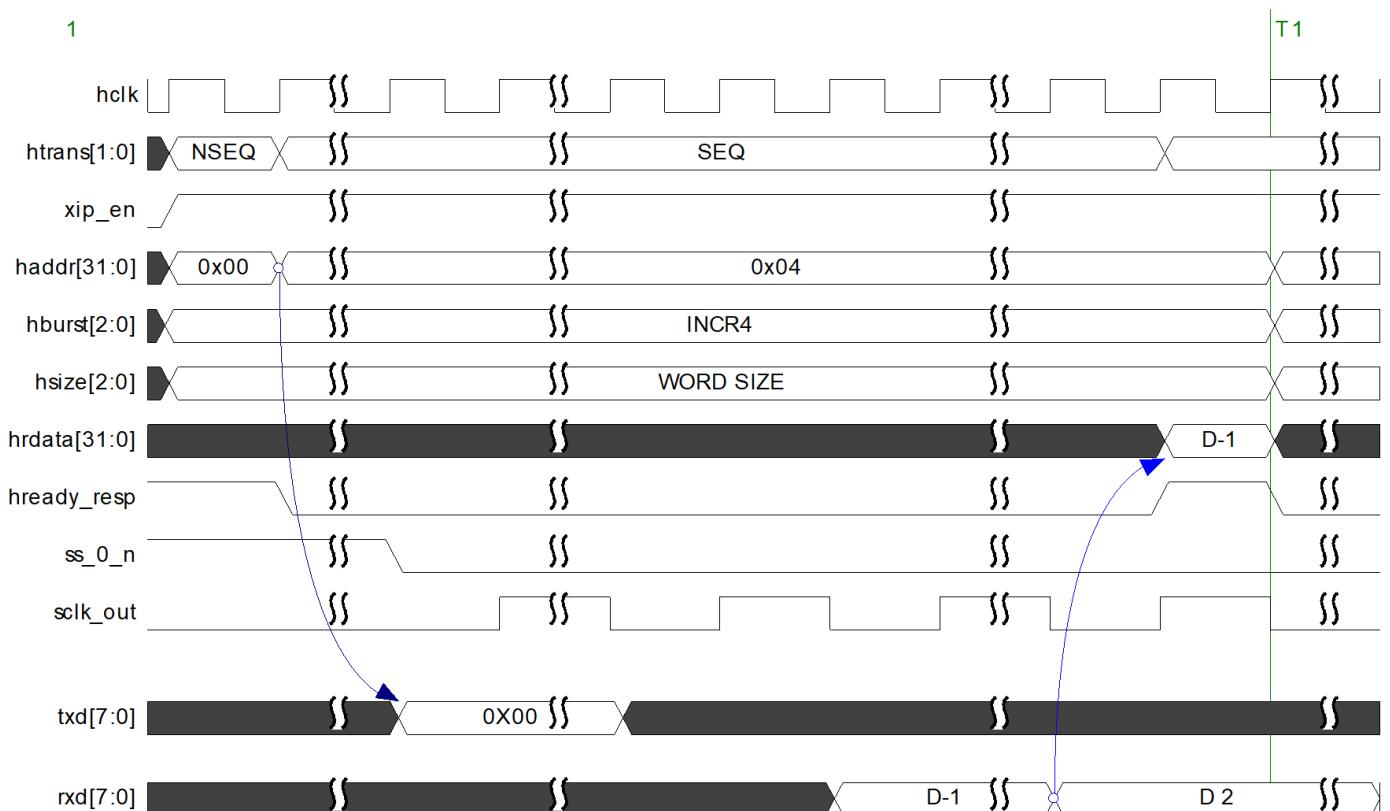
In this case, DWC\_ssi continues to fetch the data from the device unless end of burst (IDLE transfer) is detected on serial target interface. DWC\_ssi fetches a maximum of 1 KB data from SPI device for this transfer type. As shown in [Figure 2-118](#), when the HTRANS signal shows an IDLE transfer, DWC\_ssi treats it as end of INCR burst and this results in de-assertion of chip select signal by DWC\_ssi. The last data received is transmitted to the AHB manager to complete the AHB burst transfer.

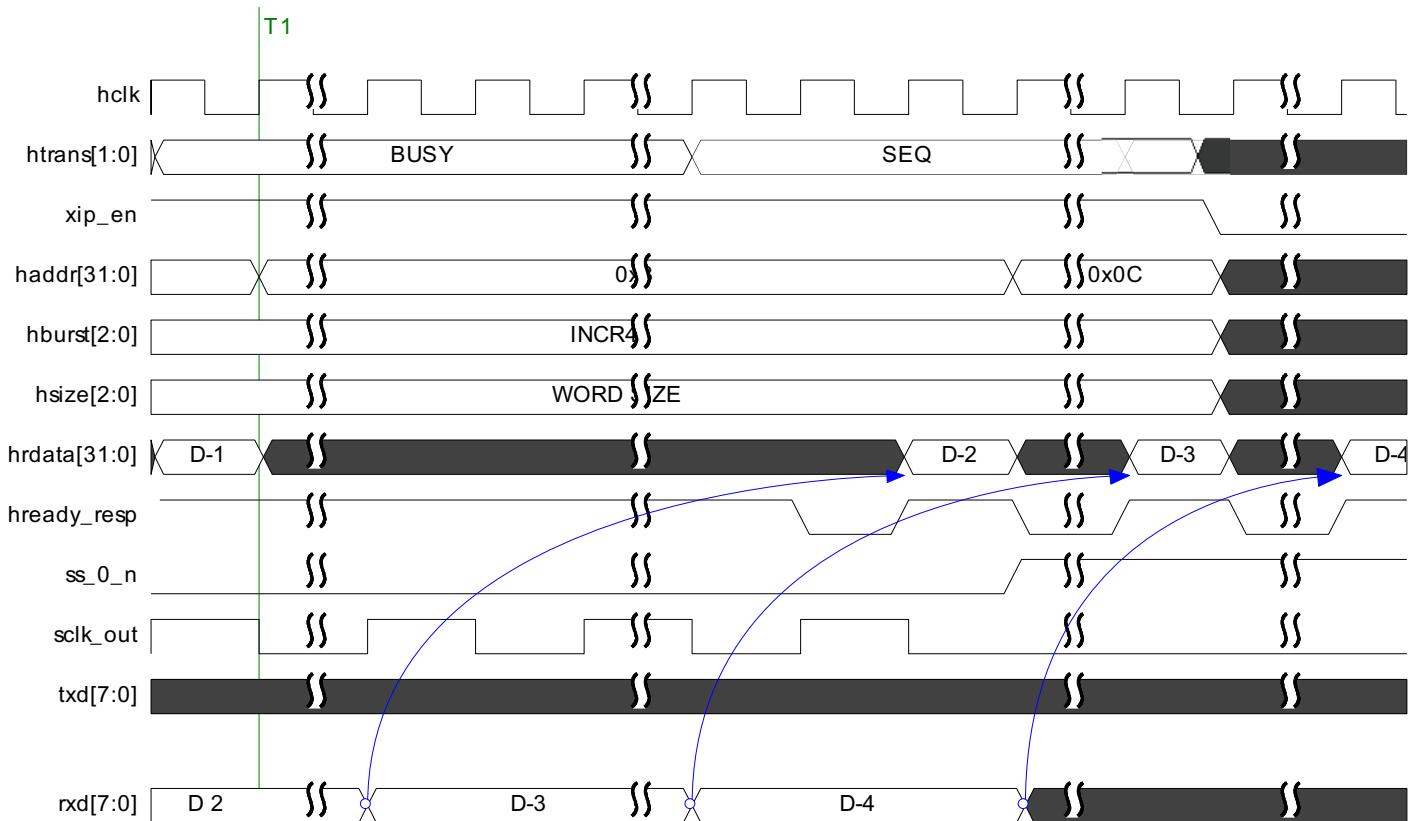
**Figure 2-118 INCR Transfer Type XIP Transfer**

### 2.27.3 AHB WAIT Transfers

During XIP transfers, AHB manager may insert wait states in the transfer. In this condition, DWC\_ssi does not break the transfer on SPI interface and continues to fetch the data from SPI target until all the required data frames are fetched for the ongoing burst. This intermediate data is stored in DWC\_ssi receive FIFO and is transmitted to AHB interface after WAIT cycles are completed.

Figure 2-119 shows an example of AHB transfer with BUSY cycles inserted by the AHB manager. After the manager receives D-1 data, AHB inserts BUSY cycles in the transfer. During this time, no data is sent to AHB manager. After AHB resumes the AHB transfer, DWC\_ssi starts sending the data received from SPI device.

**Figure 2-119 XIP Transfer with BUSY Cycles from AHB Manager**



As intermediate data frames are stored in RX FIFO, it may cause FIFO overflow condition during the transfer, for example if the FIFO depth is set to 8 and AHB interface is performing a burst transfer of 16. If AHB manager inserts wait states such that SPI interface receives 8 data frames during that time, then RX FIFO overflows and this may lead to data mismatch in the system. Therefore, it has to be taken into consideration while choosing RX FIFO depth. If RX FIFO overflow interrupt is detected by the software during XIP transfer, it must re-attempt the data transfer to avoid data corruption.

#### 2.27.4 Early Burst Termination

Early burst termination (EBT) can occur in an ongoing AHB transfer. When IDLE transfer is detected on AHB interface, DWC\_ssi gracefully completes the ongoing transfer on the SPI line (similar to INCR burst transfer in [Figure 2-118](#)), and returns the last fetched data to the AHB device. AHB manager can resume the burst from the last address and DWC\_ssi treats it as a new transfer and again transmit address to the SPI target.



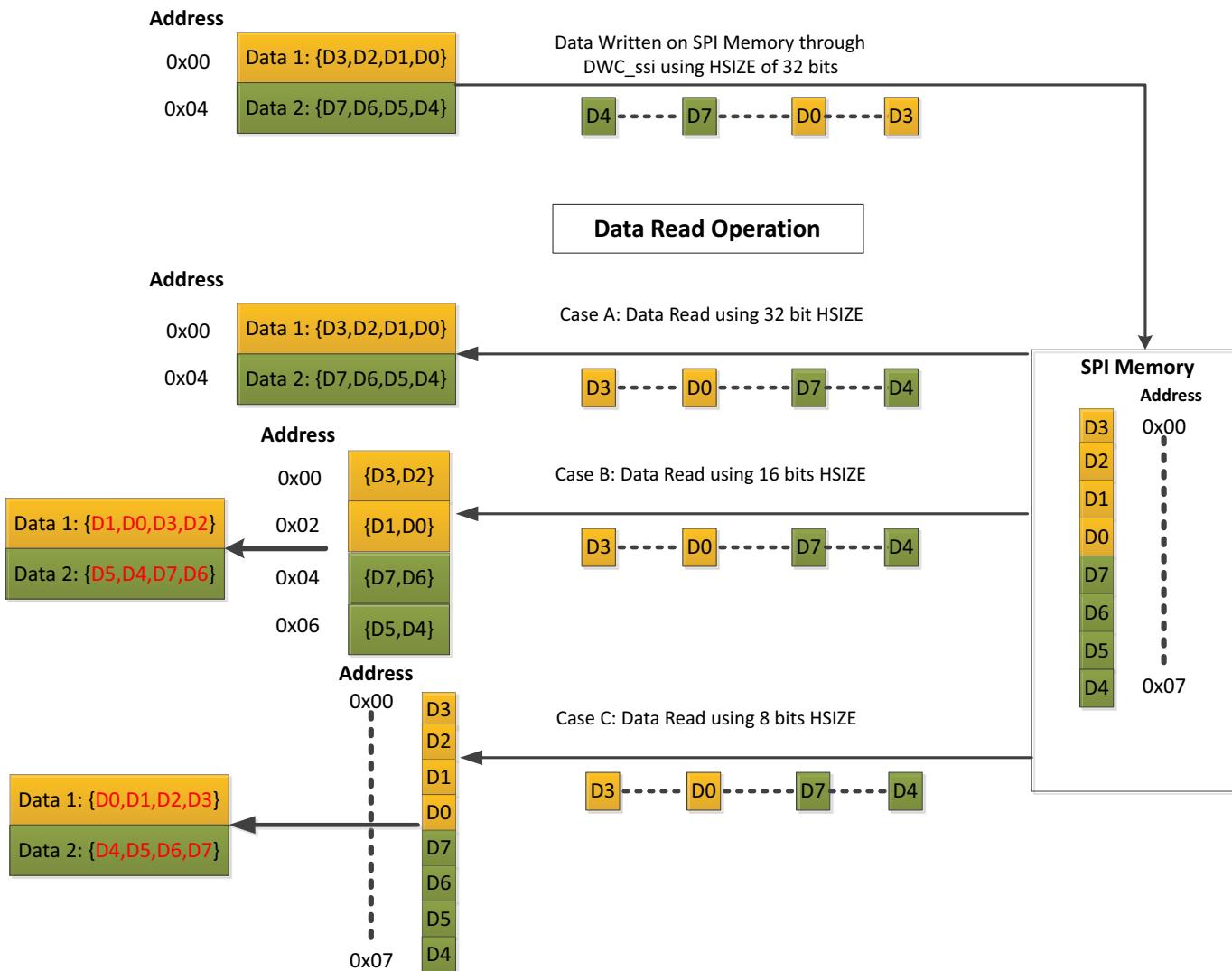
DWC\_ssi does not support early burst termination while AHB Manager is BUSY, in the XIP mode.

#### 2.27.5 Hardcode Data Frame Size for XIP Read Transfer

While reading the data from the memory, the data frame size plays a key role. The data is read using the **hsize** signal as a data frame size. This might lead to data mismatch if AHB Manager is coming up with different **hsize** values for the XIP read. For example, {D3, D2, D1, D0}, and {D7, D6, D5, D4} two data words

are written using the 32 bit data format in SPI memory and read using different data sizes (different hsize of 32 bits, 16 bits and 8 bits). [Figure 2-120](#) shows the difference in data in cases A, B, and C.

**Figure 2-120 Effect of Changing hsize on the Received Data**



As observed, the data returned is different in all the three cases. It is recommended to use hsize signal for all XIP transfers, in [Figure 2-120](#) hsize of 32 for all transfers. If you use different hsize values to read the data, then data-alignment issues occur at AHB interface.

Depending on the addressing scheme of the device, the same issue can be observed if you try to read using different values in the hsize.

- If the device is byte-addressable, then hsize of 8 bits should be used for all XIP reads.
- If the device is half-word-addressable, then hsize of 16 bits should be used for all XIP reads.
- If the device is word-addressable, then hsize of 32 bits should be used for all XIP reads.

If the AHB device cannot fix the hsize value, then DWC\_ssi provides a programmable bit (`SPI_CTRLR0.XIP_DFS_HC`), which can be used to fix the data frame size for all XIP transfers. If `SPI_CTRLR0.XIP_DFS_HC` is set to 1 then DWC\_ssi uses the value programmed in `CTRLR0.DFS` register to fetch the data.

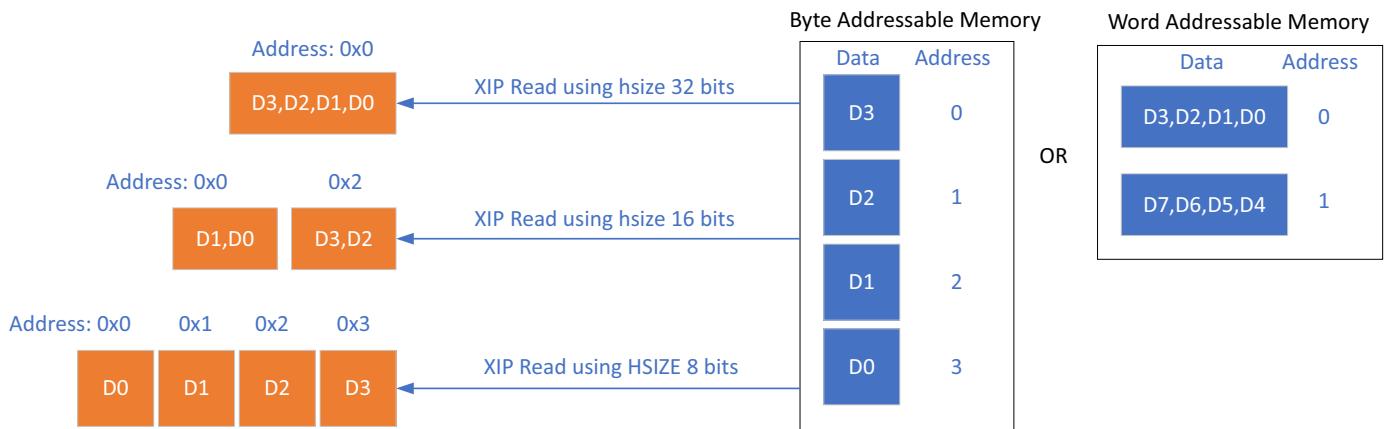
`hsize` and `hburst` signals are used to get the number of data frames to be fetched by the device, based on programmed DFS value.

```
Number of data frames (NDF)=  
HSIZEhsize*(AHB burst length))/(programmed data frame size)
```

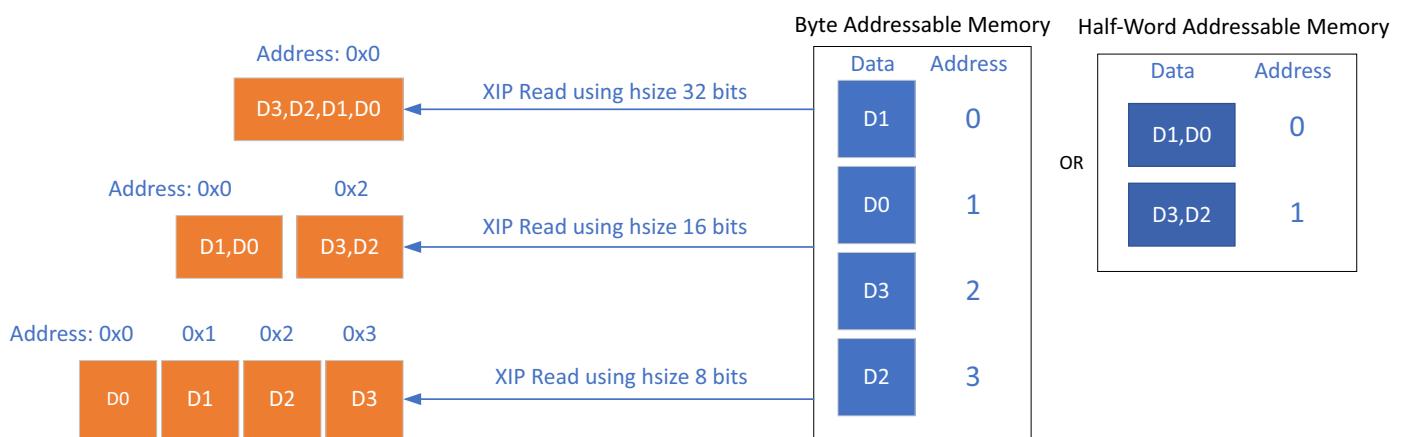
The fetched data is always on same data lane irrespective of `hsize` value on the AHB interface.

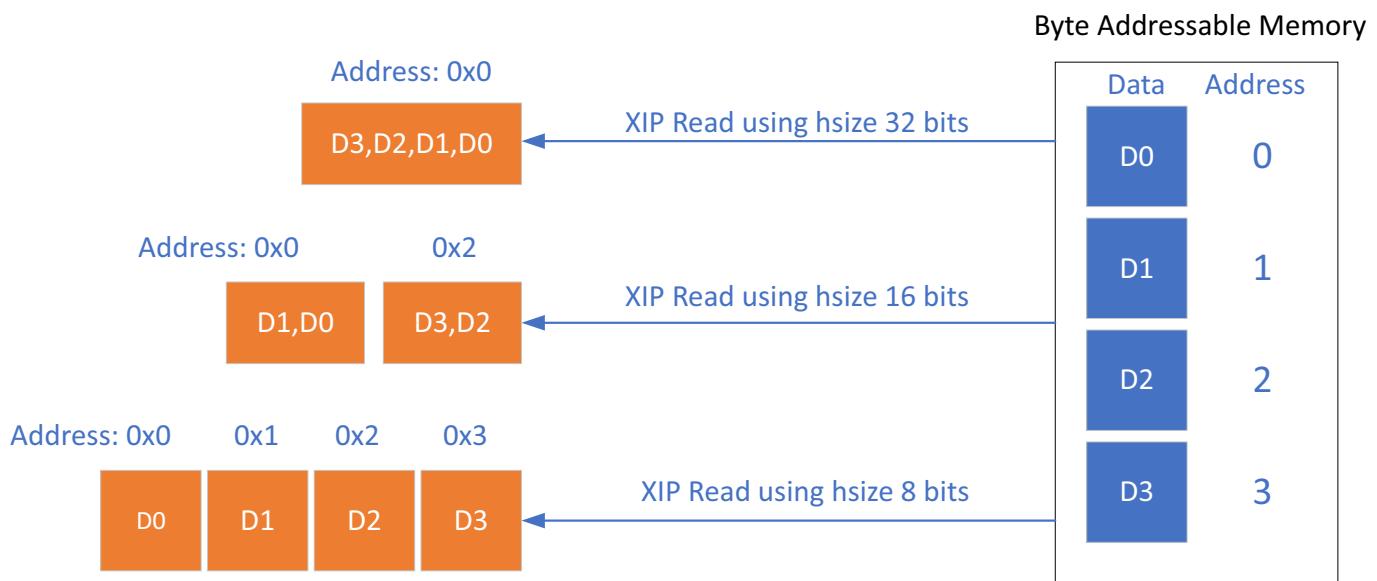
Choose the hardcoded DFS value based on how the memory is organized in the SPI memory. [Figure 2-121](#), [Figure 2-122](#), [Figure 2-123](#) show the hardcoded DFS values with the different memory organization in the SPI memories.

**Figure 2-121 Memory Organization for Hardcoded DFS Value = 32 bits**



**Figure 2-122 Memory Organization for Hardcoded DFS Value = 16-bits**



**Figure 2-123 Memory Organization for Hardcoded DFS Value = 8-bits**

If the `XIP_DFS_HC` field of the `SPI_CTRLR0` register is programmed to 1, then AHB address should always aligns with the programmed data frame size value (`CTRLR0.DFS`) for continuous or prefetch XIP transfers.

## 2.27.6 Mode Bits Support

During a serial transfer, certain devices have the capability to send additional bits that are referred to as Mode bits. In DWC\_ssi, the mode bit support can be enabled by setting the `SPI_CTRLR0` field of the `XIP_MD_BIT_EN` register. The actual value of the mode bits to be driven can be programmed by writing into the `XIP_MODE_BITS` register. The length of the mode bits can be programmed using `SPI_CTRLR0` field of the `XIP_MBL` register or `XIP_CTRL` field of the `XIP_MBL` register. Mode bits occur immediately after the address phase is completed and follow the same rules as for the address bits. Therefore, serial transfers consist of following phases when mode bit support is enabled:

Instruction Phase > Address Phase > Mode Bits > Wait cycles > Data Phase

Based on the `CTRLR0.SPI_FRF` value, mode bits are driven as follows:

- If `SPI_FRF` is Dual, the Mode bits are sent on two lanes.
- If `SPI_FRF` is Quad, the Mode bits are sent on four lanes.
- If `SPI_FRF` is Octal, the Mode bits are sent on eight lanes.



This support is available only when XIP support is enabled.

## 2.27.7 XIP Write Model

XIP operations are supported only in Enhanced SPI modes of operation (Dual, Quad and Octal modes). DWC\_ssi can write into the SPI memory without software overhead of writing the instruction and address separately inside the transmit buffer. XIP for write feature executes the write transactions directly without software intervention. DWC\_ssi supports this feature using the `xip_en` signal. Assert this signal while performing write into DWC\_ssi. After DWC\_ssi detects such transfer it inserts the address and instruction (`XIP_WRITE_WRAP_INST` or `XIP_WRITE_INCR_INST` when `SSIC_XIP_WRITE_REG_EN` is enabled and `XIP_WRAP_INST` or `XIP_INCR_INST` when `SSIC_XIP_WRITE_REG_EN` is disabled) into the transfer and initiates a write transaction in SPI interface.



If DWC\_ssi is configured for concurrent mode, use `XIP_CTRL/XIP_SER` in place of `SPI_CTRLR0/SER` respectively.

This feature has the following restrictions:

- It is available only for enhanced (Dual/Quad/Octal) SPI transfers.
- The Data Mask feature is not supported.
- The Clock Stretching feature is not supported.
- DWC\_ssi does not check for device status once the write transfer is performed. It is the software's responsibility to make sure that the device is ready to accept write transfers.
- You cannot use the XIP Write feature when XIP read pre-fetch or continuous transfers are enabled. You need to set these corresponding programming bits to 0 before performing XIP write.

### 2.27.7.1 Transfer End Conditions

DWC\_ssi ends the transfer on the SPI interface under the following conditions:

- If IDLE transfer is detected in-between an XIP write transfer.
- Transmit FIFO goes empty.

`ssi_txu_intr` interrupt is raised when the transmit FIFO goes empty. `ssi_txu_intr` interrupt is an underflow interrupt, which is used to indicate the empty state of the transmit FIFO.

For EBT conditions, the data bits sent until the target is deselected is considered valid, and are transferred to SPI side. The controller needs to resume this transfer as a new transfer including the address and instruction phase.



Before issuing a write XIP transfer on AHB interface, ensure that the following conditions are met:

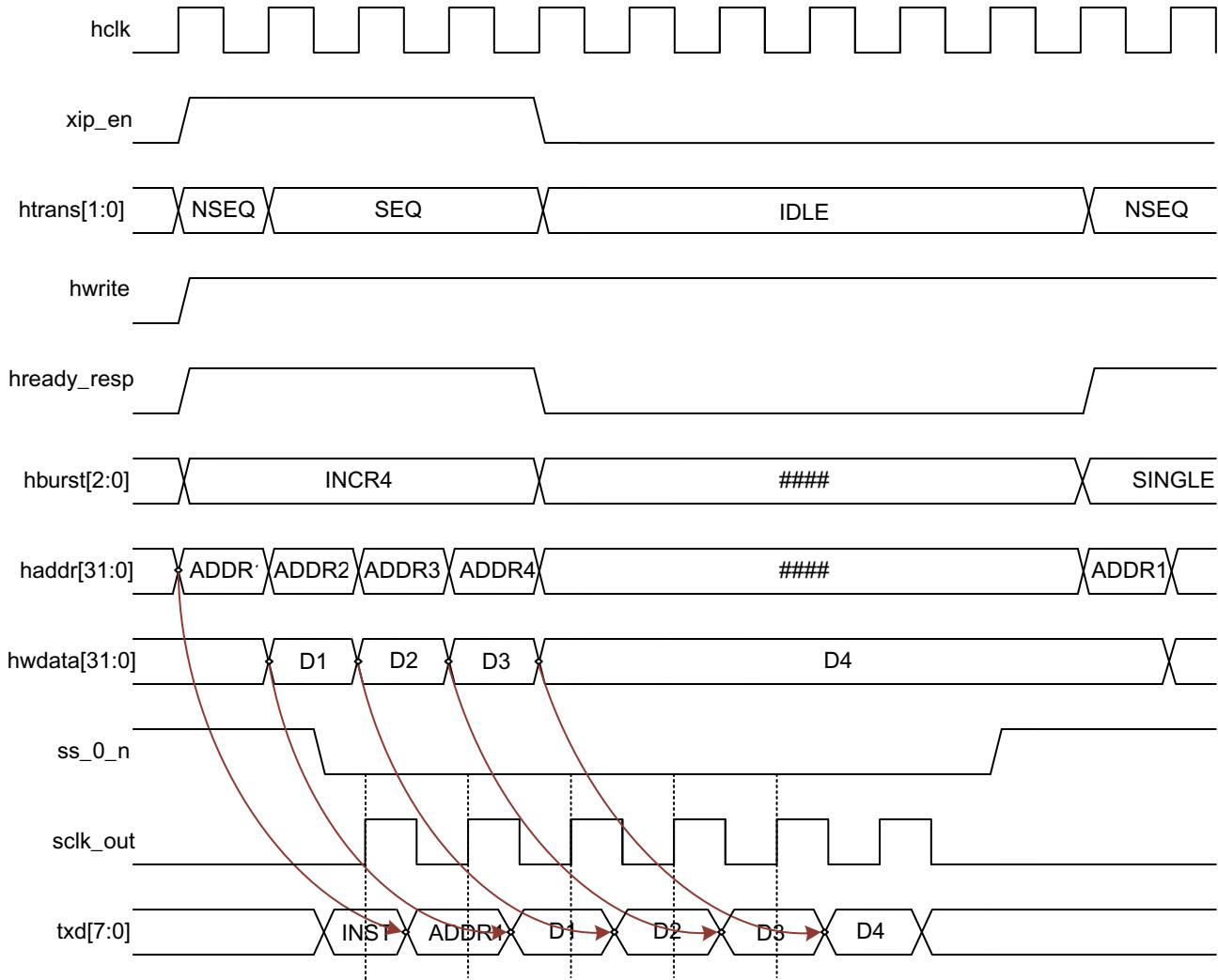
- `SSIENR` is enabled.
- Tx FIFO is empty.
- Tx underflow interrupt is cleared.

If any of these conditions are not met, DWC\_ssi issues an error response for write XIP transfer.

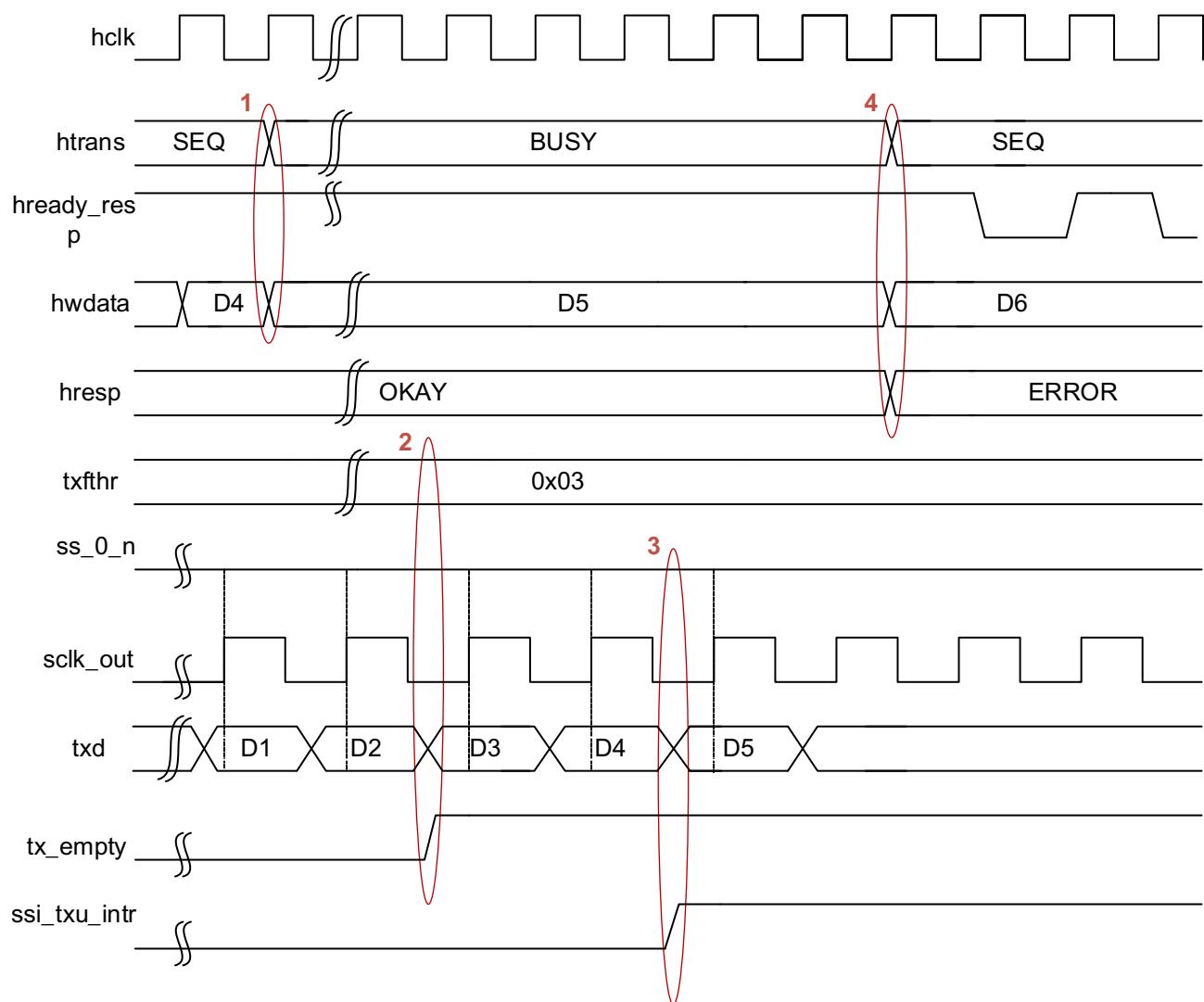
### 2.27.7.2 XIP Write Transfers

A typical XIP write transfer is shown in [Figure 2-124](#)

**Figure 2-124 XIP Write Transfer**



Every AHB transfer is treated as a separate XIP write request. Until the last XIP request completes, DWC\_ssi holds hready\_resp signal to LOW state. When AHB Manager makes a write request using AHB subordinate interface, the data is pushed into the internal transmit FIFO along with programmed instruction and address. This data is then sent to SPI interface. During the XIP burst, if the AHB Manager is unable to keep up with the data demand of the SPI interface, the transmit FIFO goes empty, and the SPI transfer ends. Under this condition, DWC\_ssi asserts the transmit underflow interrupt `ssi_txu_intr(_n)` to indicate the unsuccessful transfer completion to the software, as shown in [Figure 2-125](#).

**Figure 2-125 XIP Write When Transmit FIFO Goes Empty**

The TxFIFO empty scenario in Figure 2-125 is explained as:

- For an INCR transaction, the **htrans** signal goes to **BUSY** state when the fifth data bit is available at the **hwdata**. TxFIFO continues to push the data to SPI interface until it becomes empty.
- The **TXFTHR** (Transmit FIFO Threshold Level) register is programmed to **0x03** before starting the transfer. **tx\_empty** interrupt is set when the FIFO level is less than or equal to the programmed threshold.
- ssi\_txu\_intr** interrupt is triggered when the FIFO level goes to 0. Since **xip\_en** signal is HIGH during this instance, and the data phase is not completed by the AHB side, an error response is given through the **hresp** signal.
- After some cycles, if AHB tries to resume the transfer, DWC\_SSI discards the data bits. A new transfer can be initiated by clearing the underflow interrupt.

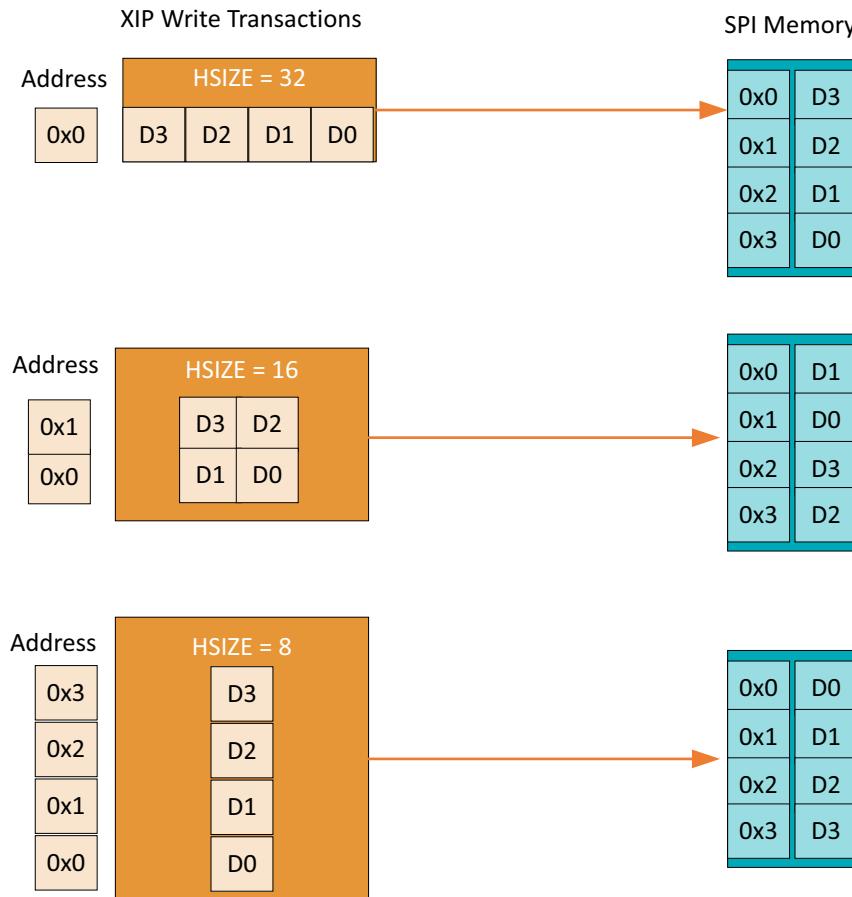


- If the faster AHB data speed and slower SPI transfer speed resulting in the Transmit FIFO full condition, then DWC\_ssi drives `hready_resp` to LOW state in the middle of the burst to prevent the overflow.
- In DDR mode, `hsize` cannot be 8 (`hsize = 3'b000`) for Octal SPI frame format.
- This feature is available only for enhanced (Dual/Quad/Octal) SPI transfers.
- The clock stretching feature is not supported for XIP Write transfers.
- DWC\_ssi does not check for the device status once the write transfer is being performed. It is the software's responsibility to make sure that the device is ready to accept a write transfer.

The XIP Write feature can not be used when XIP read pre-fetch or continuous transfers are enabled. It is required to set the corresponding programming bits to 0 before performing XIP write.

## 2.27.8 Hardcode DFS Support for XIP Write

DWC\_ssi sends the data to the memory in terms of data frame size, which is derived from the `HSIZE` signal value for XIP write operation. The software might not be able to keep `HSIZE` signal value constant for all the transfers, as a result, the data shifts in memory with different data frame size values. This might lead to data endianness issue while reading. [Figure 2-126](#) shows the effect of writing the data into a byte addressable memory using different `HSIZE` signal values.

**Figure 2-126 Effect of XIP Write Using Different HSIZE Values**

If you cannot fix the `HSIZE` signal value, then `DWC_ssi` provides a programmable bit to hardcode the data frame size for all XIP write operation irrespective of the `HSIZE` signal values. The hardcoded feature can be enabled using following programming bits:

- `SPI_CTRLR0.XIP_HC_DFS`
- If `SSIC_CONCURRENT_XIP` is set to 1, then `XIP_CTRL.DFS_HC` is used to enable this feature
- If `SSIC_XIP_WRITE_REG_EN` is set to 1, then `XIP_WRITE_CTRL.XIPWR_DFS_HC` is used to enable this feature

The value of DFS is always taken from the `CTRLR0.DFS` field. Using the hardcoded data frame size value, the data will be arranged by the IP such that it is written in the same pattern irrespective of the `HSIZE` value. The recommended value of the DFS should be based on the memory organization. If the memory is byte-addressable (8-bit) then the recommended value of DFS should be 8, if the memory is half-word (16-bit) addressable then the hardcoded DFS value should be 16 and similarly for word (32-bit) addressable memory the hardcoded DFS value should be 32.

The start address of the XIP write transfer should always be aligned with the programmed DFS value. For example, if the programmed DFS is 32, `hsize` can decode to 8, 16 or 32 bits but the start address on the AHB interface must be word aligned. Similarly, for a DFS of 16, the address should be half-word aligned. In case

a burst transaction is terminated early, and it is re-attempted, make sure that the rebuilt burst is attempted with a start address that is aligned to DFS.



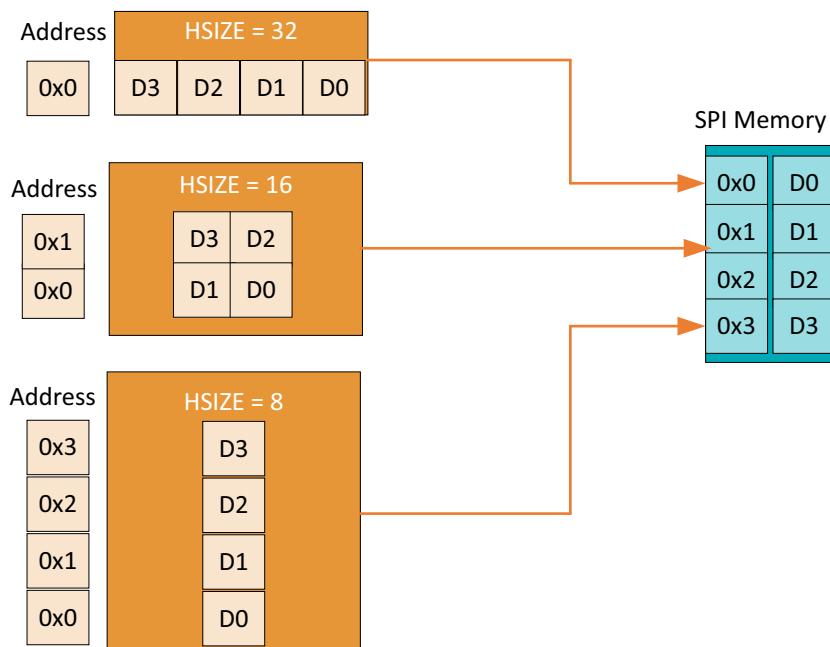
- If hardcode DFS feature is enabled, the value of `CTRLR0.DFS` can be 8/16/32 only.
- If hardcode DFS feature is enabled, the start address for the XIP Write transfer should always be aligned with programmed DFS value. In case of EBT, the start address of the re-built burst should be aligned with the programmed DFS value.

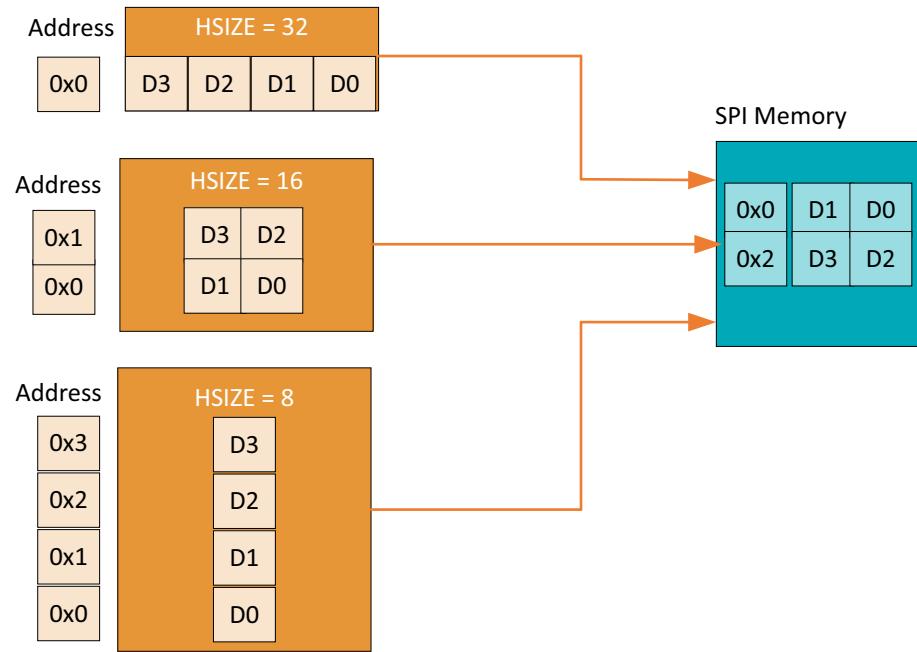
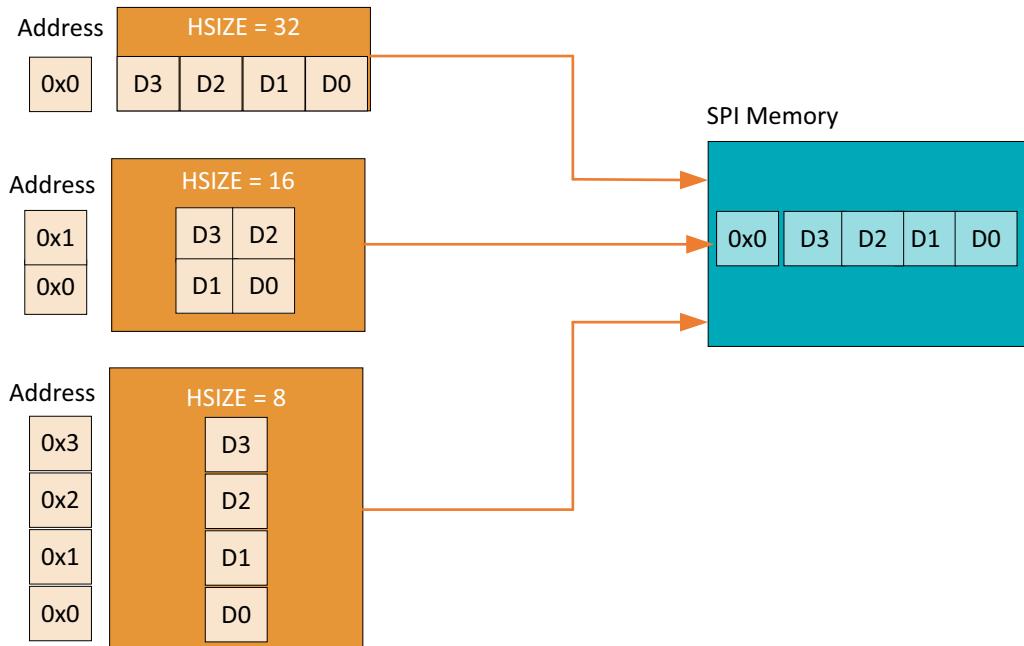
It might happen that the amount of data in the XIP transfer is not multiple of the hardcoded DFS value. For example,

- If the hardcoded data frame size is 16, AHB interface requests for a SINGLE transfer with `HSIZE = 8`. In this scenario, DWC\_ssi appends 0s to complete the data frame and push it in the FIFO.
- If there are two back-to-back write transfers with `HSIZE=8`, the IP does not combine them into one and it sends two separate transfer to the SPI device in similar fashion by appending 0s.

[Figure 2-127](#) shows the data push using different data frame size values and memory organization in the SPI memory devices.

**Figure 2-127 Data Organization When Hardcoded DFS is 8 to Byte-addressable Memory**



**Figure 2-128 Data Organization When Hardcoded DFS is 16 to Half-word Addressable Memory****Figure 2-129 Data Organization When Hardcoded DFS is 32 to Word Addressable Memory**

## 2.27.9 Data Mask Support for XIP Write

Data Mask feature enables SPI transfers to selectively provide mask for the data phase. SPI transfers makes use of the `txd_dm` signal to mask any data during the data phase. This feature is supported by XIP write transactions in octal DDR operations only.

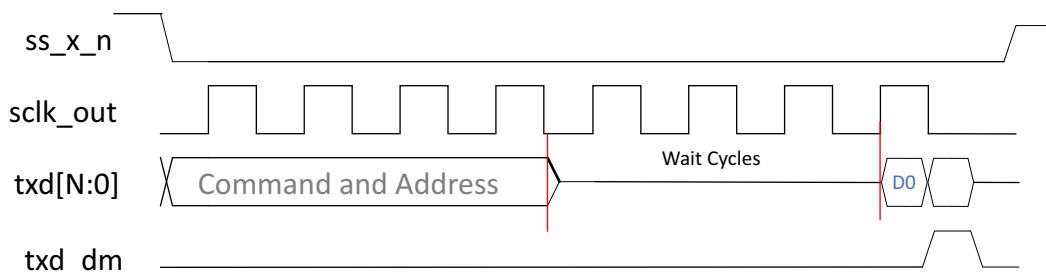
In XIP write transfers, data mask is not applicable when DFS is not hardcoded, or when the hardcoded DFS value is less than the data size indicated by hsize. Data mask is applicable when DFS is hardcoded, and the amount of the data presented in the XIP transfer is less than the programmed DFS value, which is possible in the following scenarios:

- Single transfers
- Burst transfers with odd number of beats
- DFS=32, HSIZE=8, and the number of beats in the burst is not a multiple of 4

In such cases, DWC\_ssi asserts the data mask signal to indicate to the SPI target that data is not valid for those data bits.

[Figure 2-130](#) shows an example where hardcoded data frame size is 16, and AHB interface receives a SINGLE XIP write transfer with HSIZE=8. In this scenario, DWC\_ssi appends 0s on the least significant byte to complete the data frame; and the data mask is asserted for this byte.

**Figure 2-130 Data Mask Usage Example**



## 2.27.10 External Chip Select Input /or XIP Transfers

DWC\_ssi can provide an external input signal `slave_sel` to select the target device for the transfer. This signal is sampled at the beginning of every burst request when DWC\_ssi is configured to support XIP write transfers. This enables you to select the target device without re-programming the SER register for XIP read/write operations. This pin is added on the interface when `SSIC_XIP_EXT_SSEL_EN` is set to 1.

To drive the `slave_sel` signal,

- The value of the `slave_sel` signal is expected to be stable when the AHB Burst/Transfer starts that is `htrans=NSEQ`.
- For non-XIP transfers, the Chip Select value must be programmed into the chip select Enable Register (SER). The `slave_sel` signal value is not considered for non-XIP operations.

## 2.27.11 XIP Transfers in HyperBus Mode

For XIP transfers, the 32-bit address is obtained from the HADDR bus and then packed to derive the 48-bit Command-Address (CA) frame format required by the HyperBus device. Because HyperBus memories are word (16-bit) addressable, the least significant bit of HADDR is not considered while packing the address in CA frame format. [Table 2-12](#) describes the 48-bit CA frame format.

**Table 2-12 48-Bit Data Packing for XIP Transfers**

CA Bit	Bit Name	Bit Function
47	R/W#	Set to 1 for reads and 0 for write.
46	Address Space (AS)	Fixed to 0 for memory address read.
45	Burst Type	Burst type is derived from the <code>hburst</code> signal.
44–16	Row & Upper Column Address	44 <sup>th</sup> bit set to 0, 43-16 bits are assigned to HADDR[31:4]
15–3	Reserved	Set to 0.
2–0	Lower column address	HADDR[3:1]



- Only FIXED burst types are supported in the XIP mode for HyperBus.
- Continuous burst mode (`SPI_CTRLR0.SSIC_XIP_CONT_XFER_EN = 1`) is not supported in XIP mode of operation for HyperBus.

### 2.27.12 Enabling the XIP Feature in SPI Mode

To enable the XIP feature in SPI mode in DWC\_ssi, choose the **Yes** option for the **Include XIP feature in SPI mode?** field using the **SPI Parameters** tab during the Specify Configuration Activity in coreConsultant.

To enable the XIP Write feature in XIP mode, select the **Include XIP Write feature?** checkbox using the **SPI Parameters** tab during the specify Configuration Activity in coreConsultant.

You can enable the XIP feature only when DWC\_ssi is configured to work in Dual, Quad, Octal or Dual Octal SPI mode.

### 2.27.13 Signals Related to eXecute In Place (XIP) Mode

The following signals are related to eXecute In Place (XIP) Mode:

- `xip_en`
- `slave_sel[N-1:0]`

For more information about these signals, see the *Signals Description* chapter.

### 2.27.14 Registers Related to eXecute In Place (XIP) Mode

The following are the registers related to eXecute In Place (XIP) Mode:

- `CTRLR0`
- `XIP_CTRL`
- `SPI_CTRLR0`
- `XIP_WRAP_INST`
- `XIP_INCR_INST`
- `XIP_MODE_BITS`

- XIP\_WRITE\_INCR\_INST
- XIP\_WRITE\_WRAP\_INST

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.28 Concurrent XIP and Non-XIP Operation

XIP and non-XIP transfers are mutually exclusive. Therefore, it is assumed that the receive and transmit FIFOs are empty when a new XIP or non-XIP transfer is initiated. Therefore, you must schedule both transfers sequentially to ensure that there is no overlap or data loss during any of the transfers. Due to increased complexity and throughput requirements, some of the systems may not be able to adhere with such requirement.

However, DWC\_ssi provides a separate configuration option to increase the parallelism between XIP and non-XIP transfers. DWC\_ssi has a separate FIFO for XIP data so you need not ensure that TX FIFO and RX FIFO are empty before requesting for a new XIP transfer. Thus, by enabling this feature, DWC\_ssi ensures no data corruption.

### 2.28.1 Description of Concurrent XIP and Non-XIP Operation

To enable concurrent XIP and Non-XIP operation support, set SSIC\_CONCURRENT\_XIP\_EN parameter to 1. If SSIC\_CONCURRENT\_XIP\_EN parameter is set to 1,

- You can perform concurrent non-XIP (read or write) transaction with XIP transfers. DWC\_ssi has separate transmit and receive storage for XIP transfers, which ensures no data corruption. On the serial interface, only one transaction goes through at a time. DWC\_ssi prioritizes transactions on the first-come first-serve basis, which means that if a non-XIP transfer is already in progress on the SPI interface, then XIP transfer request does not go through until the current transfer is completed and vice-versa.
- A separate XIP configuration (XIP\_CTRL) and chip select (XIP\_SER) registers are added in the register memory map. XIP transfers use the transfer characteristics from XIP\_CTRL register to define an XIP transfer. For non-XIP transfer, the existing CTRLR0 and SPI\_CTRLR0 registers determine the nature of the SPI serial transfer.

Consider a subsystem in which DWC\_ssi is connected with multiple AHB managers. Manager-A is responsible for DMA transactions; and Manager-B is responsible for XIP transfers. Both managers can request for the transfer at any point of time.

#### SSIC\_CONCURRENT\_XIP\_EN = 0

If SSIC\_CONCURRENT\_XIP\_EN parameter is set to 0 for DWC\_ssi, then controller does not handle any incoming XIP request from Manager-B while it is serving a DMA request from Manager-A. DWC\_ssi responds with ERROR status and XIP transaction is only serviced when the DMA transfer and the corresponding SPI transaction are completed. Alternatively, CPU can interrupt to allow XIP transfer using the following steps:

Whenever Manager-B requests for XIP transfer, CPU

1. Stops the ongoing DMA transfer.
2. Ensures that the TX and RX FIFOs are empty.
3. (Optional step) Issues an SPI Suspend command.
4. Ensures that the SPI device completes its internal operation (This may require polling from a status bit in device).
5. Allows XIP transfers to proceed.
6. An SPI Resume command is issued if an SPI Suspend command was issued in step 3.
7. Checks if the SPI device is ready to resume DMA transaction again. (this may require polling from a status bit in device).

8. Resumes the DMA transaction by appending command-address to the rest of the data.

In this procedure, the overall performance is affected severely because executing step 3, 4, 6, and 7 is time consuming. So, it is better to perform DMA and XIP exclusively with each other.

## **SSIC\_CONCURRENT\_XIP\_EN = 1**

If **SSIC\_CONCURRENT\_XIP\_EN** parameter is set to 1, then the steps described in the **SSIC\_CONCURRENT\_XIP\_EN = 0** section are not required. **DWC\_ssi** provides maximum throughput by applying the following programming rules:

- Set the DMA control and DMA threshold registers (**DMATDLR** and **DMARTDLR**) in **DWC\_ssi**.
- Set the subordinate number for DMA transaction in **SER** register.
- Set the subordinate number for XIP transaction in **XIP\_SER** register.
- Ensure that the DMA transaction length is equal to **TXFTLR.TXFTHR** field value.

**DWC\_ssi** waits for data frames to set in **TXFTLR.TXFTHR** register before starting new transfer on SPI interface. Set the value equal to DMA transaction length to ensure that whole DMA transaction goes through in a single attempt. So **DWC\_ssi** accumulates the data for DMA transaction in TX FIFO until the threshold level is reached. During this time, if Manager-B requests any XIP transaction, it goes through immediately. Thus, removing any CPU involvement and poling.

Once all the data required for DMA transaction is present in TX FIFO, **DWC\_ssi** initiates a write transaction to the device only if the device is not **BUSY**. This transaction does not break until the FIFO is empty. During this time, if any XIP request is made to the device, it needs to wait until the write data transfer is complete.



This behavior is true only for Adesto EcoXiP devices, which allows immediate read transaction while write transaction is in progress. For rest of the devices, CPU must ensure that the device is ready to accept the XIP transfer either by polling Write in Progress/Busy status, or issuing a Suspend command before issuing any other transaction to the device.

In this way, this feature increases the overall throughput of the design, and CPU intervention is limited to programming **DWC\_ssi** correctly.

### **2.28.2 Enabling Concurrent XIP and Non-XIP Operation**

To enable concurrent XIP and non-XIP operation for **DWC\_ssi**, choose the **Yes** option for the **Enable Concurrent XIP transfer mode?** field using the **Enhanced SPI parameters** tab during the Specify Configuration Activity in coreConsultant.

### **2.28.3 Registers Related to Concurrent XIP and Non-XIP Operation**

The following are the registers related to Concurrent XIP and Non-XIP Operation:

- **XIP\_SER**
- **XIP\_CTRL**
- **CTRLR0**

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.29 Continuous Transfer Mode in XIP

When DWC\_ssi receives an XIP request, address from the AHB interface is transmitted onto the SPI interface directly.

Each new transfer (XIP Read) on the AHB interface is treated in the same manner. Therefore, for every request, a new address must be sent to the device thereby contributing to the latency of the system.

If a memory device allows the stretching of chip select signal in between the XIP read transfers, then DWC\_ssi can be programmed for continuous XIP mode to achieve higher performance. In this mode, the host fuses two or more AHB burst requests into a single SPI command by ensuring that the command and address are not retransmitted and the host controller need not wait for any temporary cycles in between these bursts.

### 2.29.1 Description of Continuous Transfer Mode in XIP

DWC\_ssi can be enabled for continuous XIP mode by programming the `SPI_CTRLR0.XIP_CONT_XFER_EN` bit to 1. When this bit is set to 1, then DWC\_ssi functions in continuous XIP mode as soon as first XIP command is received. For the first XIP transfer, the address (and instruction if `SPI_CTRLR0.XIP_INST_EN = 1`) is sent on the SPI interface. After reception of requested data, DWC\_ssi continues to keep the target device selected and the clock (`sc1k_out`) remains in the default state. For subsequent XIP transfers on the AHB interface, DWC\_ssi resumes the clock (`sc1k_out`) and neither the command nor the address is transmitted onto the SPI interface and the data to be fetched from the device immediately (no temporary cycles).

DWC\_ssi moves out of the continuous mode due to either of the following conditions:

- A non-XIP command is received on an XIP interface (effectively any AHB transaction with `xip_en` driven to 0).
- When the AHB transaction is to a non-consecutive address, the chip select is removed and then DWC\_ssi initiates a new XIP request.



In continuous transfer mode, if `SPI_CTRLR0.XIP_DFS_HC` is set to 1 then the XIP address must always be aligned with the DFS value programmed in the `CTRLR0.DFS` bit.

DWC\_ssi supports the following continuous transfers:

- “[Continuous Incremental Transfer](#)” on page [182](#)
- “[Continuous WRAP Transfer](#)” on page [183](#)

The “[Data Sampling in Continuous Transfer Mode](#)” section describes data sampling in different scenarios for different SPI protocols.

The “[Serial Chip De-select in Continuous Transfer Mode](#)” section describes serial chip de-select option in Continuous Transfer Mode.

### 2.29.2 Continuous Incremental Transfer

A transfer is defined as continuous incremental when the first command received in the continuous XIP (`SPI_CTRLR0.SSIC_XIP_CONT_XFER_EN = 1`) mode is INCR (`hburst = 011/101/111`). In this mode, when the transfer is completed, the next command must also be incremental, and the address must be a contiguous address of the last burst.

For example, if the first received transaction has the following characteristics, then the next continuous address must start from 0x00001010:

HADDR = 0x00001000; HSIZE = 3'b010 (32 bits) and HBURST = 011 (INCR4)

If the AHB transaction (`xip_en` = 1) of any address other than 0x00001010 or a WRAP transfer is detected, then the chip select is removed, and then DWC\_ssi initiates a new XIP request.



An undefined INCR (`hburst` = 001) burst is not supported in continuous read mode.

### 2.29.3 Continuous WRAP Transfer

A transfer is defined as continuous WRAP when the first command received in continuous XIP mode is WRAP (`hburst` = 010/100/110). In this mode, when the transfer is completed, the next command must start from the address of last-burst address boundary.

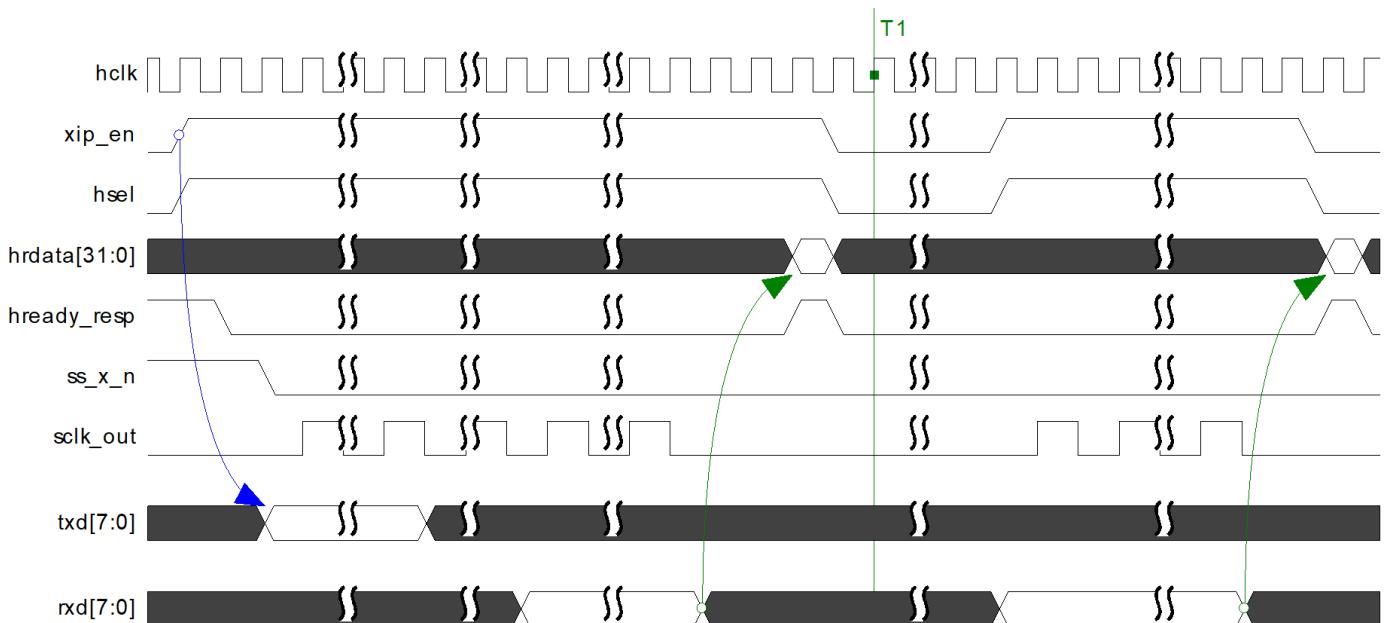
For example, if the first transaction received has the following characteristics, then the next XIP transfer address must start from 0x00001010:

HADDR = 0x0000100C; HSIZE = 3'b010 (32 bits) and HBURST = 010 (WRAP4)

If the AHB transaction (`xip_en` = 1) of any address other than 0x00001010 is detected, then the chip select is removed first, and then DWC\_ssi initiates a new XIP request.

[Figure 2-131](#) shows two back-to-back XIP transfers when SPI\_CTRLR0.SSIC\_XIP\_CONT\_XFER\_EN is set to 1. When the `ss_x_n` is selected for the first transfer, it is not de-selected even when the transfer is completed within the time-stamp T1. After the previous transfer is completed, DWC\_ssi waits for the next XIP transfer and then resumes the clock to obtain the remaining data without sending any address or instruction.

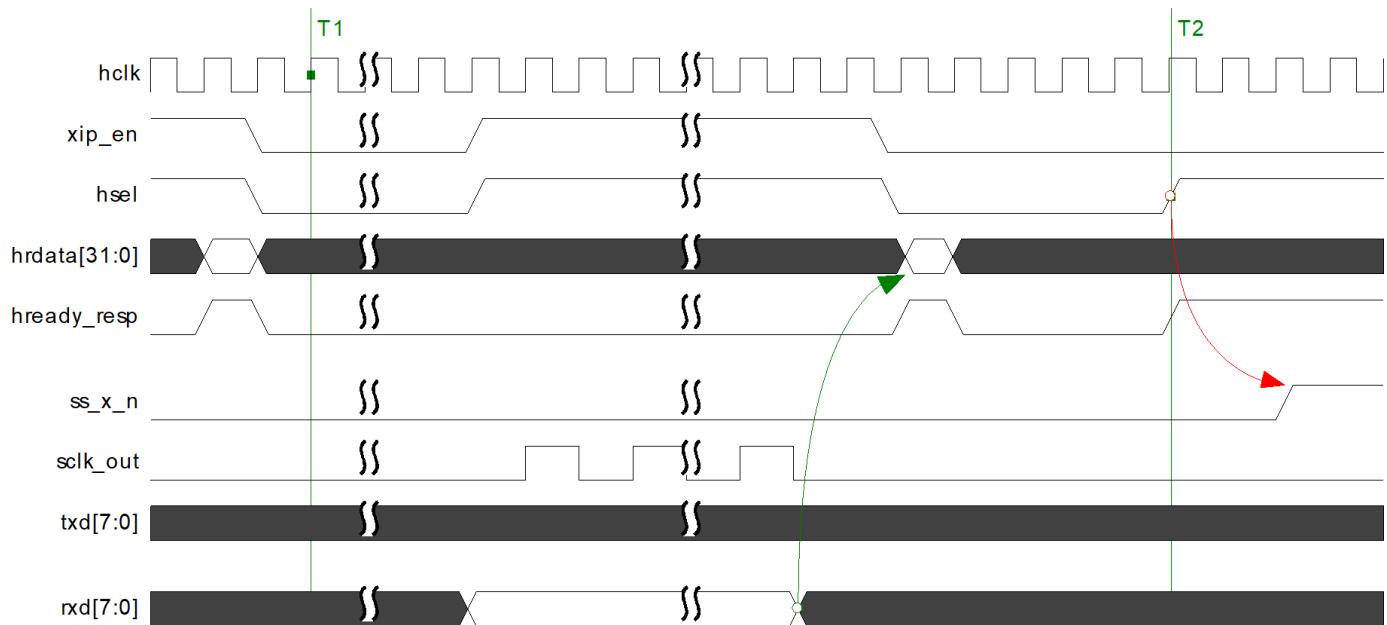
**Figure 2-131 Continuous XIP Transfer**



A continuous XIP transfer can be ended by issuing any AHB transfer with `xip_en` signal driven to 0, or when the AHB transaction (with `xip_en = 1`) is to a non-consecutive address. In this case, the chip select is removed, and then DWC\_ssi initiates a new XIP request.

In [Figure 2-132](#), the last continuous XIP transfer ends at time stamp T1 and again a new XIP transfer is detected on the line that resulted in a continuous transfer, and a chip select was retained in the same state. At time stamp T2, a new non-XIP transfer is detected that results in de-selection of the serial target device and DWC\_ssi exits from the continuous XIP mode.

**Figure 2-132 Ending a Continuous XIP transfer with a Non-XIP Transfer**



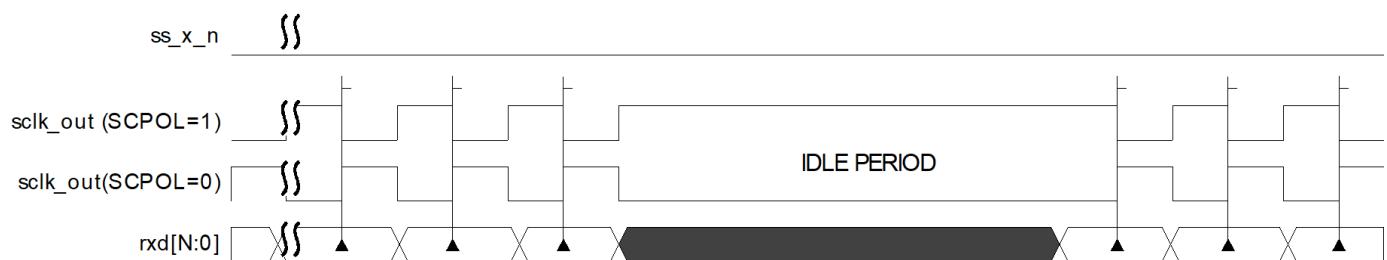
## 2.29.4 Data Sampling in Continuous Transfer Mode

The data sampling in continuous transfer mode depends on the SPI protocol type (SCPH and SCPOL values). The different cases for all the combinations are listed as follows:

Case A: SCPH = 0

In this case, when the last continuous transfer completes, the next data is sampled on the first edge of clock. The polarity of the clock during the IDLE period is dependent on the SCPOL value. [Figure 2-133](#) shows an example of a continuous transfer with an SCPH value of 0.

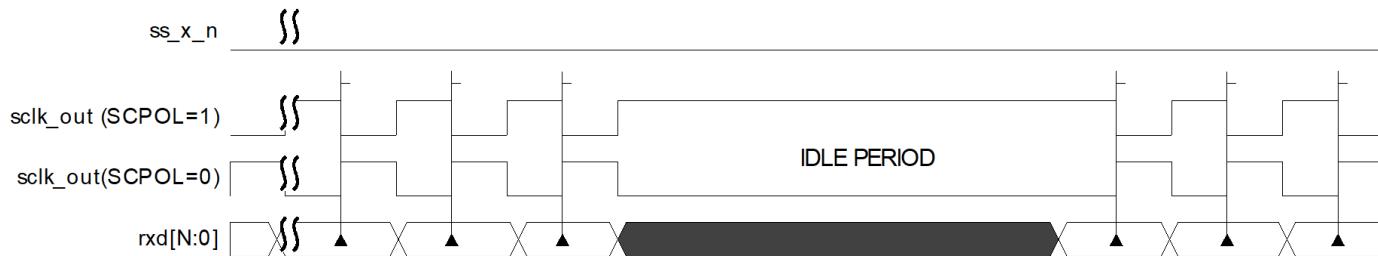
**Figure 2-133 Continuous Transfer With SCPH = 0**



Case B: SCPH = 1

In this case, when the last continuous transfer completes, the next data is driven at the first edge and sampled on the second edge of the clock. The polarity of the clock during the IDLE period is dependent on the SCPOL value. [Figure 2-134](#) shows an example of a continuous transfer with an SCPH value of 1.

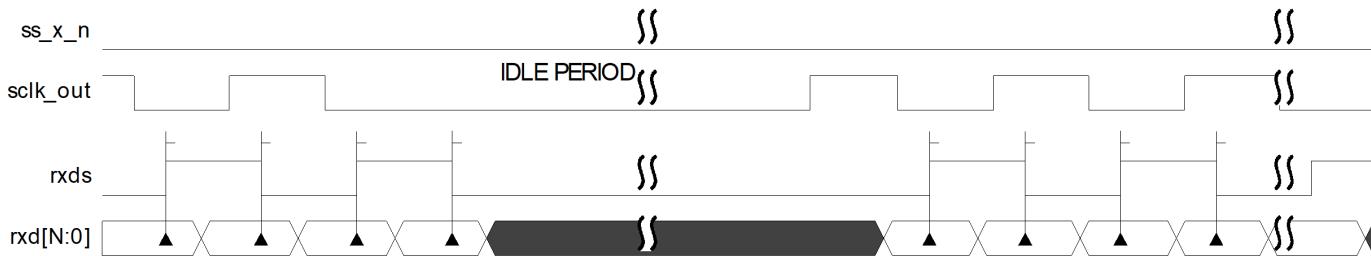
**Figure 2-134 Continuous Transfer With SCPH = 1**



For dual data-rate transfers (`SPI_CTRLR0.SPI_DDR_EN = 1`), the continuous transfer mode is supported only for  $\text{SCPH} = 0$  and  $\text{SCPOL} = 0$  configurations, and the remaining combinations are not supported.

[Figure 2-135](#) shows an example of a continuous XIP transfer in DDR mode using the RXDS signal. During the IDLE period, the RXDS signal returns to default state (LOW) and resumes the clock when DWC\_ssi resumes the transfer by toggling sclk\_out.

**Figure 2-135 Continuous Transfer in DDR Mode**



## 2.29.5 Serial Chip De-select in Continuous Transfer Mode

During the continuous transfer, a lot of power is dissipated on the serial target device since the serial target is selected all the times. To avoid such condition, DWC\_ssi provides a configuration option to enable a watchdog timer to de-select the serial target after the counter runs out.

DWC\_ssi can de-select the serial target device under the following conditions:

- A non-XIP command is received on an XIP interface (effectively any AHB transaction with `xip_en` driven to 0).
- When the AHB transaction is to a non-consecutive address, the chip select is removed and then DWC\_ssi initiates a new XIP request.
- DWC\_ssi does not detect any XIP transfer on AHB interface for the time-period specified in `XIP_CNT_TIME_OUT` register.

The counter starts its operation after the last XIP transfer is completed, and then waits for a new request on AHB interface. This counter can be enabled using the parameter `SSIC_CONT_XFER_TIMEOUT_CNT_EN`. This adds a register `XIP_CNT_TIME_OUT` in the register map of DWC\_ssi, which can be used to set the count-down value for the IDLE period. When set to 0 the counter is disabled, and DWC\_ssi continues to wait for the next XIP request without de-selecting the serial target device.

The counter runs in terms of `hclk`. After the counter reaches the value programmed in the `XIP_CNT_TIME_OUT` register, it may take some time to de-select the serial target device depending on the clock relationship between `ssi_clk` and `hclk`, and synchronization depth (`SSIC_H_2_S_SYNC_DEPTH`) used.

## 2.29.6 Enabling Continuous Transfer Mode in XIP Mode

To enable continuous transfer mode in XIP mode in `DWC_ssi`, choose the **Yes** option for the **Enable Continuous Transfer Mode in XIP mode?** field using the **SPI Parameters** tab during the Specify Configuration Activity in coreConsultant.

## 2.29.7 Registers Related to Continuous Transfer Mode in XIP

The following are the registers related to Continuous Transfer Mode in XIP:

- `SPI_CTRLR0`
- `CTRLR0`
- `XIP_CNT_TIME_OUT`

For more information about these registers, see the *Registers Descriptions* chapter.

## 2.30 Data Pre-fetch in XIP Operations

For each XIP request on the AHB interface, DWC\_ssi sends instruction (if SPI\_CTRLR0.XIP\_INST\_EN = 1) and address to the end device. The device responds with the data after some time, which is then sent to the AHB interface. In order to read a big chunk of data from the target, which is divided into multiple bursts, each of them takes the same amount of time to complete. Since DWC\_ssi needs to send the address and data for each request, the controller must wait until the instruction and address phase is completed before getting the data.

Using the data pre-fetch feature in DWC\_ssi, the controller pre-fetched the data for the successive burst during the current XIP transaction. If the next transaction request is made to the successive address, the data can be read directly from the RX FIFO instead of waiting for a new address and data to be sent to the device. This improves the overall performance of the system.

### 2.30.1 Description of Data Pre-fetch in XIP Operations

The amount of the data to be pre-fetched should be equal to the burst length of the last AHB request or the FIFO depth (whichever is lower). For example, if AHB defines a burst length of 16 starting from address 0x00, then DWC\_ssi fetches 16 beats to complete the current transfer, and then again 16 more data frames will be fetched and kept into the data register. If AHB bus again requests for data starting from end address for last transfer, then the rest of the data is sent to the device from RX FIFO itself. In parallel, DWC\_ssi again starts an XIP transfer to the device to pre-fetch the next chunk of data. If AHB manager does not define the contiguous address, then current data is flushed out from the FIFO and the controller starts a fresh transaction.

When DWC\_ssi completes the current burst, and is pre-fetching the data for the next burst, a new XIP request may be placed. In this case, DWC\_ssi can terminate the current transfer, or increase the number to data beats to be fetched depending on the address. Following are the possible use cases:



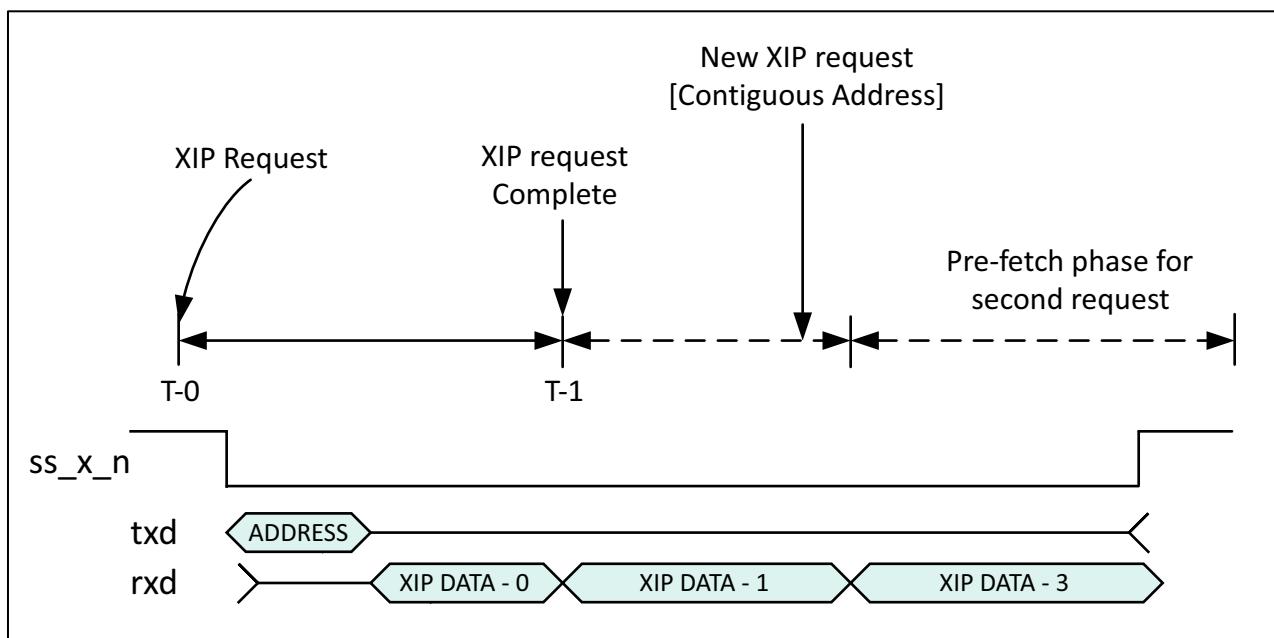
**Note** The HSIZE should be kept constant for all transfers in the Prefetch mode.

#### Case A: Transfer Request for the Contiguous Address

In this case, DWC\_ssi returns the number of beats available in the FIFO, and then fetches the rest of the beats to complete the current transfer. After the current transfer is complete, DWC\_ssi extends the same transfer to pre-fetch the data for the next transfer.

For example:

1. At T-0 DWC\_ssi receives an XIP request for burst of 16.
2. At T-1 DWC\_ssi completes the current XIP request, and proceed to pre-fetch 16 data frames.
3. While DWC\_ssi is pre-fetching next 16 beats, another XIP request comes with burst length 16.

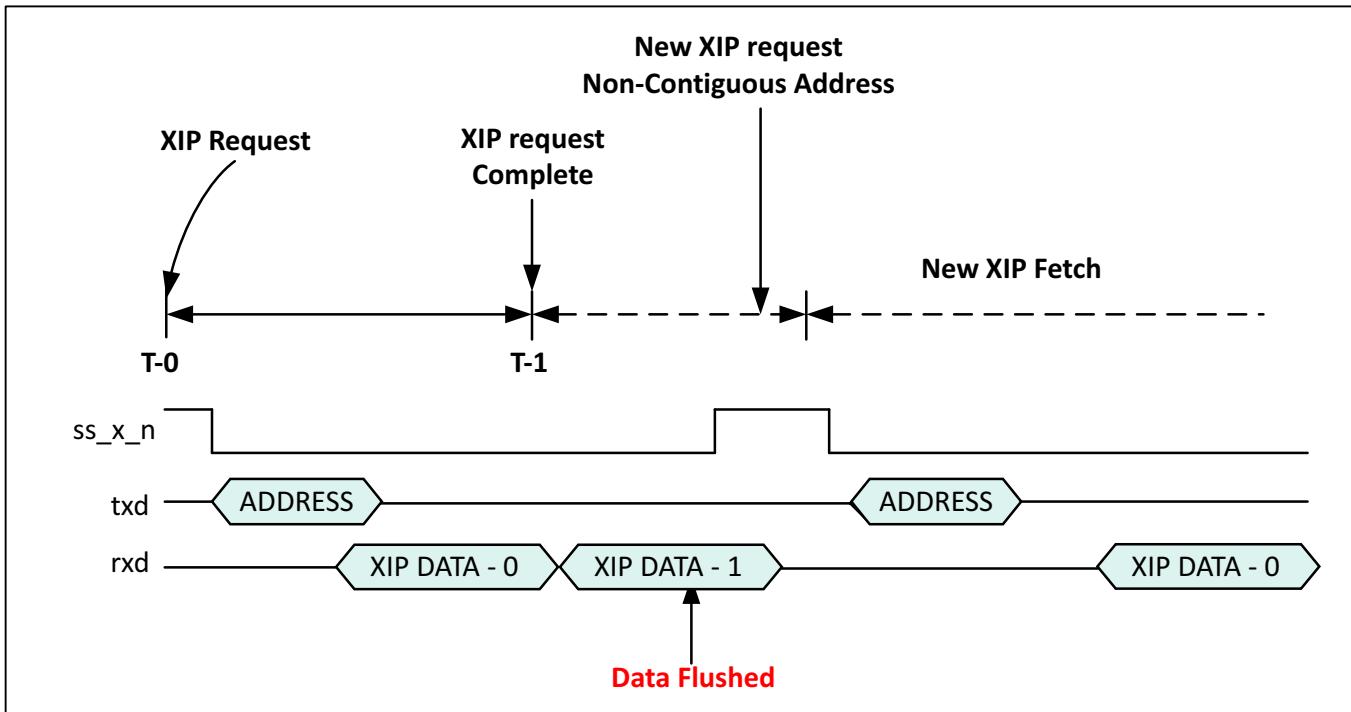
**Figure 2-136 Transfer Request for the Contiguous Address**

1. In this case, DWC\_ssi serves the second request with data available in the FIFO and then completes the burst.
2. Once the current transfer is completed, DWC\_ssi proceeds to fetch 16 more data frames to complete the pre-fetch.

This process continues if DWC\_ssi keeps on receiving the XIP requests while performing the pre-fetch. This feature can be used to extend the current XIP transfer to fetch any amount of data beats.

### Case B: Transfer Request for the Non-Contiguous Address

In this case, the current serial transfer ends gracefully, receive FIFO gets flushed and a new serial transfer is initiated on serial lane to complete the current request.

**Figure 2-137 Transfer Request for Non-Contiguous Address**

If the XIP request appears on the AHB interface after pre-fetch is completed, then a new serial transfer is initiated on the serial interface irrespective of the address of the request.

If XIP continuous mode and pre-fetching are enabled, then the data is pre-fetched from the device and the chip select pin remains asserted. During the successive request, the data is read from the RX FIFO and in parallel DWC\_ssi pre-fetches the data for the successive address. This process continues until a non-contiguous address is received, or a non-XIP transaction is detected on the AHB interface. In this case, the data that is pre-fetched is flushed from the FIFO and a new transfer is started.

For non-contiguous mode, once the pre-fetch is completed the chip-select pin gets de-asserted, and further pre-fetch happens by asserting the chip-select again and issuing a new read request to an XIP device. Hence for the continuous read mode the latency becomes less since we need not send the instruction and address to the device and data is fetched sooner.

WRAP transfer types is only supported in continuous XIP mode, in which the data from the next incremental address is pre-fetched. If another WRAP transfer is detected after a previous WRAP transaction, then the pre-fetched data is flushed and a new transfer is initiated by DWC\_ssi.

For more information on how the WRAP transfers happen in the continuous XIP mode, see "[Continuous WRAP Transfer](#)" on page 183. Other than continuous XIP mode, no pre-fetch is done for WRAP transfers in XIP transfers.

It is not mandatory to define the same burst type for pre-fetch transactions. Software can define the combination of different burst types (for example INCR4 from address 0x00 followed by INCR8 from address 0x10) until and unless the start address is contiguous. Once DWC\_ssi detects a non-contiguous address, the data stored in RX FIFO is flushed, and a new XIP transaction will be issued to device.



**Note** If XIP pre-fetching is enabled, AHB request for incremental transfer of undefined length (`hburst = 3'b001`) is not allowed.

DWC\_ssi responds to the new transfer request as described in the following scenarios:

#### Case 1: The Next Burst Length Smaller than the Pre-Fetched Data

In this case, DWC\_ssi responds with available data in the FIFO and does not initiate any new transaction on the SPI if RX FIFO has enough data to perform two bursts of the current burst length.

For example, if the initial XIP request is of INCR16, then DWC\_ssi serves this current request by reading 16 frames, and pre-fetches 16 more beats. If the next XIP request is of INCR4, then DWC\_ssi responds immediately and does not perform any new SPI transfer because the RX FIFO has sufficient data for the next pre-fetch ( $\text{burst length} * 2 = 4 * 2 = 8$ ).

#### Case 2: The Burst Length Larger than the Data Stored in RX FIFO

In this case, DWC\_ssi responds with the data available in RX FIFO and initiates an XIP burst in parallel. Once the data is available, controller completes the rest of the burst. Also, DWC\_ssi pre-fetches next burst length and keeps it in RX FIFO for the next transfer.

For example, if the initial XIP request is of INCR4, then DWC\_ssi has 4 data frames in the RX FIFO after the pre-fetch is complete. If the next XIP request is of INCR16, then DWC\_ssi responds with 4 data frames immediately and start to fetch 28 more data frames (12 for this current transfer + 16 for pre-fetch).

#### Case 3: AHB Manager with New Transfer request in Middle of Pre-fetch

If AHB manager comes up with new contiguous XIP transfer while the pre-fetch for the last transfer was not completed, DWC\_ssi responds with the available data and performs further pre-fetch. This is the same behavior as explained in “Case 2: The Burst Length Larger than the Data Stored in RX FIFO”.

If the AHB manager comes up with a new XIP transfer which is not contiguous, then the on-going SPI pre-fetch transfer is stopped gracefully and new XIP transfer is issued on the SPI lane. The pre-fetch data for the XIP transfer is flushed when DWC\_ssi is disabled.

To know if DWC\_ssi is busy performing the pre-fetch for the contiguous address software should poll for SSI busy status in SR (Status Register). Software should wait for busy status to go LOW to perform any other (non-XIP) operation.

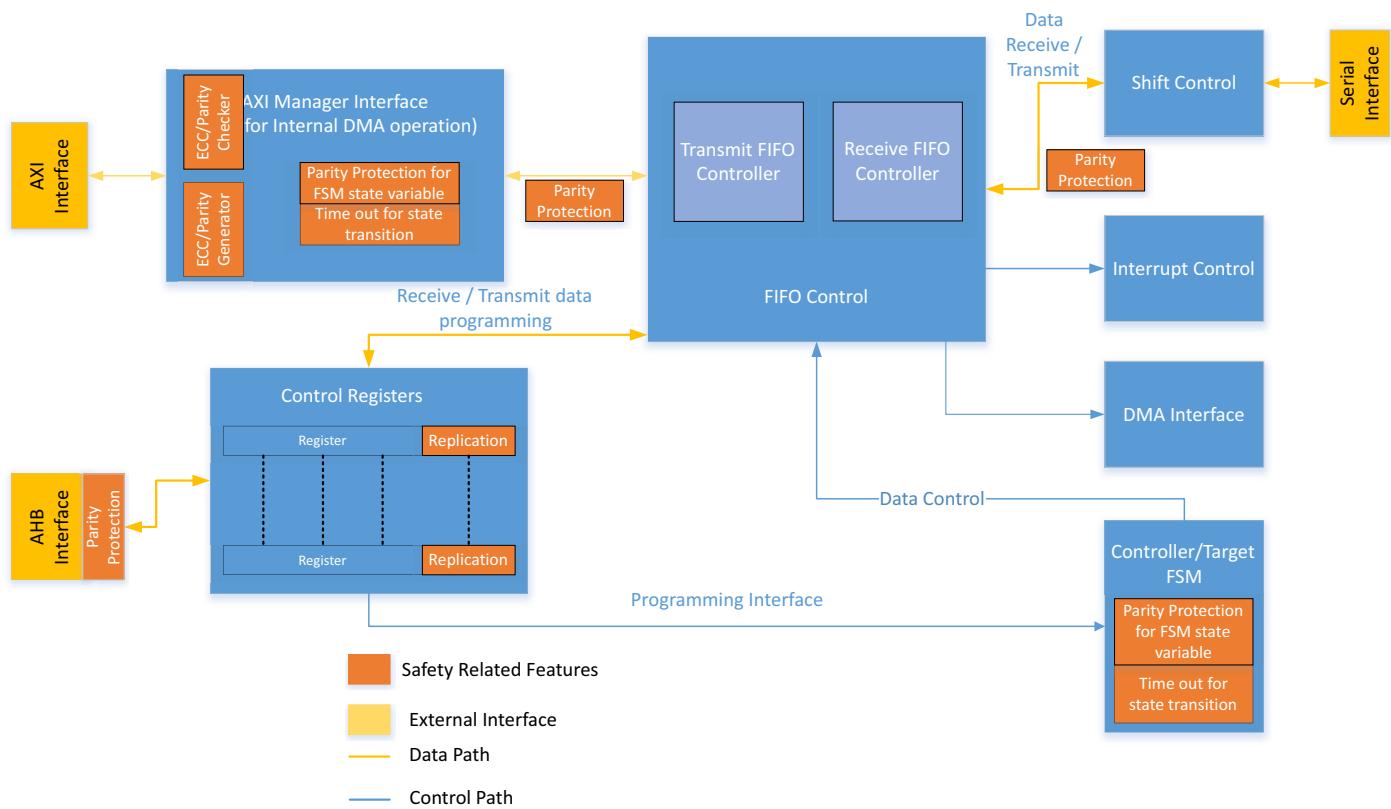
### 2.30.2 Enabling Data Pre-fetch in XIP Operations

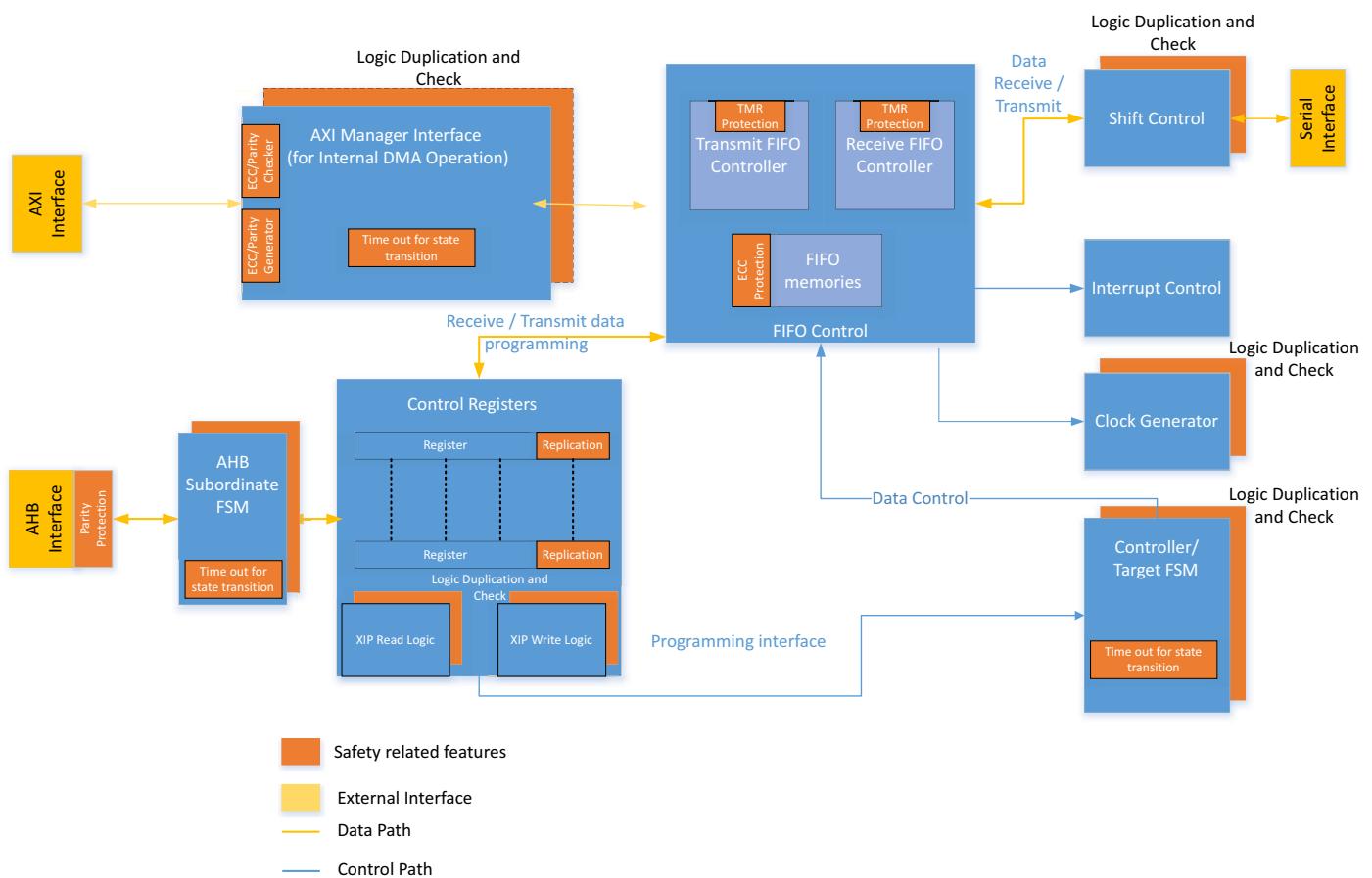
To enable the data pre-fetching feature for DWC\_ssi, choose the **Yes** option for the **Enable XIP Pre-fetch feature?** field using the **Enhanced SPI parameters** tab during the Specify Configuration Activity in coreConsultant.

# Automotive Safety

DWC\_ssi safety package contains features, which are aimed to make the IP safer for automotive applications. [Figure 3-1](#) and [Figure 3-2](#) show an overview of DWC\_ssi and the safety features that are available as part of this package.

**Figure 3-1 Safety Package 1 Safety Features Overview**



**Figure 3-2 Safety Package 2 Safety Features Overview**

Following is the brief description for the same:

- Parity protection for external interface AHB Subordinate interface.
  - AHB Control and data signals are protected by Parity.
- ECC and Parity Protection for AXI interfaces.
  - AXI Read and Write data signals are protected by ECC or Parity (configurable).
  - Control signals are protected by Parity.
- Register duplication for configuration registers.
- Time out register is added for FSM state transitions.
- Data path protection with Parity. (Safety Package 1)
 

Main data paths are protected by Parity as shown in [Figure 3-1](#) on page 191.
- Parity Protection for FSM state variables. (Safety Package 1)
- Data Path ECC Protection (Safety Package 2)
- Logic Duplication (Safety Package 2)
 

FSM's Shift Registers and Clock generator logic are duplicated for enhanced protection.

This chapter has the following sections:

- “Parity Protection on AHB Subordinate Interface” on page 194
- “AXI Manager Interface Protection” on page 197
- “Finite State Machine (FSM) Timeout Protection” on page 204
- “FSM One-hot Protection” on page 206
- “Logic Duplication” on page 207
- “TMR Protection for the FIFO Controller” on page 209
- “On-Chip Data Protection with Parity” on page 210
- “Data Path ECC Protection” on page 213
- “Register Space Protection” on page 219



DWC\_ssi Automotive Safety features are only supported via Prime Profiles. Please refer to the DesignWare Cores Synchronous Serial Interface (SSI) User Guide for more details.

## 3.1 Parity Protection on AHB Subordinate Interface

### 3.1.1 Overview of Parity Protection on AHB Subordinate Interface

The Parity Protection feature is implemented to provide high level data integrity check for all data transfers of AHB subordinate interface.

### 3.1.2 Description of Parity Protection on AHB Subordinate Interface

The Parity Protection feature can be enabled using the configuration parameter `SSIC_SLVIF_PAR_EN`. Parity check and generation can be enabled using the `PAR_EN` field of the SSI Safety Control (`SAFETYCR`) register. The `PAR_MODE` field of the `SAFETYCR` register is used to choose the odd and even parity.

If the configuration parameter `SSIC_SLVIF_PAR_EN` is set to 2, then `DWC_ssi` receives parity for both address and data signals; if the configuration parameter `SSIC_SLVIF_PAR_EN` is set to 1, then `DWC_ssi` receives parity per byte during the AHB write transfer. These parity signals are received through the side-band signals. For example, if the data bus width is 32 bit, then 4 bit parity must be received.

### 3.1.3 Address Parity Checker

The locally computed parity for the received address is compared with the parity received through the side-band interface. The result of the comparison is indicated through the `hresp` signal on the AHB interface. In the case of address parity check failure, `ERROR` response is returned on the `hresp` signal, and the write operation is unsuccessful. The address parity failure is also indicated through a separate `ssi_sfty_addr_sp_intr(_n)` interrupt output. The actions taken by `DWC_ssi` core in case of address parity protection failure are explained here for the different scenarios.

- Address Parity failure in register access (read/write)
  - Current register access is ignored. In case of write transfer, register is not updated, and in case of read transfer the data returned is all 0's.
  - Interrupt and `ERROR` response are generated.
  - The subsequent register accesses go through.
  - `ssi_sfty_addr_sp_intr(_n)` interrupt is de-asserted when the user clears it.
- Address Parity failure in Data Register (DR) access
  - Interrupt and `ERROR` response are generated.
  - Subsequent accesses to data registers are not allowed until the interrupt `ssi_sfty_addr_sp_intr(_n)` is cleared.
- Address Parity failure in XIP transfers
  - Interrupt and `ERROR` response are generated.
  - If an interrupt `ssi_sfty_addr_sp_intr(_n)` is asserted due to address parity failure, the next fresh XIP transfers goes through even if the earlier interrupt is not cleared.
  - Single transfers – Transfer is not started on the SPI side.
  - Burst transfers:
    - Address failure in the first beat – transfer is not started on the SPI side.

- If the first beat has no address parity failure and the second beat gets an address parity failure, then the DWC\_ssi subordinate is de-selected. If the manager continues the transfer, then an ERROR response is sent. Moreover, the DWC\_ssi internally flushes all data.
- Burst transfers with Data Prefetch - DWC\_ssi acts in the same way as it does for the burst transfers.

The separate interrupt `ssi_sfty_addr_sp_intr(_n)` is asserted when address parity failure occurs. The AHB address parity error is indicated by the SLVIFAPES field of the Safety Interrupt Status register (SAFETYISR).

### 3.1.4 Write Data Parity Checker

The locally computed parity for the received write data is compared with the parity received through the side-band interface (hwuser). The actions taken by DWC\_ssi core in case of write data parity protection failure is explained here for the different scenarios.

- Register access
  - An interrupt `ssi_sfty_data_sp_intr(_n)` is asserted to indicate the write data parity failure. No AHB ERROR response is given.
  - Subsequent register accesses go through.
- Data Register access
  - An interrupt `ssi_sfty_data_sp_intr(_n)` is asserted to indicate the write data parity failure and subsequent data register accesses are not allowed.

The separate interrupt `ssi_sfty_data_sp_intr(_n)` is asserted when write data parity failure occurs. The AHB write data parity error is indicated by the SLVIFDPES field of the Safety Interrupt Status register (SAFETYISR).

### 3.1.5 Read Data Parity Generation

DWC\_ssi provides the parity per byte during the AHB read transfer along with the read data. These parity signals are sent through the side-band signals and computed statically using an internal parity computation logic. The receiving manager interface can use this read parity to check the data integrity of the read data. The read data parity is always provided if SSIC\_SAFETY\_PKG\_EN parameter is set to 1, even if SSIC\_SLVIF\_PAR\_EN parameter is set to 0.

### 3.1.6 Registers Related to Parity Protection on AHB Interface

The following registers are related to Parity Protection on AHB Interface:

- SAFETYCR
- SAFETYISR
- SAFETYICR

For more information about these registers, see the *Registers Description* chapter.

### 3.1.7 Signals Related to Parity Protection on AHB Interface

The following signals are related to Parity Protection on AHB Interface:

- `ssi_sfty_data_sp_intr(_n)`: When individual interrupts are selected.
- `ssi_sfty_addr_sp_intr(_n)`: When individual interrupts are selected.

- `ssi_sfty_ce_intr(_n)`: When combined interrupts are selected.  
For more information about these signals, see the *Signals Description* chapter.

## 3.2 AXI Manager Interface Protection

### 3.2.1 Overview of Safety Features on AXI Interface

AXI Manager interface is protected with ECC and Parity protection feature. Following sections detail the ECC and parity protection feature for various channels.

### 3.2.2 Protection for Write Address and Read Address Channels

The AXI Manager Interface AR/AW Channel Safety block is implemented using the AXI Interface Parity Generator.

The AXI Manager Interface AR/AW Channel handshake signals ARVALID/AWVALID and ARREADY/AWREADY are protected with parity as described in “[AXI Interface VALID and READY Parity Generator and Checker](#)” on page [203](#)

- The ARREADY/AWREADY parity signal is sent over the non-standard sideband signal.
- The computed Parity signals are packed and sent over the ARUSER/AWUSER as described in “[AXI Parity Protection](#)” on page [200](#).
- The ARVALID/AWVALID/ARREADY/AWREADY parity signal is sent over the non-standard sideband signal.

### 3.2.3 Protection for Write Data Channel

The AXI Manager Interface W Channel Safety block is implemented using the AXI Interface ECC and Parity Generator. Refer to “[AXI Interface ECC Protection for Write and Read Data](#)” on page [198](#) for more details on the AXI Interface ECC and Parity Protection.

The AXI Manager Interface W Channel signals are grouped into the following two payloads for the computation of ECC and Parity:

- W Channel Data (WDATA)
- W Channel Control Signals (WCTRL)

The W Channel Payload 0 and Payload 1 used for computation of ECC and Parity is packed as follows:

- W Channel Payload 0 = WDATA
- W Channel Payload 1 = {WLAST, WSTRB, WID}

The AXI Manager Interface W Channel handshake signals WVALID and WREADY are protected with parity as described in “[AXI Parity Protection](#)” on page [200](#)

The computed ECC or Parity signals are packed and sent over the WUSER as follows:

$$\text{WUSER} = \{\text{W Channel Payload 1 Parity}, \text{W Channel Payload 0 ECC/Parity}\}$$

The WVALID and WREADY parity signal is sent over the non-standard sideband signal.

### 3.2.4 Protection for Read Data Channel

The AXI Manager Interface R Channel Safety block is implemented using the AXI Interface ECC and Parity Monitor. Refer the section “[AXI Interface ECC Protection for Write and Read Data](#)” on page [198](#) for more details on the AXI Interface ECC and Parity Monitor.

The AXI Manager Interface R Channel signals are grouped into the following two payloads for the validation of ECC and Parity:

- R Channel Data (RDATA)
- R Channel Control Signals (RCTRL)

The R Channel Payload 0 and Payload 1 used for validation of ECC and Parity is packed as follows:

- R Channel Payload 0 = RDATA
- R Channel Payload 1 = {RLAST, RRESP, RID}

The AXI Manager Interface R Channel handshake signals RVALID and RREADY are protected with parity as described in “[AXI Parity Protection](#)” on page [200](#). The computed ECC or Parity signals are packed and sent over the RUSER as follows:

$$\text{RUSER} = \{\text{R Channel Payload 1 Parity}, \text{R Channel Payload 0 ECC/Parity}\}$$

The RVALID and RREADY parity signal is sent over the non-standard sideband signal.

### 3.2.5 Protection for Write Response Channel

The AXI Manager Interface B Channel Safety block is implemented using the AXI Interface Parity Monitor. Refer to “[AXI Parity Protection](#)” on page [200](#) for more detail on the AXI Interface Parity Monitor.

The AXI Manager Interface B Channel signals are grouped into the following payload for the validation of ECC and Parity:

- B Channel Control Signals (BCTRL)

The B Channel Payload 0 used for validation of ECC and Parity is packed as follows:

- B Channel Payload 0 = {BRESP, BID}

The AXI Manager Interface B Channel handshake signals BVALID and BREADY are protected with parity as described in “[AXI Parity Protection](#)” on page [200](#).

The computed ECC and Parity signals are packed and sent over the BUSER as follows:

$$\text{BUSER} = \{\text{B Channel Payload 0 Parity}\}$$

The BVALID/BREADY parity signal is sent over the non-standard sideband signal.

### 3.2.6 AXI Interface ECC Protection for Write and Read Data

The AXI Interface ECC Protection is designed based on the Synopsys DWbb\_ecc module.

[Table 3-1](#) provides information about how the ECC Checkbits width changes against the input ECC Payload width.

**Table 3-1      ECC Payload Width vs ECC Check Bits Width**

SL. No.	Payload Width	ECC Checkbits Width
1	Up to 11 bits	5
2	Up to 26 bits	6
3	Up to 57 bits	7
4	Up to 120 bits	8

### 3.2.6.1 ECC Granularity

The ECC Generation is a combinational logic. As the payload size increases for ECC computation, it becomes difficult to meet the higher timing requirements. Hence, ECC Granularity feature is introduced to improve the timing characteristics of the ECC Generator logic.

The ECC Generator Supports the following combination of ECC Granularities for different payloads, which are multiple of 32 bits shown in the [Table 3-2](#).

**Table 3-2      ECC Payload and ECC Check Bits with Different ECC Granularity (multiple 32 bits)**

ECC Granularity	Full Width <code>SSIC_ECC_GRAN_TYPE = 0</code>	32 Bits <code>SSIC_ECC_GRAN_TYPE = 1</code>	64 Bits <code>SSIC_ECC_GRAN_TYPE = 2</code>
Payload Width			
32	7	7	7 <sup>a</sup>
64	8	2x7	8

- a. Indicates that if the payload width is less than Granularity (that is 64 bits in this example), then ECC is generated with full width granularity.

The ECC Generator Supports the following combination of ECC Granularities for different payloads (General) shown in the [Table 3-3](#).

**Table 3-3      Supported ECC Granularity and Payload**

ECC Granularity	Full Width <code>SSIC_ECC_GRAN_TYPE = 0</code>	32 Bits <code>SSIC_ECC_GRAN_TYPE = 1</code>	64 Bits <code>SSIC_ECC_GRAN_TYPE = 2</code>
Payload Width			
8	5	5 <sup>a</sup>	5 <sup>a</sup>
16	6	6 <sup>a</sup>	6 <sup>a</sup>
24	6	6 <sup>a</sup>	6 <sup>a</sup>
32	7	7 <sup>a</sup>	7 <sup>a</sup>
40	7	2x7 <sup>b</sup>	7 <sup>a</sup>
58	8	2x7 <sup>b</sup>	8 <sup>a</sup>
64	8	2x7	8

- a. Indicates that if the payload width is less than Granularity, then ECC is generated with Full width granularity.
- b. Indicates that the payload width is divisible by Granularity more than once, but not multiple of granularity.

In this case, ECC is computed for each whole Granularity width payload bits. The rest of the bits (which is not multiple of granularity) are prefixed with 0's to pack it to Granularity width and then ECC is computed.

#### Example-1:

Payload Width = 58 and Granularity = 32

For the lower 32 bits ECC Computed. For the upper 26 bits, zeroes are appended to make it 32 bit width and then ECC is computed. The total ECC Check bits required are  $2 \times 7 = 14$  bits.

#### Example-2

Payload Width = 96 and Granularity = 64

For the lower 64-bits ECC Computed. For the upper 32 bits, zeroes are appended to make it 64 bits width and then ECC is computed. The total ECC Check bits required are  $2 \times 8 = 16$  bits.

### 3.2.6.2 AXI Interface ECC Monitor/Checker

The AXI Interface monitors and validates the ECC of the AXI Channels that are received by AXI Subordinate.

- If ECC Monitor detects the single bit error, then received payload is corrected and a correctable interrupt is generated.
- If a multi-bit error is detected then an uncorrectable interrupt is generated, but the received payload cannot be corrected.

The AXI Interface ECC Monitor is designed based on the Synopsys DWbb\_ecc module. This ECC Generator supports payload width up to 512 bits for the generation of the ECC.

### 3.2.6.3 AXI Channel ECC Error Interrupts

The ECC Monitor generates the ECC Protection interrupts to indicate that the correctable and uncorrectable errors have been detected.

- AXI Channel ECC Correctable Error Interrupt: `ssi_sfty_axi_ce_intr(_n)`

As stated earlier, if single bit error is detected, then the Correctable Error Interrupt is generated by the ECC Monitor. In the case, ECC Monitor can correct the received payload.

- AXI Channel ECC Uncorrectable Error Interrupt: `ssi_sfty_axi_ue_intr(_n)`

If multi-bit error is detected, then the Uncorrectable Error Interrupt is generated by ECC Monitor. In this case ECC Monitor cannot correct the received payload.

### 3.2.6.4 AXI Channel ECC Error Counter

The AXI Channel ECC Monitor supports the following ECC Error Counters:

- Correctable Error Counter
- Uncorrectable Error Counter

**AXI Channel ECC Correctable Error Counter:** The AXI Channel ECC Correctable Error Counter captures the frequency of the single bit error.

**AXI Channel ECC Uncorrectable Error Counter:** The AXI Channel ECC Uncorrectable Error Counter captures the frequency of the multi-bit error.

## 3.2.7 AXI Parity Protection

The AXI Interface Parity Protection logic implements the logic to generate and check the Parity for the AXI Channels that are being driven by the AXI Manager. This Parity Protection logic is optionally selected using the `SSIC_AXI_SAFETY_EN` parameter. Depending on the configuration chosen, this logic is implemented or removed.

The Parity Generator supports the following two types of Parity Mode:

- Per Byte Parity Mode
- ARC Compatibility Parity Mode

The Parity mode is configurable through the SSIC\_PAR\_MODE parameter.

You can choose to have the Parity type as Odd or Even through the configuration parameter SSIC\_PAR\_TYPE.

### 3.2.7.1 Per Byte Mode

DWC\_ssi provides parity for AXI write address, read address, and write data channel in the similar fashion using side-band signals. The host can use this parity to check data integrity of these channels. [Table 3-4](#) shows the parity bits mapping to the corresponding user signals on the AXI interface.

**Table 3-4 AXI Interface Signal Mapping for Parity Bits**

Channel	Signal Names	Parity Bits
<b>AXI write Address Channel</b>	awaddr[SSIC_AXI_AW-1:0] {awid[SSIC_AXI_IDW-1:0] awlen[SSIC_AXI_BLW-1:0] awszie[2:0] awburst[1:0] awlock[SSIC_AXI_LTW-1:0] awcache[3:0] awprot[2:0]}	awuser [aw-1:0]  awuser[AWUSERW-1:aw] aw = SSIC_AXI_AW/8
<b>AXI Read Address Channel</b>	araddr[SSIC_AXI_AW-1:0] {arid[SSIC_AXI_IDW-1:0] arlen[SSIC_AXI_BLW-1:0] arsize[2:0], arbust[1:0] arlock[SSIC_AXI_LTW-1:0] arcache[3:0] arprot[2:0]}	aruser[aw-1:0]  aruser[ARUSERW-1:aw] aw = SSIC_AXI_AW/8
<b>AXI Write Data Channel</b>	wdata[SSIC_AXI_DW -1:0] {wid[SSIC_AXI_IDW-1:0] wstrb[SSIC_WSW-1:0] wlast} WID signal is not present for AXI4 interface.	wuser[dw-1:0]  wuser[WUSERW-1:dw] dw = SSIC_AXI_DW/8
<b>AXI Read Data Channel</b>	rdata[SSIC_AXI_DW -1:0] {rid[SSIC_AXI_IDW-1:0] rresp[1:0] rlast}	ruser [dw-1:0]  ruser[WUSERW-1:dw] dw = SSIC_AXI_DW/8

Channel	Signal Names	Parity Bits
<b>AXI Write Response channel</b>	{bid[SSIC_AXI_IDW-1:0] bresp[1:0]}	buser[BUSERW-1:dw]



**Note** If a signal group is not a multiple of 8, then extra bits are padded with 0s to get the parity value.

### 3.2.7.2 ARC Compatibility Parity Mode

The ARC Compatibility Parity mode is provided to align the Parity Generation implementation with ARC Processor Parity implementation. This Parity mode provides the interoperability between the ARC Processor's and Synopsys DesignWare Cores with Safety features.

Table 3-5 provides details of the Parity computation in the case of ARC Compatibility Parity Mode.

**Table 3-5 AXI Interface Parity Generator - ARC Compatibility Parity Mode**

SL. No.	AXI Channel Payload	ARC Compatibility Parity Mode
1	ARADDR/AWADDR	The Parity Generator computes the ARADDR/AWADDR parity per 8-bits. <b>Note:</b> If the payload is not multiple of 8, then 0s are appended to make multiple of 8-bits.
2	AR/AW Channel Control (ARCTRL/AWCTRL)	The Parity Generator computes the Parity for AR/AW Channel Control signals as follows: [0] - Parity Bit for ARID/AWID [1] - Parity Bit for ARLEN/AWLEN [2] - Parity Bit for {ARLOCK, ARBURST, ARSIZE}/{AWLOCK, AWBURST, AWSIZE} [3] - Parity Bit for {ARPROT, ARCACHE}/{AWPROT, AWCACHE}
3	WDATA	The Parity Generator computes the WDATA parity per 8-bits.
4	W Channel Control (WCTRL)	The Parity Generator computes the Parity for W Channel Control signals as follows: [0] - Parity Bit for WID [1] - Parity Bit for WSTRB [2] - Parity Bit for WLAST
5	RDATA	The Parity Generator computes the WDATA parity per 8 -bits.
6	R Channel Control (RCTRL)	The Parity Generator computes the Parity for R Channel Control signals as follows: [0] - Parity Bit for RID [1] - Parity Bit for RRESP and RLAST

SL. No.	AXI Channel Payload	ARC Compatibility Parity Mode
7	B Channel Control (BCTRL)	The Parity Generator computes the Parity for B Channel Control signals as follows: [0] - Parity Bit for BID [1] - Parity Bit for BRESP

### 3.2.7.3 AXI Channel Parity Error Interrupts

The AXI Channel Parity Monitor/Checker validates the parity of the AXI Payload 0 and/or 1 and if the check fails, then AXI Channel Payload 0/1 Parity Check Error Interrupt `ssi_sfty_ap_intr(_n)` is asserted.

### 3.2.8 AXI Interface ECC and Parity Generator - Register Slice

The ECC and Parity Generator provides an optional feature to add Register Slice after computation of the ECC or Parity. This helps to break timing path between the ECC or parity computation logic and external AXI Interconnect/AXI Manager/Subordinate Interface logic, thereby allowing design to meet higher frequency requirements.

### 3.2.9 AXI Interface VALID and READY Parity Generator and Checker

The AXI Interface handshake signals VALID and READY are protected with the parity signal. This feature is a configurable option enabled through the `SSIC_READY_VALID_PAR_EN` parameter. The parity type that is Odd or Even parity is configurable through the `SSIC_READY_VALID_PAR_TYPE` parameter.

The VALID Parity generator computes the parity for the VALID signal and drives the same along with the USER signal.

This logic block computes the parity for the following VALID signals depending on which AXI Channel it is used.

- AXI Manager Interface: ARVALID/AWVALID/WVALID
- AXI Subordinate Interface: RVALID/BVALID

The READY Parity checker validates the parity of the READY signal. The READY signal parity is received through the respective non-standard sideband signal.

If the READY Parity checker validation fails, then READY Parity Check Error Interrupt is asserted.

This block validates the parity for the following READY signals depending on which AXI Channel it is used.

- AXI Manager Interface: RREADY/BREADY
- AXI Subordinate Interface: AREADY/AREADY/WREADY



The valid/ready error for AXI Read Address and AXI Read data channels are considered only when the `CTRLR0.TMOD` register bit is set to 2b01 (SPI Write). Consequently, valid/ready error for AXI Write Address, Write Data and Write Response channels will be considered only when the `CTRLR0.TMOD` register bit is set to 2'b10 (SPI Read).

## 3.3 Finite State Machine (FSM) Timeout Protection

### 3.3.1 Overview of FSM Timeout Protection

Finite State Machine (FSM) time out feature provides the mechanism to ensure that all FSMs in DWC\_ssi can complete transaction, and return to a known completion state within a programmed time out value.

DWC\_ssi contains FSM in AHB, SSI and AXI clock domains. Each clock domain has one time out register. This can be programmed using the FSM time out registers for their respective clock domains. The timer ticks are generated based on the programmable time-out value. At every timer tick, all the FSMs are monitored for being in the active state (non-IDLE state, which indicates FSM is actively processing transactions or hand-shakes) and an internal FSM active flag field is set. The internal active flag bit is implemented for every FSM that is monitored for timeout. When the FSM reaches an IDLE or transaction completion state, the internal active flag field of FSM is reset. At subsequent timer tick, all FSM's active flag bits are monitored and any flag that is set indicates a time out for that FSM. This process of setting the flag, and checking at subsequent timer tick is repeated at every tick.

### 3.3.2 Description of FSM Timeout Protection

The FSM Time out error injection can be enabled by setting the `FSM_EI_EN` field to 1, and the `FSM_EI_TYPE` field to 0 in the `SAFETYCR` register. The FSM in which error injection should happen is selected using the `FSM_SEL` programming field of the `SAFETYCR` register. When time out error injection is enabled for an FSM, the FSM automatically times out and generates interrupt even without traffic. As error injection is an Error Insertion Enable, timeout value can be set to a lower value at which the Error Insertion Enable ticks are required.

The FSM time out state error is indicated by `AXIFSMTOS`/`SSIFSMTOS`/`AHBFSTOS` fields of the `DWC_ssi` Safety Interrupt Status (`SAFETYISR`) register for AXI, SSI and AHB clock domains respectively. Also, the safety interrupt `ssi_sfyt_fto_intr(_n)` is asserted when an FSM error is detected in any of AXI, SSI or AHB clock domain.

FSM timeout provides a mechanism to ensure all the FSMs completes a transaction, and returns to the known completion state. If an FSM stops responding in a state for more than a programmed amount of time, then it is indicated to the application layer through a safety interrupt.

### 3.3.3 Error Injection for FSM Timeout Protection

When FSM timeout protection is enabled, by setting the `EI_EN` field to 1 and `EI_TYPE` to 2'b10 in the `SAFETYCR` register can be used to enable the error insertion.

The FSM in which error injection should happen is selected using the `EI_SEL` programming field of the `SAFETYCR` register. The FSM state error is indicated by the corresponding field of `DWC_ssi` Safety Interrupt Status (`SAFETYISR`) register. Also, the safety interrupt `ssi_sfyt_fto_intr(_n)` is asserted when an FSM timeout error is detected. The FSM in which error occurred can be known by reading the Safety Interrupt register.

Error injection mode is also supported for FSM parity error check per clock domain, this can be set using dedicated register bits in SSI Safety Control (`SAFETYCR`) register.

**Table 3-6 Error Injection Field Decode Values**

EI_SEL Field Decode	Logic Name
0	SSI Manager/Subordinate FSM

EI_SEL Field Decode	Logic Name
1	AHB Subordinate FSM
2	AXI Manager FSM
3	AXI DMA FSM

### 3.3.4 Registers Related to FSM Timeout Protection

The following register are related to FSM Timeout Protection:

- SLVIFFSMTOCR
- SSIFSMTOCR
- AXIFSMTOCR

For more information about these registers, see the *Registers Description* chapter.

### 3.3.5 Signals Related to FSM Timeout Protection

The following signals are related to FSM Timeout Protection:

- `ssi_sfty_fto_intr(_n)`: When individual interrupts are selected.
- `ssi_sfty_ue_intr(_n)`: When combined interrupts are selected.

For more information about these signals, see the *Signals Description* chapter.

## 3.4 FSM One-hot Protection

### 3.4.1 Overview of FSM One-hot Protection

All the Finite State Machines (FSM) that are part of DWC\_ssi are implemented using a one-hot encoding scheme. One parity bit is added for each FSM bus.

### 3.4.2 Description of FSM One-hot Protection

The FSM parity protection mechanism in DWC\_ssi continuously monitors if the FSM bus (including parity bit) deviates from the one-hot encoding scheme. Errors introduced due to transient bit flips or due to permanent stuck at-fault are detected by the safety mechanism and an uncorrectable safety interrupt is flagged.

### 3.4.3 Error Injection for FSM One-hot Protection

When FSM state one-hot protection error injection is enabled, by setting the EI\_EN field to 1, the EI\_TYPE field to 2'b11 in the SAFETYCR register and followed by writing SSIENR register to 1.

The FSM in which error injection should happen is selected using the EI\_SEL programming field of the SAFETYCR register. The FSM state error is indicated by the corresponding field of DWC\_ssi Safety Interrupt Status (SAFETYISR) register. Also, the safety interrupt `ssi_sfty_fp_intr(_n)` is asserted when an FSM error is detected.

**Table 3-7 Error Injection Field Decode Values**

EI_SEL Field Decode	Logic Name
0	SSI Manager/Subordinate FSM
1	AHB Subordinate FSM
2	AXI Manager FSM
3	AXI DMA FSM

### 3.4.4 Registers Related to FSM Parity Protection

The following registers are related to FSM Parity Protection:

- SAFETYCR
- SAFETYISR
- SAFETYICR

For more information about these registers, see the *Registers Description* chapter.

### 3.4.5 Signals Related to FSM Parity Protection

The following signals are related to FSM Parity Protection:

- `ssi_sfty_fp_intr(_n)`: When individual interrupts are selected.
- `ssi_sfty_ue_intr(_n)`: When combined interrupts are selected.

For more information about these signals, see the *Signals Description* chapter.

## 3.5 Logic Duplication

### 3.5.1 Overview for Logic Duplication

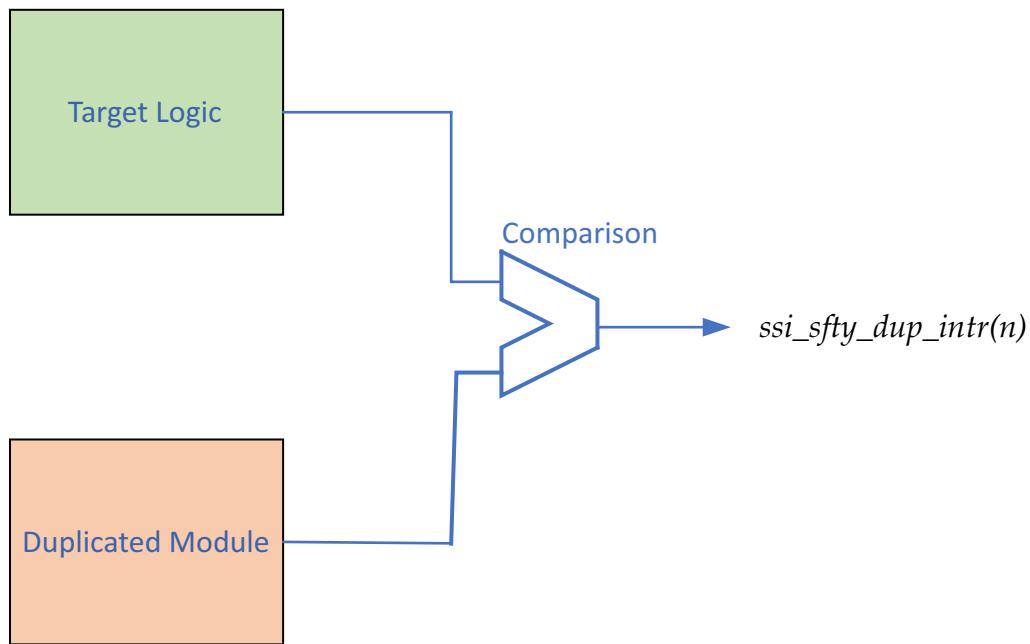
DWC\_ssi implements logic duplication for the critical logic to protect the outputs for those modules. Following modules are protected using this scheme:

- Manager/Subordinate FSM
- AXI DMA FSM
- Data packing and unpacking logic for AXI DMA functionality
- Clock Generator
- Shift Register
- XIP Read/Write Logic

### 3.5.2 Description of Logic Duplication

Under this protection scheme, the target logic is duplicated and the outputs of the target logic and the duplicated logic are compared.

**Figure 3-3 Example for Logic Duplication**



If the comparison fails, `ssi_sfty_dup_intr(_n)` / `ssi_sfty_ue_intr(_n)` interrupt will be generated. The error origin can be determined by the Safety interrupt Status Register (SAFETYISR).

### 3.5.3 Error Injection for Logic Duplication

Error injection capability is available for the Logic duplication mechanism. This can be done using dedicated programming bit in SSI Safety Control (SAFETYCR) register. Using the programming options

error can be injected in each protected logic. The EI\_SEL field is used to select the module in which error should be injected. You can select in which module (original or duplicate) the error should be inserted using the LRD\_EI\_TYPE field in the SAFETYCR register:

- 1: Error insertion in the duplicated register
- 0: Error insertion in the original register

[Table 3-8](#) lists all the instances which are protected using this feature and field decode in SEL programming field of the Safety Control Register SAFETYCR.

**Table 3-8 Logic Protected with Duplication and Field Decode Values**

EI_SEL Field Decode	Logic Name
0	DWC_ssi AHB Subordinate Interface
1	DWC_ssi Manager FSM
2	DWC_ssi Clock Generator Module
3	DWC_ssi Shifter module
4	DWC_ssi Data packer/unpacker Logic DMA Side
5	DWC_ssi Data packer/unpacker Logic Serial Side
6	DWC_ssi AXI DMA FSM
7	DWC_ssi XIP Read Logic
8	DWC_ssi XIP Write Logic
9	DWC_ssi Subordinate FSM

### 3.5.4 Registers Related to Logic Duplication Feature

The following registers are related to the Logic Duplication Feature:

- SAFETYCR
- SAFETYISR
- SAFETYICR

For more information about these registers, see the *Registers Description* chapter.

### 3.5.5 Signals Related to Logic Duplication Feature

The following signals are related to the Logic Duplication Feature:

- ssi\_sfty\_dup\_intr(\_n) when individual interrupts are selected.
- ssi\_sfty\_ue\_intr(\_n) when combined interrupts are selected.

For more information about these signals, see the *Signals Description* chapter.

## 3.6 TMR Protection for the FIFO Controller

### 3.6.1 Overview for TMR Protection for TMR Controller

DWC\_ssi implements Triple redundancy logic to protect the address points and the FIFO flags generation logic. This helps protect the error free operation of FIFO controller and the dependent logic (like FSM and interrupt generation).

### 3.6.2 Description of Logic Duplication

Triple redundancy logic protects the target logic by duplicating the logic three times and using majority vote logic to determine the correct output. The voting logic is used to detect and mask failures. Unlike comparator, the majority voter technique increases the availability by ensuring the functionality of the redundant logic even after one of them receives an error.

If one of the logic comparison fails, `ssi_sfty_tmr_intr(_n)` / `ssi_sfty_re_intr(_n)` interrupt will be generated. The error status could be determined by the Safety interrupt Status Register (SAFETYISR).

### 3.6.3 Registers Related to Logic Duplication Feature

The following registers are related to the Logic Duplication Feature:

- SAFETYISR
- SAFETYICR

For more information about these registers, see the *Registers Description* chapter.

### 3.6.4 Signals Related to Logic Duplication Feature

The following signals are related to the Logic Duplication Feature:

- `ssi_sfty_tmr_intr(_n)` when individual interrupts are selected
- `ssi_sfty_re_intr(_n)` when combined interrupts are selected

For more information about these signals, see the *Signals Description* chapter.

## 3.7 On-Chip Data Protection with Parity

### 3.7.1 Overview of On-Chip Data Protection with Parity

On-Chip Data Protection with Parity feature provides parity protection for transmit and receive data paths, where feasible. For every bite of data, one parity is generated.

### 3.7.2 Description On-Chip Data Protection with Parity

At transmit side, the data from AXI/AHB interface is protected until it reaches the shift control logic.

At receive side, the data from serial interface is protected from the serial shift register until it reaches the AHB subordinate interface, or AXI manager interface.

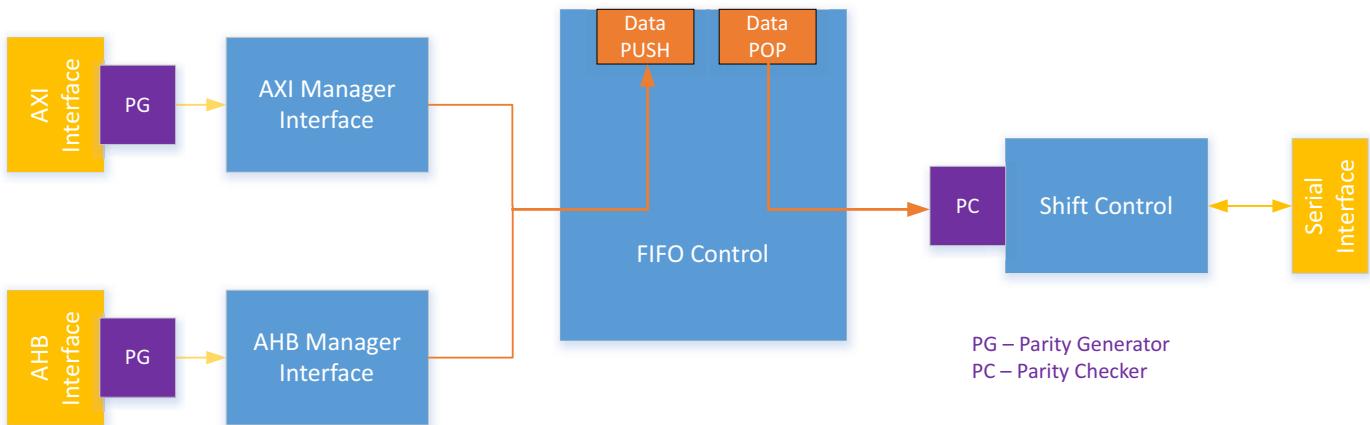


The AXI interface is used in internal DMA mode of operation, otherwise AHB subordinate interface is used for data transmission or reception. Only one of them can be active at a certain point.

### 3.7.3 Transmit Data Path Parity Protection

For data transmission the data can come from AXI or AHB subordinate interface based on internal programming. [Figure 3-4](#) describes the locations where parity generators and parity checkers are placed in the transmit data path.

**Figure 3-4 Data Path Parity Protection for Transmit Data Path**

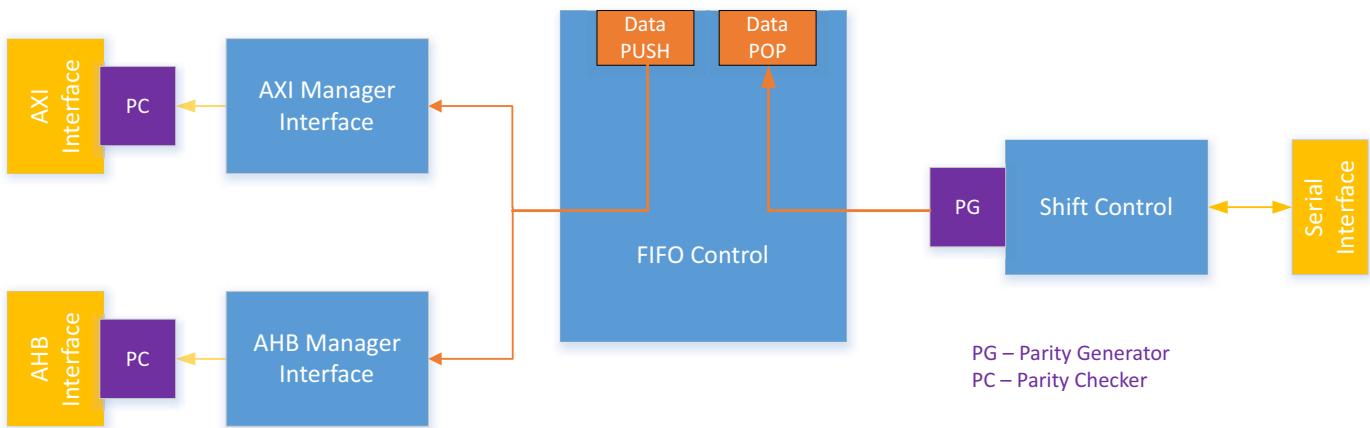


Following are the places where parity generators and checkers are present in the transmit data path:

- Parity is generated when AXI or AHB interface has data ready to shift inside the FIFO.
- Parity is checked when the data is loaded into the internal shift register.

### 3.7.4 Receive Data Path Parity Protection

[Figure 3-5](#) describes the locations where parity generators and parity checkers are placed in the receive data path.

**Figure 3-5 Data Path Parity Protection for Receive Data Path**

Following are the places where parity generators and checkers are available in the transmit data path:

- Parity is generated when serial interface has data ready to shift inside the FIFO.
- Parity is checked when the data is about to be sent in AHB/AXI interface.

### 3.7.5 Error Injection in Data Path Parity Protection

Each Parity generator supports control to inject parity error. When enabled, the parity of the incoming data is flipped until the parity error is detected by the IP. There are four possible parity error injection methods for each path, following is the programming, which should be followed to inject parity error in each path:

#### For Internal DMA Mode:

- If the transfer mode is set to Write, then Error is injected at AXI interface parity generator and error is generated at parity checker present before shift control logic.
- If the transfer mode is set to Read, then Error is injected at parity generator present after shift control logic and error is generated at the parity checker present in the AXI interface.

#### For Normal Operation Mode:

- If the transfer mode is set to Transmit Only, then Error is injected at AHB subordinate interface parity generator and error is generated at parity checker present before shift control logic.
- If the transfer mode is set to Receive Only, then Error is injected at parity generator present after shift control logic and error is generated at the parity checker present at AHB subordinate interface.

After the generator inserts error in the generated parity data, the subsequent parity checker should detect the mismatch and report the error status in SSI Safety Interrupt status (SAFETYISR) register.

For example, if DWC\_ssi needs to insert an error in the input data at parity generator 1, the sequence of operation is as follows:

1. Application sets the EI\_EN to 1 and EI\_TYPE to 2'b00 of the SSI Safety Control (SAFETYCR) register.
2. Application Enables DWC\_ssi by setting SSIENR register to 1.
3. The parity generator inserts error in the parity generated for the data.

4. As the data traverses through multiple paths, the subsequent checkers detects the parity mismatch and reports error status in SSI Safety Interrupt (SAFETYISR) register and generated `ssi_sfty_dpe_intr(_n)/ssi_sfty_ue_intr(_n)` interrupt.
5. After the interrupt is detected DWC\_ssi clears the EI\_EN bit in the SAFETYCR register.



- Note**
- For Error insertion the transfer mode field (CTRLR0 . TMOD) should not be set to 2'b00 (Transmit and Receive).
  - Error insertion is not applicable for XIP transfers.
  - For Error insertion the AXI transfer width of 64 (`DMACR . ATW=0x3`) and Data frame size of 32 `CTRLR0 . DFS=5'h1f` should be used.

The Data Path Parity feature is enabled by default, the odd or even parity can be selected using the SSIC\_PAR\_MODE parameter.

### 3.7.6 Registers Related to On-Chip Data Protection with Parity

The following signals are related to On-Chip Data Protection with Parity:

- SAFETYCR
- SAFETYISR
- SAFETYICR

For more information about these registers, see the *Registers Description* chapter.

### 3.7.7 Signals Related to On-Chip Data Protection with Parity

The following signals are related to On-Chip Data Protection with Parity:

- `ssi_sfty_dpe_intr(_n)`: When individual interrupts are selected.
- `ssi_sfty_ue_intr(_n)`: When combined interrupts are selected.

For more information about these signals, see the *Signals Description* chapter.

## 3.8 Data Path ECC Protection

### 3.8.1 Overview of Data Path ECC Protection

On-Chip Data Protection feature provides ECC protection for transmit and receive data paths, where feasible. The ECC bits are generated for the data path based on ECC description provided in “[AXI Interface ECC Protection for Write and Read Data](#)” on page 198.

### 3.8.2 Description Data Path ECC Protection

At transmit side, the data from AXI/AHB interface is protected until it reaches the shift control logic.

At receive side, the data from serial interface is protected from the serial shift register until it reaches the AHB Subordinate Interface, or AXI Manager Interface.



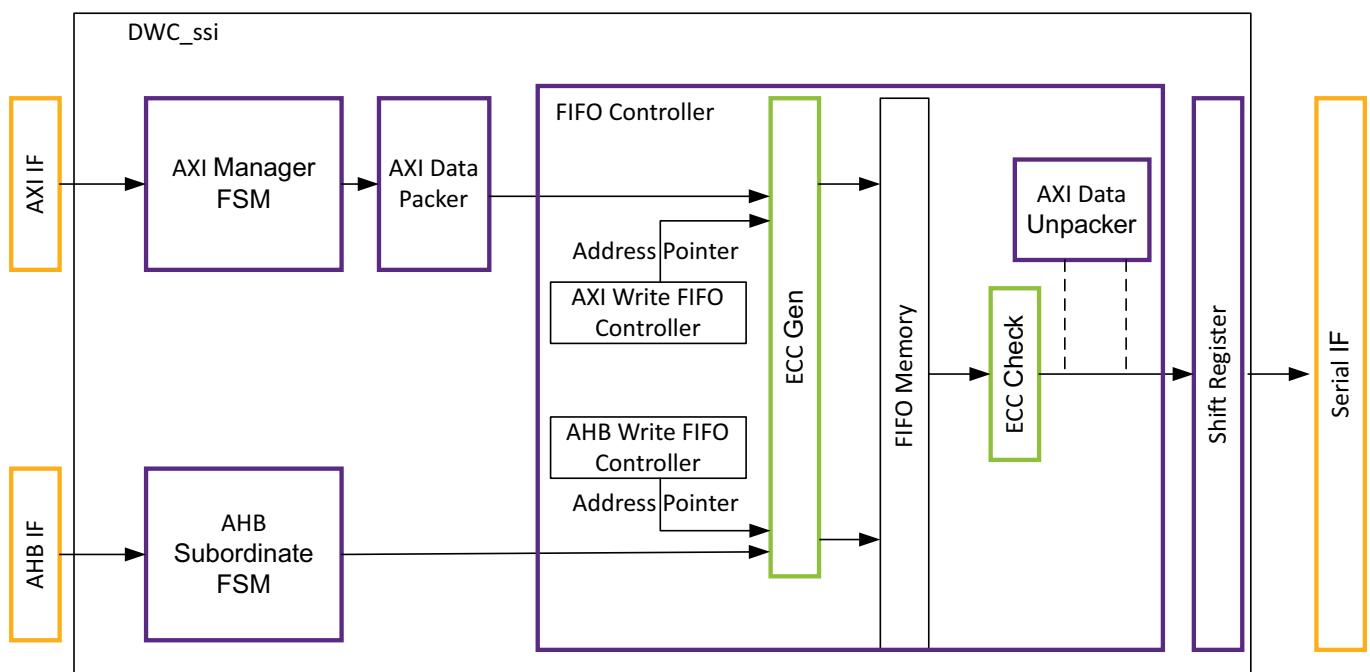
**Note** The AXI interface is used in internal DMA mode of operation, otherwise AHB subordinate interface is used for data transmission or reception. Only one of them can be active at a certain point.

### 3.8.3 Transmit Data Path ECC Protection

For data transmission the data can come from AXI or AHB subordinate interface based on internal programming. [Figure 3-6](#) describes the locations where ECC generators and ECC checkers are placed in the transmit data path.

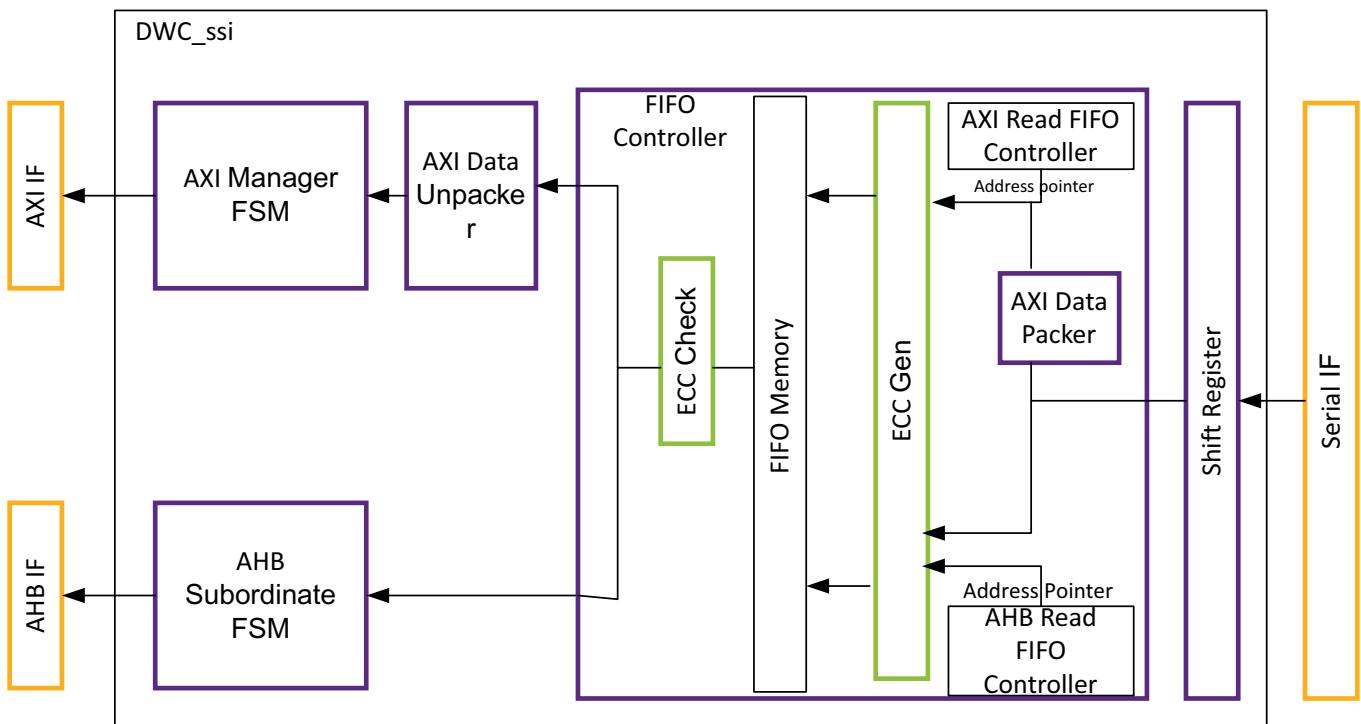
Following are the places where ECC generators and checkers are present in the transmit data path:

- ECC is generated when AXI or AHB interface has data ready to shift inside the FIFO.
  - There are two ECC generators for AXI transmit path. One for the lower 32 bits and one for the upper 32 bits.
  - For AHB transmit path only one ECC generator is used which includes 32 bits data and the address bits.
- ECC is checked when the data is loaded into the internal shift register.
  - Just like the ECC generator, there are two ECC checker modules present for the AXI transmit path and one for the AHB transmit path.

**Figure 3-6 Data Path ECC Protection for Transmit Data Path**

### 3.8.4 Receive Path ECC Protection

Figure 3-7 describes the locations where ECC generators and ECC checkers are placed in the receive data path.

**Figure 3-7 Data Path ECC Protection for Receive Data Path**

Following are the places where ECC generators and checkers are available in the receive data path:

- ECC is generated when serial interface has data ready to shift inside the FIFO.  
There are two ECC generator modules present for the AXI Receive data path and one for the AHB receive data path.
- ECC is checked when the data is about to be sent in AHB/AXI interface.  
There are two separate ECC checker modules for the AXI receive path and one for the AHB receive data path.

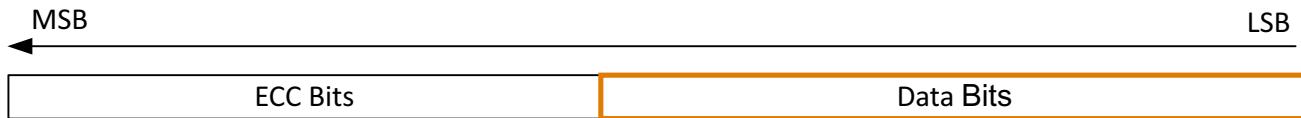
### 3.8.5 Error Injection in Data Path ECC Protection

Each ECC generator supports control to inject ECC error. When enabled, the ECC bits/data bits of incoming data is flipped until the ECC correctable or uncorrectable error is detected by the IP. Application must set the DP\_EI\_EN and ECC\_EI\_TYPE fields of the SSI Safety Control (SAFETYCR) register after the data path ECC interrupt is detected; the hardware clears the same. The user have capability to select in which error should be inserted using the SEL field of the Safety Control (SAFETYCR) register. Using this field you can insert the error in data or ECC check-bits. Following is the organization of the for the ECC data for the error injection:

- [31:0] is for data bits
- Rest of bits are 7 ECC bits

There are two ECC generators in case of Internal DMA operation, this can be selected using the AXI\_EI\_SEL bit in the SAFETYCR register.

The same can be observed in [Figure 3-8](#):

**Figure 3-8 Data Organization for ECC Data Path Protection**

There are four possible ECC error injection methods for each path, following is the programming, which should be followed to inject ECC error in each path:

#### For Internal DMA Mode:

- If the transfer mode is set to Write, then Error is injected at AXI interface ECC generator and error is generated at ECC checker present before shift control logic.
- If the transfer mode is set to Read, then Error is injected at ECC generator present after shift control logic and error is generated at the ECC checker present in the AXI interface.

#### For Normal Operation Mode:

- If the transfer mode is set to Transmit Only, then Error is injected at AHB subordinate interface ECC generator and error is generated at ECC checker present before shift control logic.
- If the transfer mode is set to Receive Only, then Error is injected at ECC generator present after shift control logic and error is generated at the ECC checker present at AHB subordinate interface.

After the generator inserts error in the generated ECC data, the subsequent ECC checker should detect the mismatch and report the error status in SSI Safety Interrupt status (SAFETYISR) register.

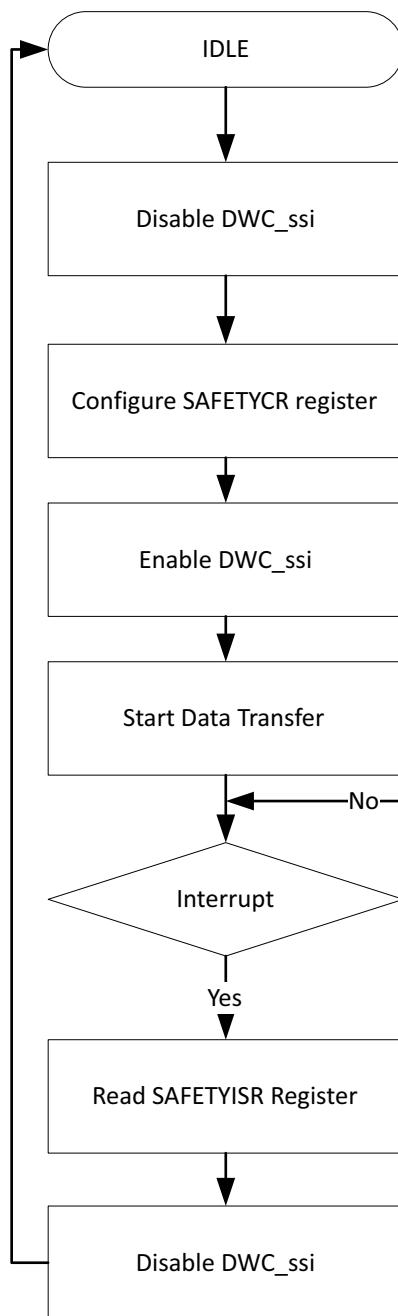
For example, if DWC\_ssi needs to insert an correctable error for Internal DMA SPI write operation, follow the sequence of operation:

1. Application sets the EI\_EN field to 1'b1 and EI\_TYPE field to 2'b00 of the SSI Safety Control () register.
2. Application sets the ECC\_EI\_TYPE to 0 for correctable error insertion in SSI Control (SAFETYCR) register.
3. Application sets the EI\_SEL to select which bit the error should be inserted in SSI Control () register.
4. Application Enables DWC\_ssi by setting SSIENR register to 1.
5. The ECC generator inserts error in the generated for the data.
6. As the data traverses through multiple paths, the subsequent checkers detects the ECC mismatch and reports error status in SSI Safety Interrupt (SAFETYISR) register and generated `ssi_sfty_dp_re_intr(_n)/ssi_sfty_dp_ecc_ce_intr(_n)` interrupt.
7. After the interrupt is detected DWC\_ssi clears the EI\_EN bit in the SAFETYCR register.

The Programming flow or data path error injection is also shown in [Figure 3-9](#).



- For Error insertion the transfer mode field (CTRLR0 . TMOD) should not be set to 2'b00 (Transmit and Receive).
- Error insertion is not applicable for XIP transfers.

**Figure 3-9 Programming Flow for Error Injection**

### 3.8.6 Registers Related to On-Chip Data Protection with ECC

The following registers are related to On-Chip Data Protection with ECC:

- SAFETYCR
- SAFETYISR

- SAFETYICR

For more information about these registers, see the *Registers Description* chapter.

### 3.8.7 Signals Related to On-Chip Data Protection with ECC

The following signals are related to On-Chip Data Protection with ECC:

- `ssi_sfty_dp_ecc_ue_intr(_n)` when individual interrupts are selected.
- `ssi_sfty_dp_ecc_ce_intr(_n)` when individual interrupts are selected.
- `ssi_sfty_ue_intr(_n)` when combined interrupts are selected.
- `ssi_sfty_re_intr(_n)` when combined interrupts are selected.

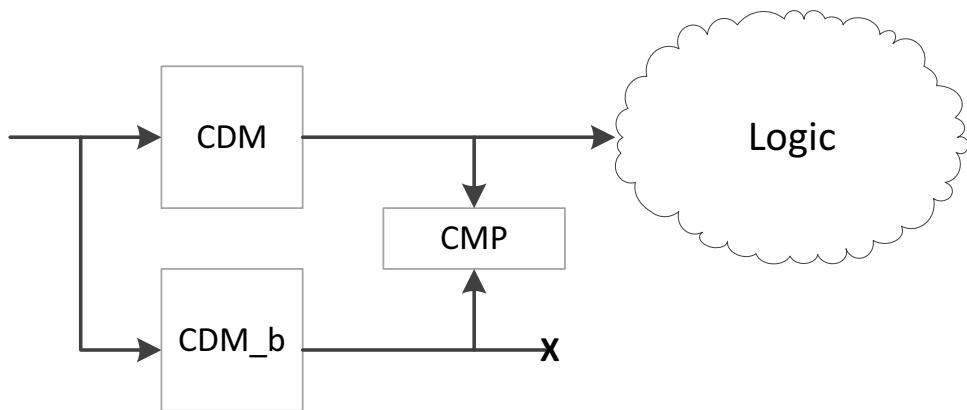
For more information about these signals, see the *Signals Description* chapter.

## 3.9 Register Space Protection

### 3.9.1 Overview of Register Space Protection

The Register Space Protection feature is based on a simple form of hardware redundancy where the set of configuration registers is duplicated. The original set is the one driving the logic where its copy is used for consistency and it is not connected to any logic as represented in [Figure 3-10](#).

**Figure 3-10 Register Space Protection Architecture**



### 3.9.2 Description of Register Space Protection

Both register sets are written with the same values when any one of the register is programmed. A state machine runs periodically traversing the register space, and comparing the values from both register sets. If difference is detected, it is signaled to the application layer using dedicated interrupt `ssi_sfty_rs_intr(_n)`.

### 3.9.3 Error Injection for Register Space Protection

Error injection capability is available for the register duplication logic. This can be done using dedicated programming bit in SSI Safety Control (SAFETYCR) register. Using the programming options error can be injected in each of the registers, which are protected through this logic. The SEL field is used to select the register in which error should be injected. User can select in which register (original or duplicate) the error should be inserted using LRD\_EI\_TYPE field in SAFETYCR Register; 1: Error insertion in the duplicated register and 0: Error will be inserted in the original register. [Table 3-9](#) lists all the registers which are protected using this feature and field decode in EI\_SEL programming field of the Safety Control Register SAFETYCR register.

**Table 3-9 Register Protected with Duplication and Field Decode Values**

Register Code	Register Name
0	CTRLR0: Control 0 Register
1	CTRLR1: Control 1 Register

Register Code	Register Name
2	SER: Subordinate Enable Register
3	BAUDR: Baud Rate Register
4	DMACR: DMA Control Register
5	RX_SAMPLE_DELAY: RX Sample Delay Register
6	SPI_CTRLR0: SPI Control 0 register
7	XIP_CTRL: XIP Control Register
8	XIP_SER: XIP Subordinate Enable Register
9	SAFETYCR: Safety Control Register
10	SSIENR: SSI Enable Register
11	MWCR: Microwire Control Register
12	SLVFSMTOCR: Subordinate FSM Time out Register
13	SSIIFSMTOCR: SSI FSM Time out Register
14	AXIIFSMTOCR: AXI FSM Time out Register
15	AXIAWLEN: AWLEN Register
16	AXIARLEN: ARLEN Register
17	AXIAR0: AXIAR0 Register
18	AXIAR1: AXIAR0 Register
19	SPIDR: SPI Device Register
20	SPIAR: Serial Device Address Register
21	TXFTLR: Transmit FIFO Threshold Register
22	RXFTLR: Receive FIFO Threshold Register
23	TXD_DRIVE_EDGE: Transmit Data Drive Edge Register
24	SPI_CTRLR1: SPI Control Register 1

### 3.9.4 Registers Related to Register Space Protection

The following registers are related to Register Space Protection:

- SAFETYCR
- SAFETYISR
- SAFETYICR

For more information about these registers, see the *Registers Description* chapter.

### 3.9.5 Signals Related to Register Space Protection

The following signals are related to Register Space Protection:

- `ssi_sfty_rs_intr(_n)`: When individual interrupts are selected
- `ssi_sfty_ue_intr(_n)`: When combined interrupts are selected.

For more information about these signals, see the *Signals Description* chapter.



# 4

## Parameter Descriptions

---

This chapter details all the configuration parameters. You can use the coreConsultant GUI configuration reports to determine the complete configuration 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 configuration options for this component.

- “Top Level Parameters” on page 224
- “Top Level Parameters / Software Interface Configuration Parameters” on page 228
- “Top Level Parameters / External Configuration Parameters” on page 229
- “Top Level Parameters / Storage Configuration Parameters” on page 230
- “Safety Features Parameters” on page 232
- “Internal DMA Parameters” on page 235
- “Enhanced SPI Parameters” on page 236
- “Register Default Value Parameters” on page 240
- “Synchronization Parameters” on page 241

## 4.1 Top Level Parameters

**Table 4-1 Top Level Parameters**

Label	Description
Device Configuration	
Enable Safety Features?	<p>Configures if DWC_ssi implements safety features targeted for automotive applications. Enabling this bit will include following features. Safety Package 1:</p> <ul style="list-style-type: none"> <li>■ FSM one-hot checking feature</li> <li>■ FSM time out feature</li> <li>■ Internal Data path parity</li> <li>■ Configuration register duplication</li> <li>■ Parity signals for AXI read and write address channels and data channels</li> <li>■ Parity signals for AHB Read data signal</li> </ul> <p>Safety Configuration 2:</p> <ul style="list-style-type: none"> <li>■ Logic Duplication for FSM, Clock Generator and Data shifter modules</li> <li>■ FSM time out feature</li> <li>■ Data Path Protection using ECC</li> <li>■ FIFO controller module protection using TMR registers</li> <li>■ Configuration register duplication</li> <li>■ Parity signals for AXI read and write address channels and data channels</li> <li>■ Parity signals for AHB Read data signal</li> </ul> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ Disable (0)</li> <li>■ Safety Package 1 (1)</li> <li>■ Safety Package 2 (2)</li> </ul> <p><b>Default Value:</b> Disable</p> <p><b>Enabled:</b> If DWC-SSI-SAFETY license exist</p> <p><b>Parameter Name:</b> SSIC_SAFETY_PKG_EN</p>
SPI Target Bridge Mode?	<p>Configures whether the core works as a Bridge between SPI and AHB interfaces with SPI being a primary interface. In this configuration, all the SPI transfers will be directly converted in AHB transfers on secondary interface of AHB Manager.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> If DWC-SSI-SPI-SLV-BRIDGE license exist</p> <p><b>Parameter Name:</b> SSIC_SPI_BRIDGE</p>
Address offset for AHB Manager Interface	<p>Address offset for the SPI bridge operations. The lower bits of the address will be picked from SPI address sent by SPI controller.</p> <p><b>Values:</b> 0x0, ..., 0xffffffff</p> <p><b>Default Value:</b> 0x0</p> <p><b>Enabled:</b> SSIC_SPI_BRIDGE==1</p> <p><b>Parameter Name:</b> SSIC_AHB_ADDR_OFFSET</p>

Label	Description
Serial controller or target configuration	<p>Configures the device as a controller or a target serial peripheral.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ Serial Target (0)</li> <li>■ Serial Controller (1)</li> <li>■ Programmable (2)</li> </ul> <p><b>Default Value:</b> (SSIC_SPI_BRIDGE == 1) ? 0: 1</p> <p><b>Enabled:</b> SSIC_SPI_BRIDGE==0</p> <p><b>Parameter Name:</b> SSIC_IS_MASTER</p>
Select Controller SPI mode	<p>Configures whether DWC_ssi works in the Standard, Dual, Quad, Octal or Dual Octal SPI Mode.</p> <ul style="list-style-type: none"> <li>■ Dual Mode: Width of txd and rxd signals are 2 bits.</li> <li>■ Quad Mode: Width of txd and rxd signals are 4 bits.</li> <li>■ Octal Mode: Width of txd and rxd signals are 8 bits.</li> <li>■ Dual Octal Mode: Width of txd and rxd signals are 16 bits.</li> </ul> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ Standard SPI Mode (0)</li> <li>■ SPI Dual Mode (1)</li> <li>■ SPI Quad Mode (2)</li> <li>■ SPI Octal Mode (3)</li> <li>■ SPI Dual Octal Mode (4)</li> </ul> <p><b>Default Value:</b> Standard SPI Mode</p> <p><b>Enabled:</b> ((SSIC_DFLT_FRF==0)    (SSIC_HC_FRF==0)) &amp;&amp; (SSIC_IS_MASTER!=0)</p> <p><b>Parameter Name:</b> SSIC_SPI_MODE</p>
Select Target SPI mode	<p>Configures whether DWC_ssi works in the Standard, Dual, Quad or Octal SPI Mode in Target configuration.</p> <ul style="list-style-type: none"> <li>■ Dual Mode: Width of txd and rxd signals are 2 bits.</li> <li>■ Quad Mode: Width of txd and rxd signals are 4 bits.</li> <li>■ Octal Mode: Width of txd and rxd signals are 8 bits.</li> </ul> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ Standard SPI Mode (0)</li> <li>■ SPI Dual Mode (1)</li> <li>■ SPI Quad Mode (2)</li> <li>■ SPI Octal Mode (3)</li> </ul> <p><b>Default Value:</b> (SSIC_SPI_BRIDGE == 1) ? 2: 0</p> <p><b>Enabled:</b> ((SSIC_DFLT_FRF==0)    (SSIC_HC_FRF==0)) &amp;&amp; (SSIC_IS_MASTER!=1)</p> <p><b>Parameter Name:</b> SSIC_SLV_SPI_MODE</p>

Label	Description
Include Enhanced Clock Ratio Architecture?	<p>Configures the device to include new architecture for Transmit and Receive FIFO. This enables the device to work on clock ratios of 4 and 6 between ssi_clk and sclk_in signals.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_IS_MASTER!=1</p> <p><b>Parameter Name:</b> SSIC_ENH_CLK_RATIO</p>
Number of chip select lines	<p>Configures the number of chip select lines from the DWC_ssi controller.</p> <p><b>Values:</b> 1, ..., 16</p> <p><b>Default Value:</b> 1</p> <p><b>Enabled:</b> SSIC_IS_MASTER!=0</p> <p><b>Parameter Name:</b> SSIC_NUM_SLAVES</p>
Include Programmable RXD Sample Logic	<p>Includes the logic to allow a 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. There are two configuration options:</p> <ul style="list-style-type: none"> <li>- 1 Use Positive edge of ssi_clk for RXD sampling</li> <li>- 2 Use both positive and negative edge of ssi_clk for RXD sampling.</li> </ul> <p>Positive and negative edge sampling increases number of sampling points within single sclk_out clock. Depending on the frequency requirement customer can choose type of sampling.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Positive Edge Sampling (1)</li> <li>■ Use Both Positive and Negative Edges (2)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_IS_MASTER!=0</p> <p><b>Parameter Name:</b> SSIC_HAS_RX_SAMPLE_DELAY</p>
Maximum RXD Sample Delay	<p>Defines the maximum number of ssi_clk cycles that can be used to delay the sampling of the rxd input. For each value, two registers are added to the design logic.</p> <p><b>Values:</b> 4, ..., 255</p> <p><b>Default Value:</b> 4</p> <p><b>Enabled:</b> SSIC_HAS_RX_SAMPLE_DELAY!=0</p> <p><b>Parameter Name:</b> SSIC_RX_DLY_SR_DEPTH</p>
Peripheral ID code	<p>Specifies the individual peripheral Identification code.</p> <p><b>Values:</b> 0x0, ..., 0xffffffff</p> <p><b>Default Value:</b> 0xffffffff</p> <p><b>Enabled:</b> Always</p> <p><b>Parameter Name:</b> SSIC_ID</p>

Label	Description
Enable boot mode for DWC_ssi?	<p>Specifies whether the boot mode is enabled for DWC_ssi. In this mode, the default value of the SSI_EN register is 1, which means that DWC_ssi is enabled on reset. You can use the SSI_DFLT_* parameters to set the default value of the control registers on reset.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"><li>■ No (0x0)</li><li>■ Yes (0x1)</li></ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> Always</p> <p><b>Parameter Name:</b> SSIC_BOOT_MODE_EN</p>

## 4.2 Top Level Parameters / Software Interface Configuration Parameters

**Table 4-2 Top Level Parameters / Software Interface Configuration Parameters**

Label	Description
Software Interface Configuration	
Programming Interface Type	<p>Programming Interface Type for DWC_ssi.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ APB (0)</li> <li>■ AHB (1)</li> </ul> <p><b>Default Value:</b> AHB</p> <p><b>Enabled:</b> SSIC_SPI_BRIDGE==1</p> <p><b>Parameter Name:</b> SSIC_SLVIF_TYPE</p>
APB Data Bus Width	<p>Width of APB data bus</p> <p><b>Values:</b> 8, 16, 32</p> <p><b>Default Value:</b> 32</p> <p><b>Enabled:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Parameter Name:</b> APB_DATA_WIDTH</p>

## 4.3 Top Level Parameters / External Configuration Parameters

**Table 4-3 Top Level Parameters / External Configuration Parameters**

Label	Description
External Configuration	
Include DMA Interface?	<p>Configures if DWC_ssi has DMA interface.</p> <p>1- DMA interface will be configured to have external handshaking signals to interface with DMA controller.</p> <p>2- Internal DMA will be configured, which will perform the transfers using AXI manager interface.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No DMA (0)</li> <li>■ External DMA (1)</li> <li>■ Internal DMA (2)</li> </ul> <p><b>Default Value:</b> No DMA</p> <p><b>Enabled:</b> SSIC_SPI_BRIDGE==0</p> <p><b>Parameter Name:</b> SSIC_HAS_DMA</p>
Configure interrupt pinout	<p>Selects interrupt-related signals that must appear as output of the design. This parameter enables you to select a single combined interrupt (the logical OR of all DWC_ssi interrupt outputs) or an individual interrupt to appear as a separate output pin on the macrocell. When configured as a controller, there are six individual interrupts. When configured as a target, there are five individual interrupts.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ Individual Interrupts (0)</li> <li>■ Combined Interrupt (1)</li> </ul> <p><b>Default Value:</b> Individual Interrupts</p> <p><b>Enabled:</b> Always</p> <p><b>Parameter Name:</b> SSIC_INTR_IO</p>
Active interrupt level	<p>Configures the active level of the output interrupt lines.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ Active Low (0)</li> <li>■ Active High (1)</li> </ul> <p><b>Default Value:</b> Active Low</p> <p><b>Enabled:</b> Always</p> <p><b>Parameter Name:</b> SSIC_INTR_POL</p>

## 4.4 Top Level Parameters / Storage Configuration Parameters

**Table 4-4 Top Level Parameters / Storage Configuration Parameters**

Label	Description
Storage Configuration	
DWC_ssi has External RAM Support?	<p>When this parameter is selected, internal FIFOs will be replaced by external Dual Port RAM. DWC_ssi will provide separate interface to communicate with external RAM.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SPI_BRIDGE==0 &amp;&amp; SSIC_HAS_DMA!=2</p> <p><b>Parameter Name:</b> SSIC_HAS_EXT_RAM</p>
Two Separate RAM interface for Transmit and Receive FIFO?	<p>Selects if DWC_ssi support two external Dual Port RAM to store both TX and RX data at same time. Enable this parameter if you want to perform one of the following transfer types: 1. For SPI/SSP transfers where TMOD is set to 0 (Receive and transmit). 2. For Microwire non-sequential transfer.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> Yes</p> <p><b>Enabled:</b> SSIC_HAS_EXT_RAM == 1</p> <p><b>Parameter Name:</b> SSIC_HAS_TX_RX_EN</p>
External RAM Depth	<p>External RAM Depth. Applicable only when SSIC_HAS_TX_RX_EN is set to 0.</p> <p><b>Values:</b> 8, 16, 32, 64, 128, 256</p> <p><b>Default Value:</b> 32</p> <p><b>Enabled:</b> SSIC_HAS_TX_RX_EN == 0</p> <p><b>Parameter Name:</b> SSIC_RAM_DEPTH</p>
Transmit FIFO buffer depth	<p>Configures the depth of the transmit FIFO buffer.</p> <p><b>Values:</b> 8, 16, 32, 64, 128, 256</p> <p><b>Default Value:</b> (SSIC_HAS_TX_RX_EN == 1) ? 32: SSIC_RAM_DEPTH</p> <p><b>Enabled:</b> SSIC_HAS_TX_RX_EN == 1</p> <p><b>Parameter Name:</b> SSIC_TX_FIFO_DEPTH</p>
Receive FIFO buffer depth	<p>Configures the depth of the receive FIFO buffer.</p> <p><b>Values:</b> 8, 16, 32, 64, 128, 256</p> <p><b>Default Value:</b> (SSIC_HAS_DMA == 2) ? SSIC_TX_FIFO_DEPTH : (SSIC_HAS_TX_RX_EN == 1) ? 32: SSIC_RAM_DEPTH</p> <p><b>Enabled:</b> SSIC_HAS_TX_RX_EN == 1 &amp;&amp; SSIC_HAS_DMA != 2</p> <p><b>Parameter Name:</b> SSIC_RX_FIFO_DEPTH</p>

Label	Description
XIP Receive FIFO buffer depth	<p>Configures the depth of the XIP receive FIFO buffer. This FIFO is used to store only intermediate data for XIP transfers. The data is stored in the FIFO when AHB subordinate interface gets WAIT/BUSY cycles in-between. If you do not expect any wait/busy states between the transfers, then configure this to a minimum value.</p> <p>The same FIFO is used for data storage and synchronization purpose in transfers where RXDS signalling is enabled. In case when RXDS signalling is used 16 FIFO locations are required for synchronization, rest of the locations will be used as data storage. User should consider this while deciding the FIFO depth in case if SSIC_HAS_RXDS is set to 1.</p> <p><b>Values:</b> 4, 8, 16, 32, 64, 128  <b>Default Value:</b> 32  <b>Enabled:</b> SSIC_CONCURRENT_XIP_EN==1  <b>Parameter Name:</b> SSIC_XIP_RX_FIFO_DEPTH</p>
XIP Transmit FIFO buffer depth	<p>Configures the depth of the XIP transmit FIFO buffer.</p> <p><b>Values:</b> 4, 8, 16, 32, 64, 128  <b>Default Value:</b> (SSIC_XIP_WRITE_EN == 1) ? 32: 4  <b>Enabled:</b> SSIC_CONCURRENT_XIP_EN==1 &amp;&amp; SSIC_XIP_WRITE_EN==1  <b>Parameter Name:</b> SSIC_XIP_TX_FIFO_DEPTH</p>

## 4.5 Safety Features Parameters

**Table 4-5 Safety Features Parameters**

Label	Description
Safety Features	
AHB Subordinate Interface Parity Protection?	<p>Configures DWC_ssi to include the AHB Subordinate interface Parity checking and generation mechanism.</p> <ul style="list-style-type: none"> <li>■ 0 - No Parity checker</li> <li>■ 1 - Check parity of data</li> <li>■ 2 - Check parity for both address and data</li> </ul> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Data Parity (1)</li> <li>■ Address and Data Parity (2)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SAFETY_PKG_EN!=0 &amp;&amp; SSIC_SLVIF_TYPE==1</p> <p><b>Parameter Name:</b> SSIC_SLVIF_PAR_EN</p>
Include Safety Protection feature on the AXI Interface?	<p>This parameter allows user to Enable the Safety feature on the AXI Manager Interface.</p> <ul style="list-style-type: none"> <li>■ 0: No AXI Safety Features</li> <li>■ 1: Parity Protection for Data and Control Signals</li> <li>■ 2: ECC - for AXI Read and Write Data signals; Parity Protection for rest of the control signals.</li> </ul> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No AXI Safety (0)</li> <li>■ Parity Protection (1)</li> <li>■ ECC on data signals &amp; Parity on rest (2)</li> </ul> <p><b>Default Value:</b> No AXI Safety</p> <p><b>Enabled:</b> SSIC_SAFETY_PKG_EN!=0 &amp;&amp; SSIC_HAS_DMA==2</p> <p><b>Parameter Name:</b> SSIC_AXI_SAFETY_EN</p>
DWC_ssi Parity Type	<p>AHB Interface, AXI Interface and Data Path Parity type selection - Even or Odd Parity.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ Even (0)</li> <li>■ Odd (1)</li> </ul> <p><b>Default Value:</b> Even</p> <p><b>Enabled:</b> SSIC_SAFETY_PKG_EN!=0</p> <p><b>Parameter Name:</b> SSIC_PAR_TYPE</p>

Label	Description
Include Parity for AXI Valid and Ready Signals?	<p>AXI Interface Ready and Valid Parity feature Enable/Disable.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> (SSIC_AXI_SAFETY_EN == 2) ? 1: 0</p> <p><b>Enabled:</b> SSIC_AXI_SAFETY_EN!=0</p> <p><b>Parameter Name:</b> SSIC_READY_VALID_PAR_EN</p>
AXI Interface ECC Granularity Type?	<p>AXI Interface ECC Resolution Type. Which is used to select the Granularity of ECC Checkbits generation.</p> <ul style="list-style-type: none"> <li>■ 1: 32 bit Granularity ECC Computation</li> <li>■ 2: 64 bit Granularity ECC Computation</li> </ul> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 32-Bit ECC (1)</li> <li>■ 64-Bit ECC (2)</li> </ul> <p><b>Default Value:</b> 32-Bit ECC</p> <p><b>Enabled:</b> SSIC_AXI_SAFETY_EN == 2</p> <p><b>Parameter Name:</b> SSIC_ECC_GRAN_TYPE</p>
Include Register Slice on the AR Channel Safety Protection feature?	<p>This parameter allows user to enable/disable the Register Slice to ease the timing on the AR Channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> (SSIC_AXI_SAFETY_EN == 2) ? 1: 0</p> <p><b>Enabled:</b> SSIC_AXI_SAFETY_EN==2</p> <p><b>Parameter Name:</b> SSIC_AR_REGSLICE_EN</p>
Include Register Slice on the AW Channel Safety Protection feature?	<p>This parameter allows user to enable/disable the Register Slice to ease the timing on the AW Channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> (SSIC_AXI_SAFETY_EN == 2) ? 1: 0</p> <p><b>Enabled:</b> SSIC_AXI_SAFETY_EN==2</p> <p><b>Parameter Name:</b> SSIC_AW_REGSLICE_EN</p>
Include Register Slice on the W Channel Safety Protection feature?	<p>This parameter allows user to enable/disable the Register Slice to ease the timing on the W Channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> (SSIC_AXI_SAFETY_EN == 2) ? 1: 0</p> <p><b>Enabled:</b> SSIC_AXI_SAFETY_EN==2</p> <p><b>Parameter Name:</b> SSIC_W_REGSLICE_EN</p>

Label	Description
Include Register Slice on the R Channel Safety Protection feature?	<p>This parameter allows user to enable/disable the Register Slice to ease the timing on the R Channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> (SSIC_AXI_SAFETY_EN == 2) ? 1: 0</p> <p><b>Enabled:</b> SSIC_AXI_SAFETY_EN==2</p> <p><b>Parameter Name:</b> SSIC_R_REGSLICE_EN</p>
Include Register Slice on the B Channel Safety Protection feature?	<p>This parameter allows user to enable/disable the Register Slice to ease the timing on the B Channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> (SSIC_AXI_SAFETY_EN == 2) ? 1: 0</p> <p><b>Enabled:</b> SSIC_AXI_SAFETY_EN==2</p> <p><b>Parameter Name:</b> SSIC_B_REGSLICE_EN</p>

## 4.6 Internal DMA Parameters

**Table 4-6 Internal DMA Parameters**

Label	Description
AXI Interface Parameters	
AXI Interface Type	<p>This parameter selects the type of AXI interface</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ AXI3 (0)</li> <li>■ AXI4 (1)</li> </ul> <p><b>Default Value:</b> AXI3</p> <p><b>Enabled:</b> SSIC_HAS_DMA==2</p> <p><b>Parameter Name:</b> SSIC_AXI_INTF_TYPE</p>
AXI Interface Data Width	<p>Data width for AXI Interface</p> <p><b>Values:</b> 32, 64</p> <p><b>Default Value:</b> 32</p> <p><b>Enabled:</b> SSIC_HAS_DMA==2</p> <p><b>Parameter Name:</b> SSIC_AXI_DW</p>
AXI Interface Address Width	<p>Address width for AXI Interface</p> <p><b>Values:</b> 32, ..., 64</p> <p><b>Default Value:</b> 32</p> <p><b>Enabled:</b> SSIC_HAS_DMA==2</p> <p><b>Parameter Name:</b> SSIC_AXI_AW</p>
AXI ID Width	<p>AXI ID width</p> <p><b>Values:</b> 1, ..., 12</p> <p><b>Default Value:</b> 6</p> <p><b>Enabled:</b> SSIC_HAS_DMA==2</p> <p><b>Parameter Name:</b> SSIC_AXI_IDW</p>
AXI Burst Length Width	<p>AXI Burst length width</p> <p><b>Values:</b> 4, 5, 6, 7, 8</p> <p><b>Default Value:</b> 4</p> <p><b>Enabled:</b> SSIC_HAS_DMA==2</p> <p><b>Parameter Name:</b> SSIC_AXI_BLW</p>

## 4.7 Enhanced SPI Parameters

**Table 4-7 Enhanced SPI Parameters**

Label	Description
Enhanced SPI Parameters	
Enable Enhanced SPI I/O mapping	<p>Configures whether the I/O Mapping can be hardcoded within DWC_ssi. When this parameter is configured, the RXD[1] signal is used to sample the incoming data during an SPI standard mode of operation in Controller configuration and txd[1] is used to transmit data in SPI standard mode of operation in Target configuration.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SPI_MODE!=0    SSIC_SLV_SPI_MODE!=0</p> <p><b>Parameter Name:</b> SSIC_IO_MAP_EN</p>
Enable Dynamic Wait State feature for SPI?	<p>Configures Dynamic Wait State feature for SPI transfers.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SPI_MODE!=0 &amp;&amp; SSIC_HAS_EXT_RAM==0</p> <p><b>Parameter Name:</b> SSIC_SPI_DYN_WS_EN</p>
Include Loop back clock for RXD signal?	<p>Specifies whether to include the logic to sample read data using the loop back clock signal.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> (SSIC_SPI_MODE!=0) &amp;&amp; ((SSIC_DFLT_FRF==0 &amp;&amp; SSIC_DFLT_SCPH == 0 &amp;&amp; SSIC_DFLT_SCPOL == 0)    (SSIC_HC_FRF==0))</p> <p><b>Parameter Name:</b> SSIC_HAS_CLK_LOOP_BACK</p>
Enable Clock Stretching in Enhanced SPI Mode	<p>Configures DWC_ssi to support clock stretching in enhanced SPI transfers.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SPI_MODE!=0</p> <p><b>Parameter Name:</b> SSIC_CLK_STRETCH_EN</p>

Label	Description
Include DDR transfers in SPI frame format?	<p>Includes the logic to support DDR operations in SPI mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SPI_MODE != 0 &amp;&amp; SSIC_IS_MASTER!=0</p> <p><b>Parameter Name:</b> SSIC_HAS_DDR</p>
Include data strobe signal for rxds line?	<p>Specifies whether to include the logic to sample read data using the read data strobe signal.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_HAS_DDR ==1</p> <p><b>Parameter Name:</b> SSIC_HAS_RXDS</p>
Include Data Mask Signal for data transfers in SPI mode?	<p>Indicates whether the data mask signal must be included on the SPI interface. When active, the data mask signal (txd_dm) and the data mask output enable signal (txd_dm_oe_n) are added to the SPI interface. The data mask signal is used to mask the write data on an SPI interface.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_HAS_DDR ==1</p> <p><b>Parameter Name:</b> SSIC_SPI_DM_EN</p>
Enable Hyperbus Transfer mode?	<p>Configures whether DWC_ssi supports Hyperbus transfers. For more information about the Hyperbus transfer feature, see the "Hyperbus Protocol Support" section in the DWC_ssi Databook.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SPI_MODE &gt;= 3 &amp;&amp; SSIC_HAS_RXDS == 1</p> <p><b>Parameter Name:</b> SSIC_HYPERBUS_EN</p>

Label	Description
Include XIP feature in SPI mode?	<p>Configures whether DWC_ssi supports XIP operations in the SPI mode. When this parameter is set to 1, an extra side-band signal, xip_en, is included on the AHB interface and this signal decides if the operation is a register read-write or an XIP transfer.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SPI_MODE != 0 &amp;&amp; SSIC_SPI_DYN_WS_EN == 0</p> <p><b>Parameter Name:</b> SSIC_XIP_EN</p>
Include XIP Write feature in SPI mode?	<p>Configures DWC_ssi to include XIP Write Feature. When this parameter is set to 1, XIP write transfer is supported in SPI interface.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_XIP_EN == 1 &amp;&amp; SSIC_HAS_EXT_RAM == 0</p> <p><b>Parameter Name:</b> SSIC_XIP_WRITE_EN</p>
Include External Chip Select for XIP transfers?	<p>External Chip Select Signal for XIP transfers. When this parameter is set to 1 an external target select signal slave_sel will be added on interface which will be used to select target for the XIP transfers.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_XIP_EN == 1</p> <p><b>Parameter Name:</b> SSIC_XIP_EXT_SSEL_EN</p>
Enable Concurrent XIP transfer mode?	<p>Configures whether the core supports concurrent XIP and non-XIP operations in SPI Mode.</p> <p>If this parameter is set to 1, DWC_ssi performs concurrent XIP and non-XIP transfers.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_XIP_EN == 1 &amp;&amp; SSIC_HAS_DMA != 2</p> <p><b>Parameter Name:</b> SSIC_CONCURRENT_XIP_EN</p>

Label	Description
Include separate register sets for XIP Write?	<p>Configures DWC_ssi to include separate register sets for XIP Read and Write. When this parameter is set to 1, separate registers need to be programmed for XIP Write.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> (SSIC_XIP_WRITE_EN==1 &amp;&amp; SSIC_CONCURRENT_XIP_EN==1) ? 1 : 0</p> <p><b>Enabled:</b> SSIC_XIP_WRITE_EN==1 &amp;&amp; SSIC_CONCURRENT_XIP_EN==1</p> <p><b>Parameter Name:</b> SSIC_XIP_WRITE_REG_EN</p>
Enable continuous transfer mode in XIP mode?	<p>Configures whether DWC_ssi supports continuous XIP transfers. In continuous XIP transfer mode DWC_ssi keeps the SPI target selected and waits for next AHB request. Once AHB request arrives it resumes the transfer. For more information please refer to section "Continuous transfer mode in XIP".</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_XIP_EN == 1</p> <p><b>Parameter Name:</b> SSIC_XIP_CONT_XFER_EN</p>
Enable time out counter for continuous transfer mode?	<p>Configures the time-out counter during continuous transfer. The counter value will be used to end the continuous transfers if no XIP transfer is detected on AHB interface once the last continuous transfer has ended.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_XIP_CONT_XFER_EN == 1</p> <p><b>Parameter Name:</b> SSIC_CONT_XFER_TIMEOUT_CNT_EN</p>
Enable XIP Pre-fetch feature?	<p>Configures whether DWC_ssi supports XIP pre-fetch feature. If this parameter is set to 1, then DWC_ssi pre-fetches a fixed amount of data from the device and keep it in the FIFO for the next XIP request. If XIP commands are staggered, then this feature helps in fetching the data faster from the device and improves the overall throughput of the system.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> (SSIC_XIP_EN == 1 &amp;&amp; (SSIC_CONCURRENT_XIP_EN == 1    SSIC_HAS_TX_RX_EN == 1) &amp;&amp; (SSIC_HAS_DMA != 2))</p> <p><b>Parameter Name:</b> SSIC_XIP_PREFETCH_EN</p>

## 4.8 Register Default Value Parameters

Table 4-8 Register Default Value Parameters

Label	Description
Register Default Value Parameters	
Default Value Parameters for DWC_ssi registers (for x = 1; x <= n)	<p>Specifies the default of DWC_ssi register fields.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"><li>■ Masked (0x0)</li><li>■ Unmasked (0x1)</li></ul> <p><b>Default Value:</b> 1</p> <p><b>Enabled:</b> Always</p> <p><b>Parameter Name:</b> SSIC_DFLT_*</p>

## 4.9 Synchronization Parameters

**Table 4-9 Synchronization Parameters**

Label	Description
Synchronization Parameters	
Are Programming Interface clock and core Clock synchronous?	<p>Defines if the Programming interface clock is synchronous to the DWC_ssi core clock. If they are synchronous then one of them need not re-time signals across the clock domains.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> Yes</p> <p><b>Enabled:</b> Always</p> <p><b>Parameter Name:</b> SSIC_SYNC_CLK</p>
Generate clock enable input for ssi_clk?	<p>Enables the ssi_clk_en signal to enable data propagation through ssi_clk flip-flops. When disabled, the ssi_clk flip-flops are always enabled.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> No</p> <p><b>Enabled:</b> SSIC_SYNC_CLK==1</p> <p><b>Parameter Name:</b> SSIC_CLK_EN_MODE</p>
hclk to ssi_clk Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the DWC_ssi Programming clock domain to DWC_ssi Core Clock domain (ssi_clk).</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4</p> <p><b>Default Value:</b> 2</p> <p><b>Enabled:</b> SSIC_SYNC_CLK == 0</p> <p><b>Parameter Name:</b> SSIC_H_2_S_SYNC_DEPTH</p>
ssi_clk to hclk Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the DWC_ssi Core Clock domain (ssi_clk) to DWC_ssi Programming clock domain.</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4</p> <p><b>Default Value:</b> 2</p> <p><b>Enabled:</b> SSIC_SYNC_CLK == 0</p> <p><b>Parameter Name:</b> SSIC_S_2_H_SYNC_DEPTH</p>

Label	Description
DWC_ssi Bridge Mode	
Are AHB Manager interface clock and core Clock synchronous?	<p>Defines if the mhclk (AHB Manager interface clock) is synchronous to the DWC_ssi core clock.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> Yes</p> <p><b>Enabled:</b> SSIC_SPI_BRIDGE==1</p> <p><b>Parameter Name:</b> SSIC_AHBM_SYNC_CLK</p>
mhclk to ssi_clk Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the AHB Manager clock domain (mhclk) to DWC_ssi Core Clock domain (ssi_clk).</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4</p> <p><b>Default Value:</b> 2</p> <p><b>Enabled:</b> SSIC_AHBM_SYNC_CLK == 0</p> <p><b>Parameter Name:</b> SSIC_MH_2_S_SYNC_DEPTH</p>
mhclk to programming interface Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the AHB Manager clock domain (mhclk) to DWC_ssi programming interface clock domain.</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4</p> <p><b>Default Value:</b> (SSIC_AHBM_SYNC_CLK == 0 &amp;&amp; SSIC_SYNC_CLK == 1) ? SSIC_MH_2_S_SYNC_DEPTH : ((SSIC_AHBM_SYNC_CLK == 1 &amp;&amp; SSIC_SYNC_CLK == 0) ? SSIC_H_2_S_SYNC_DEPTH : 2)</p> <p><b>Enabled:</b> SSIC_AHBM_SYNC_CLK == 0 &amp;&amp; SSIC_SYNC_CLK == 0</p> <p><b>Parameter Name:</b> SSIC_MH_2_H_SYNC_DEPTH</p>
ssi_clk to mhclk Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the DWC_ssi Core Clock domain (ssi_clk) to AHB Manager clock domain (mhclk).</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4</p> <p><b>Default Value:</b> 2</p> <p><b>Enabled:</b> SSIC_AHBM_SYNC_CLK == 0</p> <p><b>Parameter Name:</b> SSIC_S_2_MH_SYNC_DEPTH</p>

Label	Description
programming interface to mhclk Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the DWC_ssi programming interface clock domain to AHB Manager clock domain (mhclk).</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4</p> <p><b>Default Value:</b> (SSIC_AHBM_SYNC_CLK == 0 &amp;&amp; SSIC_SYNC_CLK == 1) ? SSIC_S_2_MH_SYNC_DEPTH : ((SSIC_AHBM_SYNC_CLK == 1 &amp;&amp; SSIC_SYNC_CLK == 0) ? SSIC_S_2_H_SYNC_DEPTH : 2)</p> <p><b>Enabled:</b> SSIC_AHBM_SYNC_CLK == 0 &amp;&amp; SSIC_SYNC_CLK == 0</p> <p><b>Parameter Name:</b> SSIC_H_2_MH_SYNC_DEPTH</p>
Internal DMA Mode	
Are AXI Manager interface clock and core Clock synchronous?	<p>Defines if the AXI Manager Clock is synchronous to the DWC_ssi core clock.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> Yes</p> <p><b>Enabled:</b> SSIC_HAS_DMA==2</p> <p><b>Parameter Name:</b> SSIC_AXIM_SYNC_CLK</p>
Are AXI Manager interface clock and AHB subordinate interface Clock synchronous?	<p>Defines if the AXI Manager Clock is synchronous to the AHB Subordinate Interface core clock.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ No (0)</li> <li>■ Yes (1)</li> </ul> <p><b>Default Value:</b> ((SSIC_AXIM_SYNC_CLK == 1 &amp;&amp; SSIC_SYNC_CLK == 1)    (SSIC_HAS_DMA!=2)) ? 1 : 0</p> <p><b>Enabled:</b> SSIC_AXIM_SYNC_CLK==0 &amp;&amp; SSIC_SYNC_CLK==0</p> <p><b>Parameter Name:</b> SSIC_AXIM_SLVIF_SYNC_CLK</p>
AXI clock to DWC_ssi Core Clock Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the DWC_ssi AXI clock domain (aclk) to DWC_ssi Core Clock domain (ssi_clk).</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4</p> <p><b>Default Value:</b> 2</p> <p><b>Enabled:</b> SSIC_AXIM_SYNC_CLK==0</p> <p><b>Parameter Name:</b> SSIC_A_2_S_SYNC_DEPTH</p>

Label	Description
DWC_ssi Core Clock to AXI clock Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the DWC_ssi Core Clock domain (ssi_clk) to DWC_ssi AXI clock domain (aclk).</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4  <b>Default Value:</b> 2  <b>Enabled:</b> SSIC_AXIM_SYNC_CLK==0  <b>Parameter Name:</b> SSIC_S_2_A_SYNC_DEPTH</p>
AXI clock to AHB subordinate clock Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the DWC_ssi AXI clock domain (aclk) to DWC_ssi AHB subordinate clock domain (hclk).</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4  <b>Default Value:</b> SSIC_A_2_S_SYNC_DEPTH  <b>Enabled:</b> SSIC_AXIM_SLVIF_SYNC_CLK==0  <b>Parameter Name:</b> SSIC_A_2_H_SYNC_DEPTH</p>
AHB subordinate clock to AXI clock Synchronization Depth	<p>Defines the number of synchronization register stages for signals passing from the DWC_ssi AHB subordinate clock domain (hclk) to DWC_ssi AXI clock domain (aclk).</p> <ul style="list-style-type: none"> <li>■ 2: Two-stage synchronization, both stages positive edge.</li> <li>■ 3: Three-stage synchronization, all stages positive edge.</li> <li>■ 4: Four-stage synchronization, all stages positive edge.</li> </ul> <p><b>Values:</b> 2, 3, 4  <b>Default Value:</b> SSIC_S_2_A_SYNC_DEPTH  <b>Enabled:</b> SSIC_AXIM_SLVIF_SYNC_CLK==0  <b>Parameter Name:</b> SSIC_H_2_A_SYNC_DEPTH</p>

# Signal Descriptions

This chapter details all possible I/O signals in the IP. 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). 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. The presence of the postfix SuperList indicates a list of all possible clocks over all possible configs. Consult coreConsultant report for which clock applies to your specific configuration.
- **Exists:** Name of configuration parameter that populates this signal in your configuration.

- **Power Domain:** Name of power/voltage domain that this signal is part of when power/voltage islands are used. The SINGLE\_DOMAIN value indicates that there are no islands.

The I/O signals are grouped as follows:

- “Serial Interface Signals” on page 247
- “AHB Subordinate Interface Signals” on page 252
- “APB Completer Interface Signals” on page 256
- “DMA Interface Signals” on page 258
- “AHB Manager Interface Signals” on page 260
- “DFT Interface Signals” on page 263
- “Debug Interface Signals” on page 264
- “Single RAM Interface Signals” on page 265
- “Transmit RAM Interface Signals” on page 268
- “Receive RAM Interface Signals” on page 271
- “AXI Manager Interface Signals” on page 274
- “Interrupt Signals” on page 285

## 5.1 Serial Interface Signals

ssi_clk	- ssi_sleep
ssi_rst_n	- txd
ssi_clk_en	- mst_txd
rxd	- slv_txd
mst_rxd	- ssi_oe_n
slv_rxd	- mst_ssi_oe_n
sclk_in	- slv_ssi_oe_n
mst_sclk_in	- txd_dm
ss_in_n	- txd_dm_oe_n
spi_mode_slv	- ss_x_n (for x = 0; x <= SSIC_NUM_SLAVES)
rxds	- dbg_spi_dtr_en
	- sclk_out

Table 5-1 Serial Interface Signals

Port Name	I/O	Description
ssi_clk	I	DWC_ssi core clock signal. <b>Exists:</b> Always <b>Synchronous To:</b> None <b>Registered:</b> N/A <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
ssi_rst_n	I	DWC_ssi core reset signal. <b>Exists:</b> Always <b>Synchronous To:</b> Asynchronous <b>Registered:</b> N/A <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low
ssi_clk_en	I	Optional. Enable signal for ssi_clk domain flip flops. When the ssi_clk input is connected to hclk, 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 hclk and ssi_clk. This signal is used only when SSIC_CLK_EN_MODE = 1. <b>Exists:</b> (SSIC_CLK_EN_MODE==1) <b>Synchronous To:</b> ssi_clk <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> High

Port Name	I/O	Description
rxd[(SERIOW-1):0]	I	<p>Receive data signal. Input data from a serial-controller or serial-target device is received on this line.</p> <p><b>Exists:</b> (SSIC_IS_MASTER!=2)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> SSIC_SPI_BRIDGE==1 ? Yes : No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mst_rxd[(SERIOW-1):0]	I	<p>Controller Interface Receive data signal. Input data from a serial-controller or serial-target device is received on this line.</p> <p><b>Exists:</b> (SSIC_IS_MASTER==2)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
slv_rxd[(SIOW-1):0]	I	<p>Target Interface Receive data signal. Input data from a serial-controller or serial-target device is received on this line.</p> <p><b>Exists:</b> (SSIC_IS_MASTER==2)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
sclk_in	I	<p>Serial bit-rate clock. Generated by the serial bus controller and used by the DWC_ssi target to regulate data transfer. This clock is not 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.</p> <p><b>Exists:</b> (SSIC_IS_MASTER!=1)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> SSIC_ENH_CLK_RATIO==0 ? Yes : No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mst_sclk_in	I	<p>Controller Loop back clock signal, when CTRLR0.CLK_LOOP_EN is set to 1, DWC_ssi uses loop back clock signal to sample the incoming data in rxd line.</p> <p><b>Exists:</b> (SSIC_HAS_CLK_LOOP_BACK==1)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

Port Name	I/O	Description
ss_in_n	I	<p>Chip select input. When configured as a serial target, this signal selects the device. When configured as a serial controller, this signal can be used to inform the system of controller contention on the bus.</p> <p><b>Exists:</b> Always  <b>Synchronous To:</b> Asynchronous  <b>Registered:</b> Yes  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> Low</p>
spi_mode_slv[1:0]	I	<p>SPI Frame format indicator input signal. Indicates the mode in which the SPI transfer is happening.</p> <ul style="list-style-type: none"> <li>■ 00: Standard SPI mode</li> <li>■ 01: SPI Dual Mode</li> <li>■ 10: SPI Quad mode</li> <li>■ 11: SPI Octal mode</li> </ul> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)  <b>Synchronous To:</b> ssi_clk  <b>Registered:</b> No  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> High</p>
rxds[(DM_W-1):0]	I	<p>Read Data strobe signal. In SPI mode, when the SPI_CTRLR0.RXDS_EN bit is set to 1, DWC_ssi uses the Read data strobe (rxds) to capture read data.</p> <p><b>Exists:</b> (SSIC_HAS_RXDS==1)  <b>Synchronous To:</b> Asynchronous  <b>Registered:</b> No  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> N/A</p>
ssi_sleep	O	<p>SSI sleep flag. This signal is asserted when DWC_ssi is disabled. It can be used by the system clock generator/controller module in order to disable the ssi_clk input of DWC_ssi, to reduce power consumption of the system.</p> <ul style="list-style-type: none"> <li>■ 0: Not safe to remove ssi_clk</li> <li>■ 1: Safe to remove ssi_clk</li> </ul> <p><b>Exists:</b> Always  <b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"  <b>Registered:</b> Yes  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> High</p>

Port Name	I/O	Description
txd[(SERIOW-1):0]	O	<p>Transmit data signal. Output data from the serial controller or serial target is transmitted on this line.</p> <p><b>Exists:</b> (SSIC_IS_MASTER!=2)</p> <p><b>Synchronous To:</b> SSIC_ENH_CLK_RATIO==1 ? (SSIC_SPI_BRIDGE==1 ? "sclk_in" : "sclk_in,ssi_clk") : "ssi_clk"</p> <p><b>Registered:</b> (SSIC_ENH_CLK_RATIO==1 &amp;&amp; SSIC_SPI_BRIDGE==0) ? No : Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mst_txd[(SERIOW-1):0]	O	<p>Controller Interface Transmit data signal. Output data from the serial controller or serial target is transmitted on this line.</p> <p><b>Exists:</b> (SSIC_IS_MASTER==2)</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
slv_txd[(SIOW-1):0]	O	<p>Target Interface Transmit data signal. Output data from the serial controller or serial target is transmitted on this line.</p> <p><b>Exists:</b> (SSIC_IS_MASTER==2)</p> <p><b>Synchronous To:</b> SSIC_ENH_CLK_RATIO==1 ? "ssi_clk,sclk_in" : "ssi_clk"</p> <p><b>Registered:</b> SSIC_ENH_CLK_RATIO==1 ? No : Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
ssi_oe_n[(SERIOW-1):0]	O	<p>Output enable signal. Used to control external buffers on serial output lines.</p> <p><b>Exists:</b> (SSIC_IS_MASTER!=2)</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
mst_ssi_oe_n[(SERIOW-1):0]	O	<p>Controller mode Output enable signal. Used to control external buffers on serial output lines.</p> <p><b>Exists:</b> (SSIC_IS_MASTER==2)</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
slv_ssi_oe_n[(SIOW-1):0]	O	<p>Target mode Output enable signal. Used to control external buffers on serial output lines.</p> <p><b>Exists:</b> (SSIC_IS_MASTER==2)</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
txd_dm[(DM_W-1):0]	O	<p>Data mask signal for the transmit line (txd). When active data on txd line is masked, this signal can be used to partially update memory locations in the SPI device.</p> <p><b>Exists:</b> (SSIC_SPI_DM_EN==1)</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
txd_dm_oe_n[(DM_W-1):0]	O	<p>Output enable signal for data mask.</p> <p><b>Exists:</b> (SSIC_SPI_DM_EN==1)</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ss_x_n (for x = 0; x <= SSIC_NUM_SLAVES)	O	<p>Chip select output. For MWire and SPI, the active polarity is logic 0; for SSP, the active polarity is logic 1. The number of target select signals depends on the number of configured targets (SSIC_NUM_SLAVES).</p> <p><b>Exists:</b> SSIC_NUM_SLAVES-1 &gt;= x</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
dbg_spi_dtr_en	O	<p>Debug signal, represents if SPI transfer is transmitting/receiving data in DTR mode.</p> <p><b>Exists:</b> (SSIC_HAS_DDR==1)</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
sclk_out	O	<p>Serial bit-rate clock. Generated by the DWC_ssi from ssi_clk.</p> <p><b>Exists:</b> (SSIC_IS_MASTER!=0)</p> <p><b>Synchronous To:</b> ssi_clk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

## 5.2 AHB Subordinate Interface Signals

hclk	- hrdata
hresetn	- hruser
hsel	- hresp
haddr	- hready_resp
hauser	
hsize	
htrans	
hburst	
hready	
hwwrite	
hwdtata	
hwuser	
xip_en	
slave_sel	

**Table 5-2 AHB Subordinate Interface Signals**

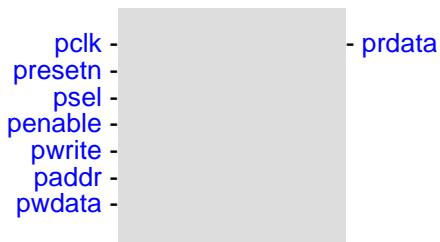
Port Name	I/O	Description
hclk	I	<p>AHB subordinate interface clock</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> None</p> <p><b>Registered:</b> N/A</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
hresetn	I	<p>AHB subordinate interface reset. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of hclk. DWC_ssi does not contain logic to perform this synchronization, so it must be provided externally.</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> N/A</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
hsel	I	<p>AHB select</p> <p>This signal is asserted when the current transfer on the AHB bus is intended to DWC_ssi.</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
haddr[31:0]	I	<p>Address</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
hauser[3:0]	I	<p>AHB Subordinate Bus Interface address parity. This address parity is driven by System manager that configures DWC_ssi.</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1) &amp;&amp; (SSIC_SLVIF_PAR_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
hsize[2:0]	I	<p>Transfer size</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
htrans[1:0]	I	<p>Transfer type</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
hburst[2:0]	I	<p>AHB Burst Type</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
hready	I	<p>Current transfer complete</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
hwwrite	I	<p>Transfer direction. When high, this signal indicates a write transfer. When low, this signal indicates a read transfer.</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> No  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> 1 for write, 0 for read</p>
hwdata[31:0]	I	<p>Write data to subordinate</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> No  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> N/A</p>
hwuser[3:0]	I	<p>AHB Subordinate Bus Interface write data parity. This write data parity is driven by the System manager that configures DWC_ssi.</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1) &amp;&amp; (SSIC_SLVIF_PAR_EN &gt;= 1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> No  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> N/A</p>
hrdata[31:0]	O	<p>Read data from subordinate</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> Yes  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> N/A</p>
hruser[3:0]	O	<p>AHB Subordinate Bus Interface read data parity. The DWC_ssi drives this read data parity along with the read data.</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1) &amp;&amp; (SSIC_SLVIF_PAR_EN &gt;= 1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> Yes  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> N/A</p>
hresp[1:0]	O	<p>Response type from AHB subordinate interface</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> Yes  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> N/A</p>

Port Name	I/O	Description
hready_resp	O	<p>Transfer complete</p> <p><b>Exists:</b> (SSIC_SLVIF_TYPE==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
xip_en	I	<p>AHB side-band signal that is used to distinguish between normal register read-write and XIP transfers. When xip_en is driven to 1, all transfers are considered to be XIP transfers.</p> <p><b>Exists:</b> (SSIC_XIP_EN==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
slave_sel[(LOG2_SSIC_NUM_SLAVES-1):0]	I	<p>External Chip select input when XIP is enabled.</p> <p><b>Exists:</b> (SSIC_XIP_EXT_SSEL_EN==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> SSIC_NUM_SLAVES&lt;=2 ? Yes : No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

## 5.3 APB Completer Interface Signals



**Table 5-3 APB Completer Interface Signals**

Port Name	I/O	Description
pclk	I	<p>APB clock.</p> <p><b>Exists:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Synchronous To:</b> None</p> <p><b>Registered:</b> N/A</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
presetn	I	<p>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.</p> <p><b>Exists:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> N/A</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
psel	I	<p>APB peripheral select that lasts for two pclk cycles. When asserted, indicates that the peripheral has been selected for read or write operation.</p> <p><b>Exists:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Synchronous To:</b> pclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
penable	I	<p>APB enable control. Asserted for a single pclk cycle and used for timing read or write operations.</p> <p><b>Exists:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Synchronous To:</b> pclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
pwrite	I	<p>APB write control.</p> <p><b>Exists:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Synchronous To:</b> pclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> When high, indicates a write access to the peripheral; when low, indicates a read access.</p>
paddr[31:0]	I	<p>APB address bus. Uses lower 8 bits of the address bus for register decode.</p> <p><b>Exists:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Synchronous To:</b> pclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
pwdata[(APB_DATA_WIDTH-1):0]	I	<p>APB write data bus. Driven by the APB requester (bridge unit) during write cycles. Can be 8, 16, or 32 bits wide.</p> <p><b>Exists:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Synchronous To:</b> pclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
prdata[(APB_DATA_WIDTH-1):0]	O	<p>APB readback data. Driven by the selected peripheral during read cycles. Can be 8, 16, or 32 bits wide</p> <p><b>Exists:</b> SSIC_SLVIF_TYPE==0</p> <p><b>Synchronous To:</b> pclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

## 5.4 DMA Interface Signals

dma\_tx\_ack -  
dma\_rx\_ack -  
dma\_tx\_req  
dma\_rx\_req  
dma\_tx\_single  
dma\_rx\_single

**Table 5-4 DMA Interface Signals**

Port Name	I/O	Description
dma_tx_ack	I	<p>DMA Transmit Acknowledgement sent by the DMA Controller to acknowledge the end of each DMA burst or single transaction to the transmit FIFO.</p> <p><b>Exists:</b> (SSIC_HAS_DMA==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
dma_rx_ack	I	<p>DMA Receive Acknowledgement sent by the DMA controller to acknowledge the end of each DMA burst or single transaction from the receive FIFO.</p> <p><b>Exists:</b> (SSIC_HAS_DMA==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
dma_tx_req	O	<p>Transmit FIFO DMA Request. This signal is asserted when the transmit FIFO requires service from the DMA Controller, that is, the transmit FIFO is at or below the watermark level.</p> <ul style="list-style-type: none"> <li>■ 0: Not requesting</li> <li>■ 1: Requesting</li> </ul> <p>The 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.</p> <p><b>Exists:</b> (SSIC_HAS_DMA==1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
dma_rx_req	O	<p>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.</p> <ul style="list-style-type: none"> <li>■ 0: Not requesting</li> <li>■ 1: Requesting</li> </ul> <p>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.</p> <p><b>Exists:</b> (SSIC_HAS_DMA==1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> Yes  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> High</p>
dma_tx_single	O	<p>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.</p> <ul style="list-style-type: none"> <li>■ 0: Transmit FIFO is full</li> <li>■ 1: Transmit FIFO is not full</li> </ul> <p><b>Exists:</b> (SSIC_HAS_DMA==1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> Yes  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> High</p>
dma_rx_single	O	<p>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.</p> <ul style="list-style-type: none"> <li>■ 0: Receive FIFO is empty</li> <li>■ 1: Receive FIFO is not empty</li> </ul> <p><b>Exists:</b> (SSIC_HAS_DMA==1)  <b>Synchronous To:</b> hclk  <b>Registered:</b> Yes  <b>Power Domain:</b> SINGLE_DOMAIN  <b>Active State:</b> High</p>

## 5.5 AHB Manager Interface Signals



**Table 5-5 AHB Manager Interface Signals**

Port Name	I/O	Description
mhclk	I	AHB Manager interface clock <b>Exists:</b> (SSIC_SPI_BRIDGE==1) <b>Synchronous To:</b> None <b>Registered:</b> N/A <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A.
mhresetn	I	AHB Manager interface reset <b>Exists:</b> (SSIC_SPI_BRIDGE==1) <b>Synchronous To:</b> Asynchronous <b>Registered:</b> N/A <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low
mhgrant	I	Grant Signal from AHB Interconnect <b>Exists:</b> (SSIC_SPI_BRIDGE==1) <b>Synchronous To:</b> mhclk <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> High
mhresp[1:0]	I	Response type from AHB subordinate <b>Exists:</b> (SSIC_SPI_BRIDGE==1) <b>Synchronous To:</b> mhclk <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A

Port Name	I/O	Description
mhrdata[31:0]	I	<p>Read data from AHB subordinate</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mhready	I	<p>Ready signal from AHB interconnect</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
mhbusreq	O	<p>Bus Request</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
mhaddr[31:0]	O	<p>AHB Manager port address</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mhtrans[1:0]	O	<p>Transfer control. Indicates type of current transfer.</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mhwrite	O	<p>Manager port transfer direction. When active high, this signal indicates a write transfer. When active low, this signal indicates a read transfer.</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> 1 for Write; 0 for Read</p>

Port Name	I/O	Description
mhwdata[31:0]	O	<p>Manager port write data</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mhsize[2:0]	O	<p>Manager port transfer size</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> None</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mhburst[2:0]	O	<p>Burst type and length control</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
mhlock	O	<p>AHB bus lock qualifier.</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> mhclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
mhprot[3:0]	O	<p>Protection control</p> <p><b>Exists:</b> (SSIC_SPI_BRIDGE==1)</p> <p><b>Synchronous To:</b> None</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

## 5.6 DFT Interface Signals

scan\_mode -

**Table 5-6 DFT Interface Signals**

Port Name	I/O	Description
scan_mode	I	<p>Scan mode. This signal should be asserted, that is, driven to logic 1 during scan testing and should be deasserted, that is tied to logic 0, at all other times.</p> <p><b>Exists:</b> (SSIC_ENH_CLK_RATIO==1    SSIC_HAS_RXDS==1    SSIC_HAS_RX_SAMPLE_DELAY ==2    SSIC_HAS_CLK_LOOP_BACK == 1)    (SSIC_HAS_DMA ==2 &amp;&amp; (SSIC_AXIM_SYNC_CLK == 0    SSIC_AXIM_SLVIF_SYNC_CLK == 0    SSIC_SYNC_CLK == 0))</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

## 5.7 Debug Interface Signals

- spi\_mode  
 - ssi\_is\_mst  
 - ssi\_busy

**Table 5-7 Debug Interface Signals**

Port Name	I/O	Description
spi_mode[(SPI_W-1):0]	O	<p>SPI Frame format indicator signal. Indicates the mode in which the SPI transfer is happening.</p> <ul style="list-style-type: none"> <li>■ 00: Standard SPI mode</li> <li>■ 01: SPI Dual Mode</li> <li>■ 10: SPI Quad mode</li> <li>■ 11: SPI Octal mode</li> </ul> <p>This signal can be used for multiplexing the I/O lines for different mode of operations.</p> <p><b>Exists:</b> (SSIC_SPI_MODE!=0    SSIC_SLV_SPI_MODE!=0) &amp;&amp; SSIC_SPI_BRIDGE == 0</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> SSIC_SPI_MODE &lt;= 3 ? "Yes" : "No"</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_is_mst	O	<p>Debug signal which shows if DWC_ssi is programmed to work as controller or target.</p> <ul style="list-style-type: none"> <li>■ 0: DWC_ssi in Target mode</li> <li>■ 1: DWC_ssi in Controller mode</li> </ul> <p><b>Exists:</b> (SSIC_IS_MASTER==2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_busy	O	<p>Debug signal which shows if DWC_ssi is busy</p> <ul style="list-style-type: none"> <li>■ 0: DWC_ssi is not busy</li> <li>■ 1: DWC_ssi is busy</li> </ul> <p><b>Exists:</b> Always</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

## 5.8 Single RAM Interface Signals

ram\_p1\_data\_out  
 ram\_p2\_data\_out  
 - ram\_p1\_clk  
 - ram\_p1\_resetn  
 - ram\_p1\_cs\_n  
 - ram\_p1\_wr\_n  
 - ram\_p1\_addr  
 - ram\_p1\_data\_in  
 - ram\_p2\_clk  
 - ram\_p2\_resetn  
 - ram\_p2\_cs\_n  
 - ram\_p2\_wr\_n  
 - ram\_p2\_addr  
 - ram\_p2\_data\_in

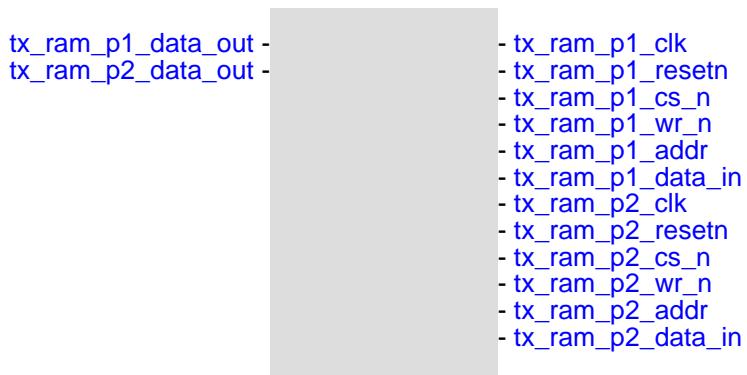
**Table 5-8 Single RAM Interface Signals**

Port Name	I/O	Description
ram_p1_clk	O	Port 1 RAM clock <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 0) <b>Synchronous To:</b> hclk <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
ram_p1_resetn	O	Port 1 RAM Asynchronous Reset <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 0) <b>Synchronous To:</b> Asynchronous <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low
ram_p1_cs_n	O	Chip Select for Port 1 of External RAM. <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 0) <b>Synchronous To:</b> hclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low
ram_p1_wr_n	O	Read or Write Control for Port 1 of External RAM. <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 0) <b>Synchronous To:</b> hclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low

Port Name	I/O	Description
ram_p1_addr[(SSIC_RAM_AW-1):0]	O	<p>Address for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
ram_p1_data_in[31:0]	O	<p>Data Input for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
ram_p1_data_out[31:0]	I	<p>Data Output for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
ram_p2_clk	O	<p>Port 2 RAM clock</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
ram_p2_resetn	O	<p>Port 2 RAM Asynchronous Reset</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ram_p2_cs_n	O	<p>Chip Select for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
ram_p2_wr_n	O	<p>Read or Write Control for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ram_p2_addr[(SSIC_RAM_AW-1):0]	O	<p>Address for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
ram_p2_data_in[31:0]	O	<p>Data Input for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
ram_p2_data_out[31:0]	I	<p>Data Output for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 0)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

## 5.9 Transmit RAM Interface Signals



**Table 5-9      Transmit RAM Interface Signals**

Port Name	I/O	Description
tx_ram_p1_clk	O	Clock for Port 1 of Transmit RAM interface <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 1) <b>Synchronous To:</b> hclk <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
tx_ram_p1_resetn	O	Asynchronous reset for Port 1 of Transmit RAM interface <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 1) <b>Synchronous To:</b> Asynchronous <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low
tx_ram_p1_cs_n	O	Transmit FIFO Chip Select for Port 1 of External RAM. <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 1) <b>Synchronous To:</b> hclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low
tx_ram_p1_wr_n	O	Transmit FIFO Read or Write Control for Port 1 of External RAM. <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 1) <b>Synchronous To:</b> hclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low

Port Name	I/O	Description
tx_ram_p1_addr[(TX_ABW-1):0]	O	<p>Transmit FIFO Address for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
tx_ram_p1_data_in[31:0]	O	<p>Transmit FIFO Data Input for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
tx_ram_p1_data_out[31:0]	I	<p>Transmit FIFO Data Output for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
tx_ram_p2_clk	O	<p>Clock for Port 2 of Transmit RAM interface</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
tx_ram_p2_resetn	O	<p>Asynchronous reset for Port 2 of Transmit RAM interface</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
tx_ram_p2_cs_n	O	<p>Transmit FIFO Chip Select for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
tx_ram_p2_wr_n	O	<p>Transmit FIFO Read or Write Control for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
tx_ram_p2_addr[(TX_ABW-1):0]	O	<p>Transmit FIFO Address for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
tx_ram_p2_data_in[31:0]	O	<p>Transmit FIFO Data Input for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
tx_ram_p2_data_out[31:0]	I	<p>Transmit FIFO Data Output for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

## 5.10 Receive RAM Interface Signals

<code>rx_ram_p1_data_out</code> - <code>rx_ram_p2_data_out</code> -	<ul style="list-style-type: none"> <li>- <code>rx_ram_p1_clk</code></li> <li>- <code>rx_ram_p1_resetn</code></li> <li>- <code>rx_ram_p1_cs_n</code></li> <li>- <code>rx_ram_p1_wr_n</code></li> <li>- <code>rx_ram_p1_addr</code></li> <li>- <code>rx_ram_p1_data_in</code></li> <li>- <code>rx_ram_p2_clk</code></li> <li>- <code>rx_ram_p2_resetn</code></li> <li>- <code>rx_ram_p2_cs_n</code></li> <li>- <code>rx_ram_p2_wr_n</code></li> <li>- <code>rx_ram_p2_addr</code></li> <li>- <code>rx_ram_p2_data_in</code></li> </ul>
------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Table 5-10    Receive RAM Interface Signals**

Port Name	I/O	Description
<code>rx_ram_p1_clk</code>	O	Clock for Port 1 of Receive RAM interface <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 1) <b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk" <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
<code>rx_ram_p1_resetn</code>	O	Asynchronous reset for Port 1 of Receive RAM interface <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 1) <b>Synchronous To:</b> Asynchronous <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low
<code>rx_ram_p1_cs_n</code>	O	Receive FIFO Chip Select for Port 1 of External RAM. <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 1) <b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk" <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low
<code>rx_ram_p1_wr_n</code>	O	Receive FIFO Read or Write Control for Port 1 of External RAM. <b>Exists:</b> (SSIC_HAS_EXT_RAM==1) && (SSIC_HAS_TX_RX_EN == 1) <b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk" <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low

Port Name	I/O	Description
rx_ram_p1_addr[(RX_ABW-1):0]	O	<p>Receive FIFO Address for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rx_ram_p1_data_in[31:0]	O	<p>Receive FIFO Data Input for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rx_ram_p1_data_out[31:0]	I	<p>Receive FIFO Data Output for Port 1 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SYNC_CLK==1 ? "hclk" : "ssi_clk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rx_ram_p2_clk	O	<p>Clock for Port 2 of Receive RAM interface</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rx_ram_p2_resetn	O	<p>Asynchronous reset for Port 2 of Receive RAM interface</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> Asynchronous</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
rx_ram_p2_cs_n	O	<p>Receive FIFO Chip Select for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
rx_ram_p2_wr_n	O	<p>Receive FIFO Read or Write Control for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
rx_ram_p2_addr[(RX_ABW-1):0]	O	<p>Receive FIFO Address for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rx_ram_p2_data_in[31:0]	O	<p>Receive FIFO Data Input for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rx_ram_p2_data_out[31:0]	I	<p>Receive FIFO Data Output for Port 2 of External RAM.</p> <p><b>Exists:</b> (SSIC_HAS_EXT_RAM==1) &amp;&amp; (SSIC_HAS_TX_RX_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

## 5.11 AXI Manager Interface Signals

ackl	- awvalid
aresetn	- awvalid_parity
awready	- awid
awready_parity	- awaddr
wready	- awlen
wready_parity	- awsize
bvalid	- awburst
bvalid_parity	- awlock
bid	- awprot
bresp	- awcache
buser	- awuser
arready	- wvalid
arready_parity	- wvalid_parity
rvalid	- wdata
rvalid_parity	- wstrb
rdata	- wlast
rresp	- wuser
rlast	- wid
rid	- bready
ruser	- bready_parity
	- arvalid
	- arvalid_parity
	- araddr
	- arlen
	- arsize
	- arbust
	- arlock
	- arprot
	- arcache
	- arid
	- aruser
	- rready
	- rready_parity

Table 5-11 AXI Manager Interface Signals

Port Name	I/O	Description
ackl	I	AXI interface clock <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> None <b>Registered:</b> N/A <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
aresetn	I	AXI interface reset. <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> Asynchronous <b>Registered:</b> N/A <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> Low

Port Name	I/O	Description
awvalid	O	AXI Manager Interface Write Address Valid <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awvalid_parity	O	AXI Manager Interface Write Address Valid Parity <b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1) <b>Synchronous To:</b> aclk <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awready	I	AXI Manager Interface Write Address Ready <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awready_parity	I	AXI Manager Interface Write Address Ready Parity <b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1) <b>Synchronous To:</b> aclk <b>Registered:</b> No <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awid[(SSIC_AXI_IDW-1):0]	O	AXI Manager Interface Write Address ID <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awaddr[(SSIC_AXI_AW-1):0]	O	AXI Manager Interface Write Address <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A

Port Name	I/O	Description
awlen[(SSIC_AXI_BLW-1):0]	O	AXI Manager Interface Write Burst Length <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awsizer[2:0]	O	AXI Manager Interface Write Burst Size <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awburst[1:0]	O	AXI Manager Interface Write Burst Type <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awlock[(SSIC_AXI_LTW-1):0]	O	AXI Manager Interface Write Lock Type <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> SSIC_AW_REGSLICE_EN==0 ? "No" : "Yes" <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awprot[2:0]	O	AXI Manager Interface Write Protection Type <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
awcache[3:0]	O	AXI Manager Interface Write Cache Type <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A

Port Name	I/O	Description
awuser[(SSIC_AXI_AWUW-1):0]	O	<p>AXI Manager write address user signal</p> <p><b>Exists:</b> (SSIC_AXI_SAFETY_EN != 0) &amp;&amp; (SSIC_HAS_DMA == 2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
wvalid	O	<p>AXI Manager Interface Write Data Valid</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
wvalid_parity	O	<p>AXI Manager Interface Write Data Valid Parity</p> <p><b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
wready	I	<p>AXI Manager Interface Write Data Ready</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
wready_parity	I	<p>AXI Manager Interface Write Data Ready Parity</p> <p><b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
wdata[(SSIC_AXI_DW-1):0]	O	<p>AXI Manager Interface Write Data</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

Port Name	I/O	Description
wstrb[(SSIC_AXI_WSW-1):0]	O	<p>AXI Manager Interface Write Data Strobe</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
wlast	O	<p>AXI Manager Interface Write Last.</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
wuser[(SSIC_AXI_WUW-1):0]	O	<p>AXI Manager Interface write data user signal</p> <p><b>Exists:</b> (SSIC_AXI_SAFETY_EN != 0) &amp;&amp; (SSIC_HAS_DMA == 2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
wid[(SSIC_AXI_IDW-1):0]	O	<p>AXI3 Manager Interface Write Data ID</p> <p><b>Exists:</b> (SSIC_AXI_INTF_TYPE == 0 &amp;&amp; SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
bvalid	I	<p>AXI Manager Interface Write Response Valid</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> (SSIC_SAFETY_PKG_EN!=0) ? "No" : "Yes"</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
bvalid_parity	I	<p>AXI Manager Interface Write Response Valid Parity</p> <p><b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

Port Name	I/O	Description
bready	O	<p>AXI Manager Interface Write Response Ready</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> (SSIC_SAFETY_PKG_EN!=0) ? "No" : "Yes"</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
bready_parity	O	<p>AXI Manager Interface Write Response Ready Parity</p> <p><b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
bid[(SSIC_AXI_IDW-1):0]	I	<p>AXI Manager Interface Write Response ID</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
bresp[1:0]	I	<p>AXI Manager Interface Write Response</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> (SSIC_SAFETY_PKG_EN!=0) ? "No" : "Yes"</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
buser[(SSIC_AXI_BUW-1):0]	I	<p>AXI Manager Interface write response user signal</p> <p><b>Exists:</b> (SSIC_AXI_SAFETY_EN != 0)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
arvalid	O	<p>AXI Manager Interface Read Address Valid</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

Port Name	I/O	Description
arvalid_parity	O	<p>AXI Manager Interface Read Address Valid Parity</p> <p><b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
arready	I	<p>AXI Manager Interface Read Address Ready</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
arready_parity	I	<p>AXI Manager Interface Read Address Ready Parity</p> <p><b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
araddr[(SSIC_AXI_AW-1):0]	O	<p>AXI Manager Interface Read Address</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
arlen[(SSIC_AXI_BLW-1):0]	O	<p>AXI Manager Interface Read Burst Length</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
arsize[2:0]	O	<p>AXI Manager Interface Read Burst Size</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

Port Name	I/O	Description
arburst[1:0]	O	AXI Manager Interface Read Burst Type <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
arlock[(SSIC_AXI_LTW-1):0]	O	AXI Manager Interface Read Lock Type <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> SSIC_AR_REGSLICE_EN==0 ? "No" : "Yes" <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
arprot[2:0]	O	AXI Manager Interface Read Protection Type <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
arcache[3:0]	O	AXI Manager Interface Read Cache Type <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
arid[(SSIC_AXI_IDW-1):0]	O	AXI Manager Interface Read Address ID <b>Exists:</b> (SSIC_HAS_DMA==2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A
aruser[(SSIC_AXI_ARUW-1):0]	O	AXI Manager Interface Read address user signal <b>Exists:</b> (SSIC_AXI_SAFETY_EN != 0) && (SSIC_HAS_DMA == 2) <b>Synchronous To:</b> aclk <b>Registered:</b> Yes <b>Power Domain:</b> SINGLE_DOMAIN <b>Active State:</b> N/A

Port Name	I/O	Description
rvalid	I	<p>AXI Manager Interface Read Data Valid</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rvalid_parity	I	<p>AXI Manager Interface Read Data Valid Parity</p> <p><b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rready	O	<p>AXI Manager Interface Read Data Ready</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> (SSIC_SAFETY_PKG_EN!=0) ? "No" : "Yes"</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rready_parity	O	<p>AXI Manager Interface Read Data Ready Parity</p> <p><b>Exists:</b> (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rdata[(SSIC_AXI_DW-1):0]	I	<p>AXI Manager Interface Read Data</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rresp[1:0]	I	<p>AXI Manager Interface Read Response</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

Port Name	I/O	Description
rlast	I	<p>AXI Manager Interface Read Last</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
rid[({SSIC_AXI_IDW}-1):0]	I	<p>AXI Manager Interface Read Data ID</p> <p><b>Exists:</b> (SSIC_HAS_DMA==2)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>
ruser[({SSIC_AXI_RUW}-1):0]	I	<p>AXI Manager Interface read data user signal</p> <p><b>Exists:</b> (SSIC_AXI_SAFETY_EN != 0)</p> <p><b>Synchronous To:</b> aclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> N/A</p>

## 5.12 Interrupt Signals

```
- ssi_intr
- ssi_txe_intr
- ssi_spite_intr
- ssi_txo_intr
- ssi_rxf_intr
- ssi_rxo_intr
- ssi_txu_intr
- ssi_ahbe_intr
- ssi_spime_intr
- ssi_rxu_intr
- ssi_mst_intr
- ssi_xrxo_intr
- ssi_sfyt_addr_sp_intr
- ssi_sfyt_data_sp_intr
- ssi_intr_n
- ssi_txe_intr_n
- ssi_spite_intr_n
- ssi_txo_intr_n
- ssi_rxf_intr_n
- ssi_rxo_intr_n
- ssi_txu_intr_n
- ssi_ahbe_intr_n
- ssi_spime_intr_n
- ssi_xrxo_intr_n
- ssi_sfyt_addr_sp_intr_n
- ssi_sfyt_data_sp_intr_n
- ssi_rxu_intr_n
- ssi_done_intr
- ssi_done_intr_n
- ssi_axie_intr
- ssi_axie_intr_n
- ssi_sfyt_ue_intr
- ssi_sfyt_ue_intr_n
- ssi_sfyt_re_intr
- ssi_sfyt_re_intr_n
- ssi_sfyt_fp_intr
- ssi_sfyt_fp_intr_n
- ssi_sfyt_fto_intr
- ssi_sfyt_fto_intr_n
- ssi_sfyt_ap_intr
- ssi_sfyt_ap_intr_n
- ssi_sfyt_axi_vrp_intr
- ssi_sfyt_axi_vrp_intr_n
- ssi_sfyt_axi_ecc_ue_intr
- ssi_sfyt_axi_ecc_ue_intr_n
- ssi_sfyt_axi_ecc_ce_intr
- ssi_sfyt_axi_ecc_ce_intr_n
- ssi_sfyt_ld_intr
- ssi_sfyt_ld_intr_n
- ssi_sfyt_tmr_intr
- ssi_sfyt_tmr_intr_n
- ssi_sfyt_dpe_intr
- ssi_sfyt_dpe_intr_n
- ssi_sfyt_dp_ecc_ce_intr
- ssi_sfyt_dp_ecc_ce_intr_n
- ssi_sfyt_dp_ecc_ue_intr
- ssi_sfyt_dp_ecc_ue_intr_n
- ssi_sfyt_rs_intr
- ssi_sfyt_rs_intr_n
```


 - ssi\_mst\_intr\_n
**Table 5-12** Interrupt Signals

Port Name	I/O	Description
ssi_intr	O	<p>Optional. Combined SSI active high interrupt flag. Logical OR of all individual interrupts.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_COMBINED) &amp;&amp; (SSIC_INTR_POL==1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_txe_intr	O	<p>Optional. Transmit FIFO empty active high interrupt. Active when the transmit FIFO is equal to or below the threshold value (TFT).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_spite_intr	O	<p>Optional. SPI Transmit Error active high interrupt.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SPI_DYN_WS_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_txo_intr	O	<p>Optional. Transmit FIFO overflow active high interrupt. Active when the AHB attempts to write to a full transmit FIFO.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SPI_BRIDGE == 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_rxf_intr	O	<p>Optional. Receive FIFO full active high interrupt. Active when the receive FIFO is equal to or above the threshold value (RFT) plus 1.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
ssi_rxo_intr	O	<p>Optional. Receive FIFO overflow active high interrupt. Active when the receive logic attempts to write into a full receive FIFO.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_txu_intr	O	<p>Optional. Transmit FIFO underflow active high interrupt. Active when the transmit logic attempts to read from empty transmit FIFO.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SPI_BRIDGE == 1    SSIC_HAS_DMA == 2    SSIC_XIP_WRITE_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_ahbe_intr	O	<p>Optional. AHB Error active high interrupt. Active when any error response if received on AHB interface.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SPI_BRIDGE == 1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_spime_intr	O	<p>Optional. SPI Controller error active high interrupt. Active when any error occurs from SPI Controller interface.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SPI_BRIDGE == 1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_rxu_intr	O	<p>Optional. Receive FIFO underflow active high interrupt. Active when the AHB attempts to read from an empty receive FIFO.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SPI_BRIDGE == 0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
ssi_mst_intr	O	<p>Optional. Multi-controller contention active high interrupt. Informs of possible bus contention.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_IS_MASTER!=0) &amp;&amp; (SSIC_INTR_POL==1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_xrxo_intr	O	<p>Optional. XIP Receive FIFO overflow active high interrupt. Active when the XIP receive FIFO overflows, this may happen due to prolonged WAIT states inserted by Controller during the transfer or by choosing lower FIFO depth required for the given frequency ratio between hclk and ssi_clk.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_CONCURRENT_XIP_EN==1) &amp;&amp; (SSIC_INTR_POL==1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_addr_sp_intr	O	<p>SSI AHB subordinate interface parity interrupt. This signal will be asserted when DWC_ssi encounters an address parity error on AHB subordinate interface. Active High.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_SLVIF_TYPE==1) &amp;&amp; (SSIC_SLVIF_PAR_EN == 2) &amp;&amp; (SSIC_INTR_POL==1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_data_sp_intr	O	<p>SSI AHB subordinate interface parity interrupt. This signal will be asserted when DWC_ssi encounters a data parity error on AHB subordinate interface. Active High.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_SLVIF_TYPE==1) &amp;&amp; (SSIC_SLVIF_PAR_EN &gt;= 1) &amp;&amp; (SSIC_INTR_POL==1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
ssi_intr_n	O	<p>Optional. Combined SSI interrupt flag. Logical OR of all individual interrupts.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_COMBINED) &amp;&amp; (SSIC_INTR_POL==0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_txe_intr_n	O	<p>Optional. Transmit FIFO empty interrupt. Active when the transmit FIFO is equal to or below the threshold value (TFT).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_spite_intr_n	O	<p>Optional. SPI Transmit Error interrupt.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SPI_DYN_WS_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_txo_intr_n	O	<p>Optional. Transmit FIFO overflow interrupt. Active when the AHB attempts to write to a full transmit FIFO.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SPI_BRIDGE == 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_rxf_intr_n	O	<p>Optional. Receive FIFO full interrupt. Active when the receive FIFO is equal to or above the threshold value (RFT) plus 1.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
ssi_rxo_intr_n	O	<p>Optional. Receive FIFO overflow interrupt. Active when the receive logic attempts to write into a full receive FIFO.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_txu_intr_n	O	<p>Optional. Transmit FIFO underflow interrupt. Active when the transmit logic attempts to read from empty transmit FIFO.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SPI_BRIDGE == 1    SSIC_HAS_DMA == 2    SSIC_XIP_WRITE_EN == 1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_ahbe_intr_n	O	<p>Optional. AHB Error interrupt. Active when any error response if received on AHB interface.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SPI_BRIDGE == 1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_spime_intr_n	O	<p>Optional. SPI Controller error interrupt. Active when any error occurs from SPI Controller interface.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SPI_BRIDGE == 1)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_xrxo_intr_n	O	<p>Optional. XIP Receive FIFO overflow interrupt. Active when the XIP receive FIFO overflows, this may happen due to prolonged WAIT states inserted by Controller during the transfer or by choosing lower FIFO depth required for the given frequency ratio between hclk and ssi_clk.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_CONCURRENT_XIP_EN==1) &amp;&amp; (SSIC_INTR_POL==0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
ssi_sfty_addr_sp_intr_n	O	<p>SSI AHB subordinate interface parity interrupt. This signal will be asserted when DWC_ssi encounters an address parity error on AHB subordinate interface.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_SLVIF_TYPE==1) &amp;&amp; (SSIC_SLVIF_PAR_EN == 2) &amp;&amp; (SSIC_INTR_POL==0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_data_sp_intr_n	O	<p>SSI AHB subordinate interface parity interrupt. This signal will be asserted when DWC_ssi encounters a data parity error on AHB subordinate interface.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_SLVIF_TYPE==1) &amp;&amp; (SSIC_SLVIF_PAR_EN &gt;= 1) &amp;&amp; (SSIC_INTR_POL==0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_rxu_intr_n	O	<p>Optional. Receive FIFO underflow interrupt. Active when the AHB attempts to read from an empty receive FIFO.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SPI_BRIDGE == 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_done_intr	O	<p>Optional. Transfer done interrupt (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_HAS_DMA == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_done_intr_n	O	<p>Optional. Transfer done interrupt</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_HAS_DMA == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
ssi_axie_intr	O	<p>Optional. AXI Error interrupt (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_HAS_DMA == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_axie_intr_n	O	<p>Optional. AXI Error interrupt</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_HAS_DMA == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_ue_intr	O	<p>Optional. Combined Active High SSI Unrecoverable safety interrupt flag. Logical OR of all individual unrecoverable safety interrupts.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_COMBINED) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN != 0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_ue_intr_n	O	<p>Optional. Combined Active Low SSI Unrecoverable safety interrupt flag. Logical OR of all individual unrecoverable safety interrupts.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_COMBINED) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN != 0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_re_intr	O	<p>Optional. Combined Active High SSI Recoverable safety interrupt flag. Logical OR of all individual recoverable safety interrupts.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_COMBINED) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN != 0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
ssi_sfty_re_intr_n	O	<p>Optional. Combined Active Low SSI Recoverable safety interrupt flag. Logical OR of all individual recoverable safety interrupts.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_COMBINED) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN != 0)</p> <p><b>Synchronous To:</b> SSIC_SLVIF_TYPE==0 ? "pclk" : "hclk"</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_fp_intr	O	<p>Optional. FSM one-hot encoding error interrupt (active HIGH)</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_fp_intr_n	O	<p>Optional. FSM one-hot encoding error interrupt</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_fto_intr	O	<p>Optional. SSI AXI FSM time out interrupt (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN != 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_fto_intr_n	O	<p>Optional. FSM time out interrupt (active LOW).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN != 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
ssi_sfty_ap_intr	O	<p>Optional. SSI AXI interface parity interrupt (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_AXI_SAFETY_EN != 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_ap_intr_n	O	<p>Optional. SSI AXI interface parity interrupt (active LOW)</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_AXI_SAFETY_EN != 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_axi_vrp_intr	O	<p>Optional. SSI AXI interface Valid/Ready parity interrupt (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_axi_vrp_intr_n	O	<p>Optional. SSI AXI interface Valid/Ready parity interrupt (active LOW).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_READY_VALID_PAR_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_axi_ecc_ue_intr	O	<p>Optional. AXI Manager Interface ECC uncorrectable error (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_AXI_SAFETY_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
ssi_sfty_axi_ecc_ue_intr_n	O	<p>Optional. AXI Manager Interface ECC uncorrectable error (active LOW).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_AXI_SAFETY_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_axi_ecc_ce_intr	O	<p>Optional. AXI Manager Interface ECC correctable error (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_AXI_SAFETY_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_axi_ecc_ce_intr_n	O	<p>Optional. AXI Manager Interface ECC correctable error (active LOW)</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_AXI_SAFETY_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_id_intr	O	<p>Optional. Logic Duplication Safety Error. This signal will be asserted when DWC_ssi encounters error in logic duplication safety mechanism. (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_id_intr_n	O	<p>Optional. Logic Duplication Safety Error. This signal will be asserted when DWC_ssi encounters error in logic duplication safety mechanism. (active LOW)</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
ssi_sfty_tmr_intr	O	<p>Optional. TMR Safety Error. This signal will be asserted when DWC_ssi encounters error in TMR safety mechanism in FIFO controllers. (active HIGH).</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_tmr_intr_n	O	<p>Optional. TMR Safety Error. This signal will be asserted when DWC_ssi encounters error in TMR safety mechanism in FIFO controllers. (active LOW)</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_dpe_intr	O	<p>Optional. SSI Data path parity interrupt. This signal will be asserted when DWC_ssi encounters a parity error on internal data path. Active High</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_dpe_intr_n	O	<p>Optional. SSI Data path parity interrupt. This signal will be asserted when DWC_ssi encounters a parity error on internal data path</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN == 1)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>

Port Name	I/O	Description
ssi_sfty_dp_ecc_ce_intr	O	<p>Optional. SSI Data path ECC Correctable interrupt. This signal will be asserted when DWC_ssi encounters a parity error on internal data path. Active High</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_dp_ecc_ce_intr_n	O	<p>Optional. SSI Data path ECC Correctable interrupt. This signal will be asserted when DWC_ssi encounters a parity error on internal data path</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_dp_ecc_ue_intr	O	<p>Optional. SSI Data path ECC Uncorrectable interrupt. This signal will be asserted when DWC_ssi encounters a parity error on internal data path. Active High</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>
ssi_sfty_dp_ecc_ue_intr_n	O	<p>Optional. SSI Data path ECC Uncorrectable interrupt. This signal will be asserted when DWC_ssi encounters a parity error on internal data path</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN == 2)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_sfty_rs_intr	O	<p>Optional. SSI Data Register space interrupt. This signal will be asserted when DWC_ssi encounters an inconsistency in register space programming. Active High</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==1) &amp;&amp; (SSIC_SAFETY_PKG_EN != 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> Yes</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> High</p>

Port Name	I/O	Description
ssi_sfty_rs_intr_n	O	<p>Optional. SSI Data Register space interrupt. This signal will be asserted when DWC_ssi encounters an inconsistency in register space programming.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_INTR_POL==0) &amp;&amp; (SSIC_SAFETY_PKG_EN != 0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>
ssi_mst_intr_n	O	<p>Optional. Multi-Controller contention interrupt. Informs of possible bus contention.</p> <p><b>Exists:</b> (SSIC_INTR_IO==SSIC_INDIVIDUAL) &amp;&amp; (SSIC_IS_MASTER!=0) &amp;&amp; (SSIC_INTR_POL==0)</p> <p><b>Synchronous To:</b> hclk</p> <p><b>Registered:</b> No</p> <p><b>Power Domain:</b> SINGLE_DOMAIN</p> <p><b>Active State:</b> Low</p>



# Register Descriptions

---

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

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

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

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

## Exists Expressions

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

## Offset

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

## Memory Access Attributes

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

**Table 6-1 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 in a manner not described by any of the above.
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	A write modifies this register field in a manner not described by any of the above.
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 6-2 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 6-3** 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 controller 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.

Register definitions for each component memory map.

**Table 6-4** Registers for the DWC\_ssi\_memory\_map Memory Map

Register	Offset	Description
DWC_ssi register address block - 1 DWC_ssi_address_block Exists: SSIC_SPI_BRIDGE == 0		
"CTRLR0" on page 306	0x0	Control Register 0
"CTRLR1" on page 314	0x4	Control Register 1
"SSIENR" on page 315	0x8	SSI Enable Register
"MWCR" on page 316	0xc	Microwire Control Register
"SER" on page 318	0x10	Target Enable Register
"BAUDR" on page 319	0x14	Baud Rate Select
"TXFTLR" on page 321	0x18	Transmit FIFO Threshold Level
"RXFTLR" on page 323	0x1c	Receive FIFO Threshold Level

Register	Offset	Description
"TXFLR" on page 324	0x20	Transmit FIFO Level Register
"RXFLR" on page 325	0x24	Receive FIFO Level Register
"SR" on page 326	0x28	Status Register
"IMR" on page 329	0x2c	Interrupt Mask Register
"ISR" on page 332	0x30	Interrupt Status Register
"RISR" on page 335	0x34	Raw Interrupt Status Register
"TXEICR" on page 339	0x38	Transmit FIFO Error Interrupt Clear Registers
"RXOICR" on page 340	0x3c	Receive FIFO Overflow Interrupt Clear Register
"RXUICR" on page 341	0x40	Receive FIFO Underflow Interrupt Clear Register
"MSTICR" on page 342	0x44	Multi-Controller Interrupt Clear Register
"ICR" on page 343	0x48	Interrupt Clear Register
"DMACR" on page 344	0x4c	DMA Control Register
"DMATDLR" on page 347	0x50	DMA Transmit Data Level
"AXIAWLEN" on page 348	0x50	Destination Burst Length
"DMARDLR" on page 349	0x54	DMA Receive Data Level
"AXIARLEN" on page 350	0x54	Source Burst Length
"IDR" on page 351	0x58	Identification Register
"SSIC_VERSION_ID" on page 352	0x5c	DWC_ssi component version
"DR <sub>x</sub> (for i = 0; i <= 35)" on page 353	0x60 +i*0x4	DWC_ssi Data Register
"RX_SAMPLE_DELAY" on page 354	0xf0	RX Sample Delay Register
"SPI_CTRLR0" on page 356	0xf4	SPI_CTRLR0 - SPI Control Register
"DDR_DRIVE_EDGE" on page 362	0xf8	DDR_DRIVE_EDGE - Transmit Drive Edge Register
"XIP_MODE_BITS" on page 363	0xfc	eXecute in Place - Mode bits
DWC_ssi Bridge address block DWC_ssi_bridge_block Exists: SSIC_SPI_BRIDGE == 1		
"CTRLR0" on page 365	0x0	Control Register 0
"SSIENR" on page 367	0x8	SSI Enable Register
"RXFBTR" on page 368	0x14	Receive FIFO Burst Threshold Register

Register	Offset	Description
"TXFTLR" on page 369	0x18	Transmit FIFO Threshold Level
"RXFTLR" on page 370	0x1c	Receive FIFO Threshold Level
"SR" on page 371	0x28	Status Register
"IMR" on page 372	0x2c	Interrupt Mask Register
"ISR" on page 374	0x30	Interrupt Status Register
"RISR" on page 376	0x34	Raw Interrupt Status Register
"TXUICR" on page 379	0x38	Transmit FIFO Underflow Interrupt Clear Register
"RXOICR" on page 380	0x3c	Receive FIFO Overflow Interrupt Clear Register
"SPIMECR" on page 381	0x40	SPI Controller Error interrupt Clear Register
"AHBECR" on page 382	0x44	AHB Error Clear Register
"ICR" on page 383	0x48	Interrupt Clear Register
"IDR" on page 384	0x58	Identification Register
"SSIC_VERSION_ID" on page 385	0x5c	DWC_ssi component version
DWC_ssi register address block - 2 DWC_ssi_address_block2 Exists: SSIC_SPI_BRIDGE == 0		
"XIP_INCR_INST" on page 387	0x100	XIP_INCR_INST - XIP INCR transfer opcode
"XIP_WRAP_INST" on page 388	0x104	XIP_WRAP_INST - XIP WRAP transfer opcode
"XIP_CTRL" on page 389	0x108	XIP_CTRL - XIP Control Register
"XIP_SER" on page 395	0x10c	Target Enable Register
"XRXOICR" on page 396	0x110	XIP Receive FIFO Overflow Interrupt Clear Register
"XIP_CNT_TIME_OUT" on page 397	0x114	XIP time out register for continuous transfers
"SPI_CTRLR1" on page 398	0x118	SPI Control 1 register
"SPITECR" on page 400	0x11c	SPI Transmit Error Interrupt Clear Register
"SPIDR" on page 401	0x120	SPI Device Register
"SPIAR" on page 402	0x124	SPI Device Address Register
"AXIAR0" on page 403	0x128	AXI Address Register 0
"AXIAR1" on page 404	0x12c	AXI Address Register 1
"AXIECR" on page 405	0x130	AXI Manager Error Interrupt Clear Register
"DONECR" on page 406	0x134	Transfer Done Clear Interrupt Clear Register

Register	Offset	Description
" <a href="#">XIP_WRITE_INCR_INST</a> " on page <a href="#">407</a>	0x140	XIP_WRITE_INCR_INST - XIP Write INCR transfer opcode
" <a href="#">XIP_WRITE_WRAP_INST</a> " on page <a href="#">408</a>	0x144	XIP_WRITE_WRAP_INST - XIP Write WRAP transfer opcode
" <a href="#">XIP_WRITE_CTRL</a> " on page <a href="#">409</a>	0x148	XIP_WRITE_CTRL - XIP Write Control Register
" <a href="#">SAFETYCR</a> " on page <a href="#">413</a>	0x180	Safety Control register
" <a href="#">SLVIFFSMTOCR</a> " on page <a href="#">416</a>	0x184	Subordinate interface Clock domain FSM time out register
" <a href="#">SSIFSMTOCR</a> " on page <a href="#">417</a>	0x188	SSI Clock domain FSM time out register
" <a href="#">AXIFSMTOCR</a> " on page <a href="#">419</a>	0x18c	AXI Clock domain FSM time out register
" <a href="#">SAFETYISR</a> " on page <a href="#">420</a>	0x1a0	Safety Interrupt Status Register
" <a href="#">SAFETYICR</a> " on page <a href="#">424</a>	0x1a4	Safety Interrupt Clear Register

## 6.1 **DWC\_ssi\_address\_block Registers**

### 6.1.1 CTRLR0

- **Name:** Control Register 0
- **Description:** This register controls the serial data transfer. It is impossible to write to this register when the DWC\_ssi is enabled.
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always

SSI_IS_MST	31
RSVD_CTRLR0_27_30	30:27
CLK_LOOP_EN	26
SPI_DWS_EN	25
SPI_HYPERBUS_EN	24
SPI_FRF	23:22
RSVD_CTRLR0_20_21	21:20
CFS	19:16
RSVD_CTRLR0_15	15
SSTE	14
SRL	13
SLV_OE	12
TMOD	11:10
SCPOL	9
SCP自称	8
FRF	7:6
RSVD_CTRLR0_5	5
DFS	4:0

Table 6-5 Fields for Register: CTRLR0

Bits	Name	Memory Access	Description
31	SSI_IS_MST	* Varies	<p>This field selects if DWC_ssi is working in Controller or Target mode</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (CONTROLLER): DWC_ssi is Controller</li> <li>■ 0x0 (TARGET): DWC_ssi is Target</li> </ul> <p><b>Value After Reset:</b> "(SSIC_IS_MASTER!=2) ? \"0x0\" : \"SSIC_IS_DFLT_MST\""</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_IS_MASTER!=2) ? \"read-only\" : \"read-write\""</p>
30:27	RSVD_CTRLR0_27_30	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
26	CLK_LOOP_EN	* Varies	<p>Clock loop back enable bit. Once this bit is set to 1, DWC_ssi will use looped back clock (mst_sclk_in) to capture read data</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (DISABLE): Disable Clock Loop Back</li> <li>■ 0x1 (ENABLE): Enable Clock Loop Back</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_CLK_LOOP_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_CLK_LOOP_BACK==0) ? \"read-only\": \"read-write\""</p>
25	SPI_DWS_EN	* Varies	<p>Enable Dynamic wait states in SPI mode of operation. This field is only applicable when CTRLR0.FRF is set to 0 (Motorola SPI Frame Format).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (DISABLE): Disable SPI Dynamic Wait State</li> <li>■ 0x1 (ENABLE): Enable SPI Dynamic Wait State</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_SPI_DWS_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_SPI_DYN_WS_EN==0) ? \"read-only\": \"read-write\""</p>
24	SPI_HYPERBUS_EN	* Varies	<p>SPI Hyperbus Frame format enable.</p> <p>Selects if data frame format for Transmitting/Receiving the data is in Hyperbus mode. This field is effective only when CTRLR0.FRF is set to SPI frame format.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (DISABLE): Disable Hyperbus Format</li> <li>■ 0x1 (ENABLE): Enable Hyperbus Format</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_HYPERBUS_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HYPERBUS_EN==0) ? \"read-only\": \"read-write\""</p>
23:22	SPI_FRF	* Varies	<p>SPI Frame Format</p> <p>Selects data frame format for Transmitting/Receiving the data. Bits only valid when SSIC_SPI_MODE is either set to "Dual" or "Quad" or "Octal" mode.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (SPI_STANDARD): Standard SPI Format</li> <li>■ 0x1 (SPI_DUAL): Dual SPI Format</li> <li>■ 0x2 (SPI_QUAD): Quad SPI Format</li> <li>■ 0x3 (SPI_OCTAL): Octal SPI Format</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_SPI_FRF</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_SPI_MODE==0 &amp;&amp; SSIC_SLV_SPI_MODE==0) ? \"read-only\": \"read-write\""</p>

Bits	Name	Memory Access	Description
21:20	RSVD_CTRLR0_20_21	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
19:16	CFS	R/W	<p>Control Frame Size.</p> <p>Selects the length of the control word for the Microwire frame format.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (SIZE_01_BIT): 01-bit Control Word</li> <li>■ 0x1 (SIZE_02_BIT): 02-bit Control Word</li> <li>■ 0x2 (SIZE_03_BIT): 03-bit Control Word</li> <li>■ 0x3 (SIZE_04_BIT): 04-bit Control Word</li> <li>■ 0x4 (SIZE_05_BIT): 05-bit Control Word</li> <li>■ 0x5 (SIZE_06_BIT): 06-bit Control Word</li> <li>■ 0x6 (SIZE_07_BIT): 07-bit Control Word</li> <li>■ 0x7 (SIZE_08_BIT): 08-bit Control Word</li> <li>■ 0x8 (SIZE_09_BIT): 09-bit Control Word</li> <li>■ 0x9 (SIZE_10_BIT): 10-bit Control Word</li> <li>■ 0xa (SIZE_11_BIT): 11-bit Control Word</li> <li>■ 0xb (SIZE_12_BIT): 12-bit Control Word</li> <li>■ 0xc (SIZE_13_BIT): 13-bit Control Word</li> <li>■ 0xd (SIZE_14_BIT): 14-bit Control Word</li> <li>■ 0xe (SIZE_15_BIT): 15-bit Control Word</li> <li>■ 0xf (SIZE_16_BIT): 16-bit Control Word</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_CFS</p> <p><b>Exists:</b> Always</p>
15	RSVD_CTRLR0_15	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
14	SSTE	R/W	<p>Target Select Toggle Enable.</p> <p>While operating in SPI mode with clock phase (SCPH) set to 0, this register controls the behavior of the chip select line (ss_*_n) between data frames.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (TOGGLE_EN): 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</li> <li>■ 0x0 (TOGGLE_DISABLE): ss_*_n will stay low and sclk will run continuously for the duration of the transfer</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_TGL_EN</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
13	SRL	* Varies	<p>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-target and serial-controller modes. When the DWC_ssi is configured as a target in loopback mode, the ss_in_n and ssi_clk signals must be provided by an external source. In this mode, the target cannot generate these signals because there is nothing to which to loop back.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (TESTING_MODE): Test Mode Operation</li> <li>■ 0x0 (NORMAL_MODE): Normal mode operation</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_TX_RX_EN==1) ? \"read-write\" : \"read-only\""</p>
12	SLV_OE	R/W	<p>Target Output Enable. Relevant only when the DWC_ssi is configured as a serial-target device. When configured as a serial controller, this bit field has no functionality. This bit enables or disables the setting of the ssi_oe_n output from the DWC_ssi serial target. 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 target, a high impedance state is always present on the target txd output when SLV_OE = 1. This is useful when the controller transmits in broadcast mode (controller transmits data to all target devices). Only one target may respond with data on the controller 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.</p> <p>When SSIC_SLV_SPI_MODE is set to 1 and SPI is programmed to work in Enhanced SPI mode, then for correct operation this bit should be programmed to 0.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (DISABLED): Target Output is disabled</li> <li>■ 0x0 (ENABLED): Target Output is enabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
11:10	TMOD	R/W	<p>Transfer Mode.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (TX_AND_RX): Transmit &amp; Receive; Not Applicable in enhanced SPI operating mode or when SSIC_HAS_TX_RX_EN is set to 0</li> <li>■ 0x1 (TX_ONLY): Transmit only mode; Or Write in enhanced SPI operating mode</li> <li>■ 0x2 (RX_ONLY): Receive only mode; Or Read in enhanced SPI operating mode</li> <li>■ 0x3 (EEPROM_READ): EEPROM Read mode; Not Applicable in enhanced SPI operating mode</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_TMOD</p> <p><b>Exists:</b> Always</p>
9	SCPOL	* Varies	<p>Serial Clock Polarity.</p> <p>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 DWC_ssi controller is not actively transferring data on the serial bus.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (INACTIVE_HIGH): Inactive state of serial clock is low</li> <li>■ 0x1 (INACTIVE_LOW): Inactive state of serial clock is high</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_SCPOL</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HC_FRF==0) ? \"read-write\" : \"read-only\""</p>

Bits	Name	Memory Access	Description
8	SCPH	* Varies	<p>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 chip 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 chip select line is activated, and data are captured on the second edge of the serial clock.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (START_BIT): Serial clock toggles at start of first bit</li> <li>■ 0x0 (MIDDLE_BIT): Serial clock toggles in middle of first bit</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_SCPH</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HC_FRF==0) ? \"read-write\" : \"read-only\""</p>
7:6	FRF	* Varies	<p>Frame Format. Selects which serial protocol transfers the data.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (SPI): Motorola SPI Frame Format</li> <li>■ 0x1 (SSP): Texas Instruments SSP Frame Format</li> <li>■ 0x2 (MICROWIRE): National Semiconductors Microwire Frame Format</li> <li>■ 0x3 (RESERVED): Reserved</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_FRF</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HC_FRF==0) ? \"read-write\" : \"read-only\""</p>
5	RSVD_CTRLR0_5	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
4:0	DFS	R/W	<p>Data Frame Size.</p> <p>Selects the data frame length. When the data frame size is programmed to be less than 32 bits, the receive data is automatically right-justified by the receive logic, with the upper bits of the receive FIFO zero-padded.</p> <p>You must right-justify transmit data before writing into the transmit FIFO. The transmit logic ignores the upper unused bits when transmitting the data.</p> <p><b>Note:</b> When SSIC_SPI_MODE is set to "Dual", "Quad" or "Octal" mode and SPI_FRF is not set to 2'b00:</p> <ul style="list-style-type: none"> <li>■ DFS value must be a multiple of 2 if SPI_FRF = 01</li> <li>■ DFS value must be multiple of 4 if SPI_FRF = 10</li> <li>■ DFS value must be multiple of 8 if SPI_FRF = 11</li> </ul> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (DFS_01_BIT): Reserved</li> <li>■ 0x1 (DFS_02_BIT): Reserved</li> <li>■ 0x2 (DFS_03_BIT): Reserved</li> <li>■ 0x3 (DFS_04_BIT): 04-bit serial data transfer</li> <li>■ 0x4 (DFS_05_BIT): 05-bit serial data transfer</li> <li>■ 0x5 (DFS_06_BIT): 06-bit serial data transfer</li> <li>■ 0x6 (DFS_07_BIT): 07-bit serial data transfer</li> <li>■ 0x7 (DFS_08_BIT): 08-bit serial data transfer</li> <li>■ 0x8 (DFS_09_BIT): 09-bit serial data transfer</li> <li>■ 0x9 (DFS_10_BIT): 10-bit serial data transfer</li> <li>■ 0xa (DFS_11_BIT): 11-bit serial data transfer</li> <li>■ 0xb (DFS_12_BIT): 12-bit serial data transfer</li> <li>■ 0xc (DFS_13_BIT): 13-bit serial data transfer</li> <li>■ 0xd (DFS_14_BIT): 14-bit serial data transfer</li> <li>■ 0xe (DFS_15_BIT): 15-bit serial data transfer</li> <li>■ 0xf (DFS_16_BIT): 16-bit serial data transfer</li> <li>■ 0x10 (DFS_17_BIT): 17-bit serial data transfer</li> </ul>

Bits	Name	Memory Access	Description
4:0...(co nt.)	DFS.	R/W	<ul style="list-style-type: none"> <li>■ 0x11 (DFS_18_BIT): 18-bit serial data transfer</li> <li>■ 0x12 (DFS_19_BIT): 19-bit serial data transfer</li> <li>■ 0x13 (DFS_20_BIT): 20-bit serial data transfer</li> <li>■ 0x14 (DFS_21_BIT): 21-bit serial data transfer</li> <li>■ 0x15 (DFS_22_BIT): 22-bit serial data transfer</li> <li>■ 0x16 (DFS_23_BIT): 23-bit serial data transfer</li> <li>■ 0x17 (DFS_24_BIT): 24-bit serial data transfer</li> <li>■ 0x18 (DFS_25_BIT): 25-bit serial data transfer</li> <li>■ 0x19 (DFS_26_BIT): 26-bit serial data transfer</li> <li>■ 0x1a (DFS_27_BIT): 27-bit serial data transfer</li> <li>■ 0x1b (DFS_28_BIT): 28-bit serial data transfer</li> <li>■ 0x1c (DFS_29_BIT): 29-bit serial data transfer</li> <li>■ 0x1d (DFS_30_BIT): 30-bit serial data transfer</li> <li>■ 0x1e (DFS_31_BIT): 31-bit serial data transfer</li> <li>■ 0x1f (DFS_32_BIT): 32-bit serial data transfer</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_DFS</p> <p><b>Exists:</b> Always</p>

## 6.1.2 CTRLR1

- **Name:** Control Register 1
- **Description:** This register exists only when the DWC\_ssi is configured as a controller device. When the DWC\_ssi is configured as a serial target, 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 DWC\_ssi is enabled.
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** SSIC\_IS\_MASTER != 0

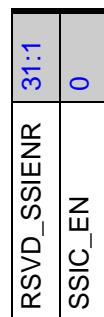


**Table 6-6 Fields for Register: CTRLR1**

Bits	Name	Memory Access	Description
31:16	RSVD_CTRLR1	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
15:0	NDF	R/W	<p>Number of Data Frames.</p> <p>When TMOD = 10 or TMOD = 11 , this register field sets the number of data frames to be continuously received by the DWC_ssi. The DWC_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 256 KB of data in a continuous transfer.</p> <p>When SPI_CTRLR0.CLK_STRETCH_EN=1 and TMOD = 01, this register field sets the number of data frames to be continuously transmitted by DWC_ssi. If the Transmit FIFO goes empty in-between, DWC_ssi masks the serial clock (sclk_out) and wait for rest of the data until the programmed amount of frames are transferred successfully.</p> <p>When the DWC_ssi is configured as a serial target, the transfer continues for as long as the target is selected.</p> <p>Therefore, this register serves no purpose and is not present when the DWC_ssi is configured as a serial target.</p> <p><b>Value After Reset:</b> SSIC_DFLT_NDF</p> <p><b>Exists:</b> Always</p>

### 6.1.3 SSIENR

- **Name:** SSI Enable Register
- **Description:** This register enables and disables the DWC\_ssi.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always

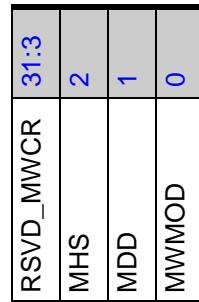


**Table 6-7 Fields for Register: SSIENR**

Bits	Name	Memory Access	Description
31:1	RSVD_SSIENR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
0	SSIC_EN	R/W	SSI Enable. Enables and disables all DWC_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 DWC_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. <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (ENABLED): Enables DWC_ssi</li> <li>■ 0x0 (DISABLE): Disables DWC_ssi</li> </ul> <b>Value After Reset:</b> SSIC_BOOT_MODE_EN <b>Exists:</b> Always

### 6.1.4 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 DWC\_ssi is enabled.
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** Always



**Table 6-8 Fields for Register: MWCR**

Bits	Name	Memory Access	Description
31:3	RSVD_MWCR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
2	MHS	* Varies	Microwire Handshaking. Relevant only when the DWC_ssi is configured as a serial-controller device. When configured as a serial target, this bit field has no functionality. Used to enable and disable the busy/ready handshaking interface for the Microwire protocol. When enabled, the DWC_ssi checks for a ready status from the target target, after the transfer of the last data/control bit, before clearing the BUSY status in the SR register. <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (ENABLED): handshaking interface is enabled</li> <li>■ 0x0 (DISABLE): handshaking interface is disabled</li> </ul> <b>Value After Reset:</b> SSIC_IS_DFLT_MHS <b>Exists:</b> Always <b>Memory Access:</b> { (SSIC_IS_MASTER != 0) ? "read-write" : "read-only" }

Bits	Name	Memory Access	Description
1	MDD	R/W	<p>Microwire Control.</p> <p>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 DWC_ssi MacroCell from the external serial device. When this bit is set to 1, the data word is transmitted from the DWC_ssi MacroCell to the external serial device.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (TRANSMIT): SSI transmits data</li> <li>■ 0x0 (RECEIVE): SSI receives data</li> </ul> <p><b>Value After Reset:</b> SSIC_IS_DFLT_MDD</p> <p><b>Exists:</b> Always</p>
0	MWMOD	R/W	<p>Microwire Transfer Mode.</p> <p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (SEQUENTIAL): Sequential Transfer</li> <li>■ 0x0 (NON_SEQUENTIAL): Non-Sequential Transfer</li> </ul> <p><b>Value After Reset:</b> SSIC_IS_DFLT_MWMOD</p> <p><b>Exists:</b> Always</p>

### 6.1.5 SER

- **Name:** Target Enable Register
- **Description:** This register is valid only when the DWC\_ssi is configured as a controller device. When the DWC\_ssi is configured as a serial target, writing to this location has no effect; reading from this location returns 0. The register enables the individual chip select output lines from the DWC\_ssi controller. Up to 16 chip-select output pins are available on the DWC\_ssi controller. You cannot write to this register when DWC\_ssi is busy and when SSIC\_EN = 1.
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** SSIC\_IS\_MASTER != 0

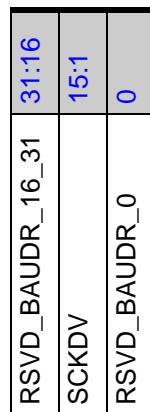


**Table 6-9 Fields for Register: SER**

Bits	Name	Memory Access	Description
31:y	RSVD_SER	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[y]:</b> SSIC_NUM_SLAVES</p>
x:0	SER	R/W	<p>Chip Select Enable Flag.</p> <p>Each bit in this register corresponds to a chip select line (ss_x_n) from the DWC_ssi controller. When a bit in this register is set (1), the corresponding chip select line from the controller 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 chip select outputs until a transfer is started. Before beginning a transfer, you should enable the bit in this register that corresponds to the target device with which the controller wants to communicate. When not operating in broadcast mode, only one bit in this field should be set.</p> <p><b>Value After Reset:</b> SSIC_DFLT_SLV</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[x]:</b> SSIC_NUM_SLAVES - 1</p>

## 6.1.6 BAUDR

- **Name:** Baud Rate Select
- **Description:** This register is valid only when the DWC\_ssi is configured as a controller device. When the DWC\_ssi is configured as a serial target, 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 DWC\_ssi is enabled.
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** SSIC\_IS\_MASTER != 0



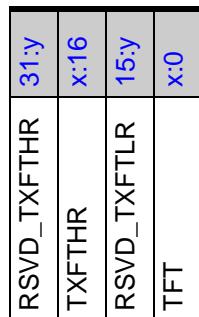
**Table 6-10 Fields for Register: BAUDR**

Bits	Name	Memory Access	Description
31:16	RSVD_BAUDR_16_31	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
15:1	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: $Fsclk_{out} = Fssi_{clk}/BAUDR$ where BAUDR is any even value between 2 and 65534 (BAUDR = {SCKDV*2}). For example: for Fssi_clk = 3.6864MHz and BAUDR = 2 Fsclk_out = 3.6864/2 = 1.8432MHz <b>Value After Reset:</b> SSIC_DFLT_BAUDR/2 <b>Exists:</b> Always

Bits	Name	Memory Access	Description
0	RSVD_BAUDR_0	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always

### 6.1.7 TXFTLR

- **Name:** Transmit FIFO Threshold Level
- **Description:** This register controls the threshold value for the transmit FIFO memory..
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** Always



**Table 6-11 Fields for Register: TXFTLR**

Bits	Name	Memory Access	Description
31:y	RSVD_TXFTHR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[y]:</b> TXFTHR_W + 16</p>
x:16	TXFTHR	* Varies	<p>Transfer start FIFO level.</p> <p>Used to control the level of entries in transmit FIFO above which transfer will start on serial line. This register can be used to ensure that sufficient data is present in transmit FIFO before starting a write operation on serial line.</p> <p>In Internal DMA mode, this field sets the minimum amount of data frames present in the FIFO after which DWC_ssi starts the transfer.</p> <p>This field is valid only for Controller mode of operation.</p> <p><b>Value After Reset:</b> "(SSIC_IS_MASTER!=0) ? SSIC_DFLT_TXFTHR : 0"</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[x]:</b> TXFTHR_W + 15</p> <p><b>Memory Access:</b> "(SSIC_IS_MASTER!=0) ? \"read-write\" : \"read-only\""</p>
15:y	RSVD_TXFTLR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[y]:</b> TX_ABW</p>

Bits	Name	Memory Access	Description
x:0	TFT	R/W	<p>Transmit FIFO Threshold.</p> <p>Controls the level of entries (or below) at which the transmit FIFO controller triggers an interrupt. The FIFO depth is configurable in the range 8-256; 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.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[x]:</b> TX_ABW - 1</p>

### 6.1.8 RXFTLR

- **Name:** Receive FIFO Threshold Level
- **Description:** This register controls the threshold value for the receive FIFO memory..
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** Always



**Table 6-12 Fields for Register: RXFTLR**

Bits	Name	Memory Access	Description
31:y	RSVD_RXFTLR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[y]:</b> RX_ABW</p>
x:0	RFT	R/W	<p>Receive FIFO Threshold.</p> <p>Controls the level of entries (or above) at which the receive FIFO controller triggers an interrupt. The FIFO depth is configurable in the range 8-256. 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.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[x]:</b> RX_ABW - 1</p>

### 6.1.9 TXFLR

- **Name:** Transmit FIFO Level Register
- **Description:** This register contains the number of valid data entries in the transmit FIFO memory.
- **Size:** 32 bits
- **Offset:** 0x20
- **Exists:** Always



**Table 6-13 Fields for Register: TXFLR**

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

### 6.1.10 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 read at any time.
- **Size:** 32 bits
- **Offset:** 0x24
- **Exists:** Always

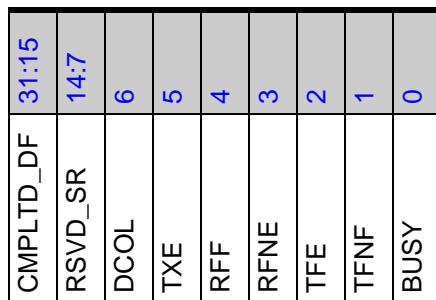


**Table 6-14 Fields for Register: RXFLR**

Bits	Name	Memory Access	Description
31:y	RSVD_RXFLR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p> <p><b>Range Variable[y]:</b> RX_ABW + 1</p>
x:0	RXTFL	R	<p>Receive FIFO Level.</p> <p>Contains the number of valid data entries in the receive FIFO.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p> <p><b>Range Variable[x]:</b> RX_ABW</p>

### 6.1.11 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.
- **Size:** 32 bits
- **Offset:** 0x28
- **Exists:** Always



**Table 6-15 Fields for Register: SR**

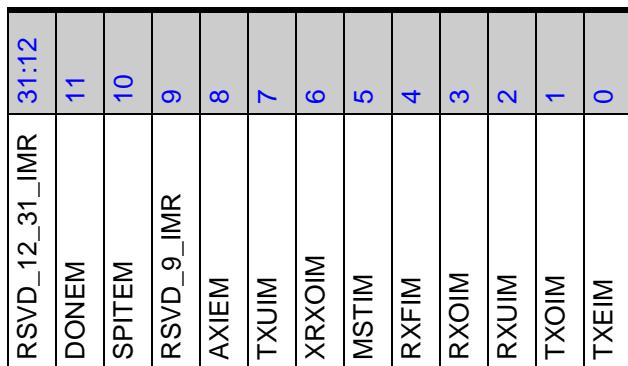
Bits	Name	Memory Access	Description
31:15	CMPLTD_DF	R	Completed Data frames This field indicates total data frames transferred in the previous internal DMA transfer <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
14:7	RSVD_SR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
6	DCOL	R	Data Collision Error. Relevant only when the DWC_ssi is configured as a controller device. This bit will be set if ss_in_n input is asserted by other controller, when the DWC_ssi controller 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. <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (TX_COLLISION_ERROR): Transmit Data Collision Error</li> <li>■ 0x0 (NO_ERROR_CONDITION): No Error</li> </ul> <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

Bits	Name	Memory Access	Description
5	TXE	R	<p>Transmission Error. Set if the transmit FIFO is empty when a transfer is started. This bit can be set only when the DWC_ssi is configured as a target device. Data from the previous transmission is resent on the txd line. This bit is cleared when read.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (TX_ERROR): Transmission Error</li> <li>■ 0x0 (NO_ERROR): No Error</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
4	RFF	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (FULL): Receive FIFO is full</li> <li>■ 0x0 (NOT_FULL): Receive FIFO is not full</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
3	RFNE	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (NOT_EMPTY): Receive FIFO is not empty</li> <li>■ 0x0 (EMPTY): Receive FIFO is empty</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
2	TFE	R	<p>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.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (EMPTY): Transmit FIFO is empty</li> <li>■ 0x0 (NOT_EMPTY): Transmit FIFO is not empty</li> </ul> <p><b>Value After Reset:</b> 0x1</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
1	TFNF	R	<p>Transmit FIFO Not Full. Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (NOT_FULL): Tx FIFO is not Full</li> <li>■ 0x0 (FULL): Tx FIFO is full</li> </ul> <p><b>Value After Reset:</b> 0x1</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	BUSY	R	<p>SSI Busy Flag. When set, indicates that a serial transfer is in progress; when cleared indicates that the DWC_ssi is idle or disabled.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): DWC_ssi is actively transferring data</li> <li>■ 0x0 (INACTIVE): DWC_ssi is idle or disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.1.12 IMR

- **Name:** Interrupt Mask Register
- **Description:** This read/write register masks or enables all interrupts generated by the DWC\_ssi. When the DWC\_ssi is configured as a target device, the MSTIM bit field is not present. This changes the reset value from 0x3F for serial-controller configurations to 0x1F for serial-target configurations.
- **Size:** 32 bits
- **Offset:** 0x2c
- **Exists:** Always



**Table 6-16 Fields for Register: IMR**

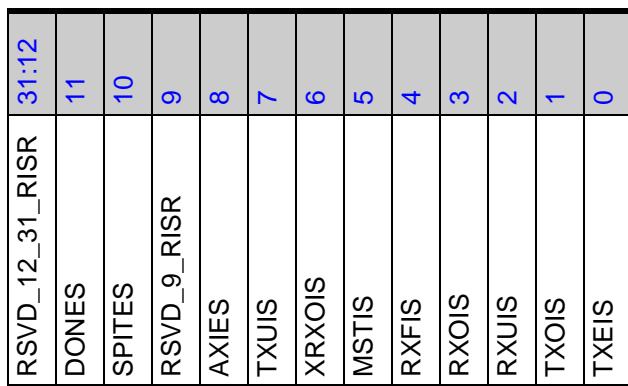
Bits	Name	Memory Access	Description
31:12	RSVD_12_31_IMR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
11	DONEM	* Varies	<b>SSI Done Interrupt Mask</b> <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_done_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_done_intr interrupt is masked</li> </ul> <b>Value After Reset:</b> { (SSIC_HAS_DMA==2) ? 1 : 0 } <b>Exists:</b> Always <b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""
10	SPITEM	* Varies	<b>SPI Transmit Error Interrupt Mask</b> <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_spite_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_spite_intr interrupt is masked</li> </ul> <b>Value After Reset:</b> { (SSIC_SPI_DYN_WS_EN==1) ? 1 : 0 } <b>Exists:</b> Always <b>Memory Access:</b> "(SSIC_SPI_DYN_WS_EN==1) ? \"read-write\" : \"read-only\""

Bits	Name	Memory Access	Description
9	RSVD_9_IMR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
8	AXIEM	* Varies	<p>AXI Error Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_axie_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_axie_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> {(SSIC_HAS_DMA==2) ? 1 : 0}</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""</p>
7	TXUIM	* Varies	<p>Transmit FIFO Underflow Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_txu_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_txu_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> "(SSIC_HAS_DMA==2    SSIC_XIP_WRITE_EN==1) ? 1 : 0"</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==2    SSIC_XIP_WRITE_EN==1) ? \"read-write\" : \"read-only\""</p>
6	XRXOIM	* Varies	<p>XIP Receive FIFO Overflow Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_xrxo_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_xrxo_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> {(SSIC_CONCURRENT_XIP_EN==1) ? 1 : 0}</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_CONCURRENT_XIP_EN==1) ? \"read-write\" : \"read-only\""</p>
5	MSTIM	* Varies	<p>Multi-Controller Contention Interrupt Mask. This bit field is not present if the DWC_ssi is configured as a serial-controller device.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_mst_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_mst_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> {(SSIC_IS_MASTER!=0) ? 1 : 0}</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_IS_MASTER!=0) ? \"read-write\" : \"read-only\""</p>

Bits	Name	Memory Access	Description
4	RXFIM	R/W	<p>Receive FIFO Full Interrupt Mask 0 - ssi_rxf_intr interrupt is masked 1 - ssi_rxf_intr interrupt is not masked</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_rxf_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_rxf_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p> <p><b>Exists:</b> Always</p>
3	RXOIM	R/W	<p>Receive FIFO Overflow Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_rxo_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_rxo_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p> <p><b>Exists:</b> Always</p>
2	RXUIM	R/W	<p>Receive FIFO Underflow Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_rxu_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_rxu_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p> <p><b>Exists:</b> Always</p>
1	TXOIM	R/W	<p>Transmit FIFO Overflow Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_txo_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_txo_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p> <p><b>Exists:</b> Always</p>
0	TXEIM	R/W	<p>Transmit FIFO Empty Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_txe_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_txe_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> SSIC_IS_DFLT_TXEIM</p> <p><b>Exists:</b> Always</p>

### **6.1.13 ISR**

- **Name:** Interrupt Status Register
  - **Description:** This register reports the status of the DWC\_ssi interrupts after they have been masked.
  - **Size:** 32 bits
  - **Offset:** 0x30
  - **Exists:** Always



**Table 6-17 Fields for Register: ISR**

Bits	Name	Memory Access	Description
31:12	RSVD_12_31_RISR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
11	DONE\$	R	SSI Done Interrupt Status <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_done_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_done_intr interrupt is not active after masking</li> </ul> <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
10	SPITES	R	SPI Transmit Error Interrupt <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_spite_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_spite_intr interrupt is not active after masking</li> </ul> <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

Bits	Name	Memory Access	Description
9	RSVD_9_RISR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
8	AXIES	R	<p>AXI Error Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_axie_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_axie_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
7	TXUIS	R	<p>Transmit FIFO Underflow Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txu_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_txu_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
6	XRXOIS	R	<p>XIP Receive FIFO Overflow Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_xrxo_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_xrxo_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
5	MSTIS	R	<p>Multi-Controller Contention Interrupt Status. This bit field is not present if the DWC_ssi is configured as a serial-target device.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_mst_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_mst_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
4	RXFIS	R	<p>Receive FIFO Full Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxf_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_rxf_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
3	RXOIS	R	<p>Receive FIFO Overflow Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxo_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_rxo_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
2	RXUIS	R	<p>Receive FIFO Underflow Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxu_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_rxu_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
1	TXOIS	R	<p>Transmit FIFO Overflow Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txo_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_txo_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	TXEIS	R	<p>Transmit FIFO Empty Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txe_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_txe_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> "(SSIC_IS_DFLT_TXEIM == 1 &amp;&amp; SSIC_BOOT_MODE_EN == 1)"</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### **6.1.14 RISR**

- **Name:** Raw Interrupt Status Register
  - **Description:** Raw Interrupt Status Register
  - **Size:** 32 bits
  - **Offset:** 0x34
  - **Exists:** Always

RSVD_12_31_RISR	31:12
DONER	11
SPLITTER	10
RSVD_9_RISR	9
AXIIR	8
TXUIR	7
XRXOIR	6
MSTIR	5
RXFIR	4
RXOIR	3
RXUIR	2
TXOIR	1
TXEIR	0

**Table 6-18 Fields for Register: RISR**

Bits	Name	Memory Access	Description
31:12	RSVD_12_31_RISR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
11	DONER	R	SSI Done Interrupt Raw Status <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_done_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_done_intr interrupt is not active prior to masking</li> </ul> <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

Bits	Name	Memory Access	Description
10	SPITER	R	<p>SPI Transmit Error Interrupt status. This bit gets set, If SPI Controller fails to get a READY status from the target until the amount of time defined in SPI_CTRLR1.MAX_WS field, then it will stop the SPI transfer and the FIFO is flushed (in case of write operation).</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_spiter_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_spiter_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
9	RSVD_9_RISR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
8	AXIER	R	<p>AXI Error Interrupt Raw Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_axier_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_axier_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
7	TXUIR	R	<p>Transmit FIFO Underflow Interrupt Raw Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txuir_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_txuir_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
6	XRXOIR	R	<p>XIP Receive FIFO Overflow Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_xrxo_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_xrxo_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
5	MSTIR	R	<p>Multi-Controller Contention Raw Interrupt Status. This bit field is not present if the DWC_ssi is configured as a serial-target device.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_mst_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_mst_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
4	RXFIR	R	<p>Receive FIFO Full Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxf_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_rxf_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
3	RXOIR	R	<p>Receive FIFO Overflow Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxo_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_rxo_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
2	RXUIR	R	<p>Receive FIFO Underflow Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxu_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_rxu_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
1	TXOIR	R	<p>Transmit FIFO Overflow Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txo_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_txo_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	TXEIR	R	<p>Transmit FIFO Empty Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txe_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_txe_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> SSIC_BOOT_MODE_EN</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.1.15 TXEICR

- **Name:** Transmit FIFO Error Interrupt Clear Registers
- **Description:** Transmit FIFO Error Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x38
- **Exists:** Always



**Table 6-19 Fields for Register: TXEICR**

Bits	Name	Memory Access	Description
31:1	RSVD_TXEICR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	TXEICR	R	<p>Clear Transmit FIFO Overflow/Underflow Interrupt.</p> <p>This register reflects the status of the interrupt. A read from this register clears the ssi_txo_intr/ssi_txu_intr interrupt; writing has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.1.16 RXOICR

- **Name:** Receive FIFO Overflow Interrupt Clear Register
- **Description:** Receive FIFO Overflow Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x3c
- **Exists:** Always



**Table 6-20 Fields for Register: RXOICR**

Bits	Name	Memory Access	Description
31:1	RSVD_RXOICR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> 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. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

### 6.1.17 RXUICR

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



**Table 6-21 Fields for Register: RXUICR**

Bits	Name	Memory Access	Description
31:1	RSVD_RXUICR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> 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. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

### 6.1.18 MSTICR

- **Name:** Multi-Controller Interrupt Clear Register
- **Description:** Multi-Controller Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x44
- **Exists:** SSIC\_IS\_MASTER > 0

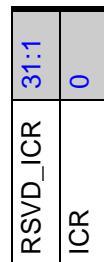


**Table 6-22 Fields for Register: MSTICR**

Bits	Name	Memory Access	Description
31:1	RSVD_MSTICR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
0	MSTICR	R	Clear Multi-Controller 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. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

### 6.1.19 ICR

- **Name:** Interrupt Clear Register
- **Description:** Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x48
- **Exists:** Always



**Table 6-23 Fields for Register: ICR**

Bits	Name	Memory Access	Description
31:1	RSVD_ICR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	ICR	R	<p>Clear Interrupts.</p> <p>This register is set if any of the interrupts below are active. A read clears the ssi_done_intr, ssi_axie_intr, ssi_spitie_intr, ssi_txo_intr, ssi_txu_intr, ssi_rxu_intr, ssi_xrxo_intr, ssi_rxo_intr and the ssi_mst_intr interrupts. Writing to this register has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.1.20 DMACR

- **Name:** DMA Control Register

- **Description:** DMA Control Register.

This register is only valid when DWC\_ssi is configured with a set of DMA Controller interface signals (SSIC\_HAS\_DMA = 1) or Internal DMA operation (SSIC\_HAS\_DMA = 2). When DWC\_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.

- **Size:** 32 bits

- **Offset:** 0x4c

- **Exists:** SSIC\_HAS\_DMA > 0

RSVD_DMCR	31:y														
AID	x:15														
APROT	14:12														
ACACHE	11:8														
RSVD_DMCR7	7														
AINC	6														
RSVD_DMCR5	5														
ATW	4:3														
IDMAE	2														
TDMAE	1														
RDMAE	0														

Table 6-24 Fields for Register: DMACR

Bits	Name	Memory Access	Description
31:y	RSVD_DMCR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Range Variable[y]:</b> SSIC_AXI_IDW + 15
x:15	AID	* Varies	AXI awid/arid signal value. <b>Value After Reset:</b> SSIC_DFLT_AID <b>Exists:</b> Always <b>Range Variable[x]:</b> SSIC_AXI_IDW + 14 <b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""
14:12	APROT	* Varies	AXI arprot/awprot signal value. <b>Value After Reset:</b> SSIC_DFLT_APROT <b>Exists:</b> Always <b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""

Bits	Name	Memory Access	Description
11:8	ACACHE	* Varies	<p>AXI arcache/awcache signal value.</p> <p><b>Value After Reset:</b> SSIC_DFLT_ACACHE</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""</p>
7	RSVD_DMCR7	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
6	AINC	* Varies	<p>Address Increment. Indicates whether to increment the AXI address on every transfer. 1 = Increment 0 = No Change Note: Increment aligns the address to the next DMACR.ATW boundary</p> <p><b>Value After Reset:</b> SSIC_DFLT_AINC</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""</p>
5	RSVD_DMCR5	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
4:3	ATW	* Varies	<p>AXI transfer width for DMA transfers mapped to arsize/awsize. This value must be less than or equal to SSIC_AXI_DW.</p> <p>Values:</p> <ul style="list-style-type: none"> <li>0x0: 1 byte</li> <li>0x1: 2 bytes</li> <li>0x2: 4 bytes</li> <li>0x3: 8 bytes</li> </ul> <p>Note: When SSIC_AXI_DW is set to 32 bits, if user programs this field to 0x8(3 bytes). DWC_ssi will use 4 bytes as transfer size for the AXI transfers.</p> <p><b>Value After Reset:</b> SSIC_DFLT_ATW</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""</p>
2	IDMAE	* Varies	<p>Internal DMA Enable. This bit should be enabled only when CTRLR0.FRF = 0 (Motorola SPI) and CTRLR0.SPI_FRF &gt; 0.</p> <p><b>Value After Reset:</b> SSIC_DFLT_IDMAE</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""</p>

Bits	Name	Memory Access	Description
1	TDMAE	* Varies	<p>Transmit DMA Enable. This bit enables/disables the transmit FIFO DMA channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ENABLED): Transmit DMA enabled</li> <li>■ 0x0 (DISABLE): Transmit DMA disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==1) ? \"read-write\" : \"read-only\""</p>
0	RDMAE	* Varies	<p>Receive DMA Enable. This bit enables/disables the receive FIFO DMA channel.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ENABLED): Receive DMA enabled</li> <li>■ 0x0 (DISABLE): Receive DMA disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==1) ? \"read-write\" : \"read-only\""</p>

### 6.1.21 DMATDLR

- **Name:** DMA Transmit Data Level
- **Description:** This register is only valid when the DWC\_ssi is configured with a set of DMA interface signals (SSIC\_HAS\_DMA = 1). When DWC\_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.
- **Size:** 32 bits
- **Offset:** 0x50
- **Exists:** SSIC\_HAS\_DMA == 1



**Table 6-25 Fields for Register: DMATDLR**

Bits	Name	Memory Access	Description
31:y	RSVD_DMATDLR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[y]:</b> TX_ABW</p>
x:0	DMATDL	R/W	<p>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.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[x]:</b> TX_ABW - 1</p>

### 6.1.22 AXIAWLEN

- **Name:** Destination Burst Length
- **Description:** Destination Burst Length
- **Size:** 32 bits
- **Offset:** 0x50
- **Exists:** SSIC\_HAS\_DMA == 2



**Table 6-26 Fields for Register: AXIAWLEN**

Bits	Name	Memory Access	Description
31:y	RSVD_AXIAWLEN2	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Range Variable[y]:</b> SSIC_AXI_BLW + 8
x:8	AWLEN	R/W	Destination Burst Length <b>Value After Reset:</b> SSIC_DFLT_AWLEN <b>Exists:</b> Always <b>Range Variable[x]:</b> SSIC_AXI_BLW + 7
7:0	RSVD_AXIAWLEN	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always

### 6.1.23 DMARDLR

- **Name:** DMA Receive Data Level
- **Description:** This register is only valid when DWC\_ssi is configured with a set of DMA interface signals (SSIC\_HAS\_DMA = 1). When DWC\_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.
- **Size:** 32 bits
- **Offset:** 0x54
- **Exists:** SSIC\_HAS\_DMA == 1



**Table 6-27 Fields for Register: DMARDLR**

Bits	Name	Memory Access	Description
31:y	RSVD_DMARDLR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Range Variable[y]:</b> 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. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Range Variable[x]:</b> RX_ABW - 1

### 6.1.24 AXIARLEN

- **Name:** Source Burst Length
- **Description:** Source Burst Length
- **Size:** 32 bits
- **Offset:** 0x54
- **Exists:** SSIC\_HAS\_DMA == 2



**Table 6-28 Fields for Register: AXIARLEN**

Bits	Name	Memory Access	Description
31:y	RSVD_AXIARLEN2	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Range Variable[y]:</b> SSIC_AXI_BLW + 8
x:8	ARLEN	R/W	Source Burst Length <b>Value After Reset:</b> SSIC_DFLT_ARLEN <b>Exists:</b> Always <b>Range Variable[x]:</b> SSIC_AXI_BLW + 7
7:0	RSVD_AXIARLEN	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always

### 6.1.25 IDR

- **Name:** Identification Register
- **Description:** This register contains the peripherals identification code, which is written into the register at configuration time using coreConsultant.
- **Size:** 32 bits
- **Offset:** 0x58
- **Exists:** Always



**Table 6-29 Fields for Register: IDR**

Bits	Name	Memory Access	Description
31:0	IDCODE	R	<p>Identification code. The register contains the identification code of the peripheral, which is written into the register at configuration time using CoreConsultant.</p> <p><b>Value After Reset:</b> SSIC_ID</p> <p><b>Exists:</b> Always</p>

### 6.1.26 SSIC\_VERSION\_ID

- **Name:** DWC\_ssi component version
- **Description:** This read-only register stores the specific DWC\_ssi component version.
- **Size:** 32 bits
- **Offset:** 0x5c
- **Exists:** Always

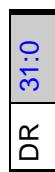


**Table 6-30 Fields for Register: SSIC\_VERSION\_ID**

Bits	Name	Memory Access	Description
31:0	SSIC_COMP_VERSION	R	<p>Contains the hex representation of the Synopsys component version. Consists of ASCII value for each number in the version, followed by *. For example 31_30_33_2A represents the version 1.03*.</p> <p><b>Value After Reset:</b> SSIC_VERSION_ID</p> <p><b>Exists:</b> Always</p>

### 6.1.27 DRx (for i = 0; i <= 35)

- **Name:** DWC\_ssi Data Register
- **Description:** The DWC\_ssi data register is a 32-bit read/write buffer for the transmit/receive FIFOs. 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 SSIC\_EN = 1. FIFOs are reset when SSIC\_EN = 0.  
**Note** The DR register in the DWC\_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 hrdata bus. The FIFO buffers on the DWC\_ssi are not addressable.
- **Size:** 32 bits
- **Offset:** 0x60 +i\*0x4
- **Exists:** Always

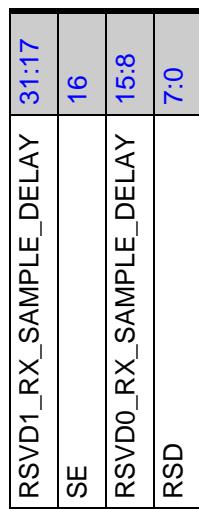


**Table 6-31 Fields for Register: DRx (for i = 0; i <= 35)**

Bits	Name	Memory Access	Description
31:0	DR	R/W	<p>Data Register. When writing to this register, you must right-justify the data. Read data are automatically right-justified.</p> <p>Read = Receive FIFO buffer Write = Transmit FIFO buffer.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always <b>Volatile:</b> true</p>

### 6.1.28 RX\_SAMPLE\_DELAY

- **Name:** RX Sample Delay Register
- **Description:** This register is only valid when the DWC\_ssi is configured with rxd sample delay logic (SSIC\_HAS\_RX\_SAMPLE\_DELAY!=0). When the DWC\_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 DWC\_ssi is enabled.
- **Size:** 32 bits
- **Offset:** 0xf0
- **Exists:** SSIC\_HAS\_RX\_SAMPLE\_DELAY != 0



**Table 6-32 Fields for Register: RX\_SAMPLE\_DELAY**

Bits	Name	Memory Access	Description
31:17	RSVD1_RX_SAMPLE_DELAY	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
16	SE	* Varies	Receive Data (rxd) Sampling Edge. This register is used to decide the sampling edge for RXD signal with ssi_clk. Then this bit is set to 1 then negative edge of ssi_clk will be used to sample the incoming data, otherwise positive edge will be used for sampling. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Memory Access:</b> "(SSIC_HAS_RX_SAMPLE_DELAY==2) ? \"read-write\" : \"read-only\""

Bits	Name	Memory Access	Description
15:8	RSVD0_RX_SAMPLE_DELAY	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
7:0	RSD	R/W	<p>Receive Data (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.</p> <p><b>Note:</b> If this register is programmed with a value that exceeds the depth of the internal shift registers (SSIC_RX_DLY_SR_DEPTH) zero delay will be applied to the rxd sample.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>

### 6.1.29 SPI\_CTRLR0

- **Name:** SPI\_CTRLR0 - SPI Control Register
- **Description:** This register is used to control the serial data transfer in enhanced SPI mode of operation. It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0xf4
- **Exists:** SSIC\_SPI\_MODE != 0

RSVD_SPI_CTRLR0	31
CLK_STRETCH_EN	30
XIP_PREFETCH_EN	29
RSVD_SPI_CTRLR0_28	28
XIP_MBL	27:26
SPI_RXDS_SIG_EN	25
SPI_DM_EN	24
RXDS_VL_EN	23
RSVD_SPI_CTRLR0_22	22
SSIC_XIP_CONT_XFER_EN	21
XIP_INST_EN	20
XIP_DFS_HC	19
SPI_RXDS_EN	18
INST_DDR_EN	17
SPI_DDR_EN	16
WAIT_CYCLES	15:11
RSVD_SPI_CTRLR0_10	10
INST_L	9:8
XIP_MD_BIT_EN	7
RSVD_SPI_CTRLR0_6	6
ADDR_L	5:2
TRANS_TYPE	1:0

Table 6-33 Fields for Register: SPI\_CTRLR0

Bits	Name	Memory Access	Description
31	RSVD_SPI_CTRLR0	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
30	CLK_STRETCH_EN	* Varies	Enables clock stretching capability in SPI transfers. In case of write, if the FIFO becomes empty DWC_ssi will stretch the clock until FIFO has enough data to continue the transfer. In case of read, if the receive FIFO becomes full DWC_ssi will stop the clock until data has been read from the FIFO. <b>Value After Reset:</b> "(SSIC_CLK_STRETCH_EN==0) ? \"0x0\" : \"SSIC_DFLT_CLK_STRETCH\"" <b>Exists:</b> Always <b>Memory Access:</b> "(SSIC_CLK_STRETCH_EN==0) ? \"read-only\" : \"read-write\""

Bits	Name	Memory Access	Description
29	XIP_PREFETCH_EN	* Varies	<p>Enables XIP pre-fetch functionality in DWC_ssi. Once enabled DWC_ssi will pre-fetch data frames from next contiguous location, to reduce the latency for the upcoming contiguous transfer. If the next XIP request is not contiguous then pre-fetched bits will be discarded.</p> <p><b>Value After Reset:</b> "(SSIC_XIP_PREFETCH_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"0x0\": \"SSIC_DFLT_XIP_PREFETCH\""</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_PREFETCH_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"read-only\": \"read-write\""</p>
28	RSVD_SPI_CTRLR0_28	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
27:26	XIP_MBL	* Varies	<p>XIP Mode bits length. Sets the length of mode bits in XIP mode of operation. These bits are valid only when SPI_CTRLR0.XIP_MD_BIT_EN is set to 1.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (MBL_2): Mode bits length equal to 2</li> <li>■ 0x1 (MBL_4): Mode bits length equal to 4</li> <li>■ 0x2 (MBL_8): Mode bits length equal to 8</li> <li>■ 0x3 (MBL_16): Mode bits length equal to 16</li> </ul> <p><b>Value After Reset:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"0x0\": \"SSIC_DFLT_XIP_MBL\""</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"read-only\": \"read-write\""</p>
25	SPI_RXDS_SIG_EN	* Varies	<p>Enable rxds signaling during address and command phase of Hyperbus transfer.</p> <p>This bit enables rxds signaling by Hyperbus target devices during Command-Address (CA) phase. If the rxds signal is set to 1 during the CA phase of transfer, DWC_ssi transmits (2*SPI_CTRLR0.WAIT_CYCLES-1) wait cycles after the address phase is complete.</p> <p><b>Value After Reset:</b> SSIC_DFLT_RXDS_SIG_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HYPERBUS_EN==0) ? \"read-only\": \"read-write\""</p>

Bits	Name	Memory Access	Description
24	SPI_DM_EN	* Varies	<p>SPI data mask enable bit. When this bit is enabled, the <i>txd_dmsignal</i> is used to mask the data on the txd data line. This bit is enabled only when the SSIC_DM_EN parameter is set to 1.</p> <p><b>Value After Reset:</b> SSIC_DFLT_DM_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_SPI_DM_EN==0) ? \"read-only\": \"read-write\""</p>
23	RXDS_VL_EN	* Varies	<p>RXDS variable latency enable bit. When this bit is set DWC_ssi waits for all the data to be sampled before stopping the SCLK_OUT clock. This enables the support for the devices which support variable latencies within the transfers for RXDS transfers.</p> <p><b>Value After Reset:</b> SSIC_DFLT_RXDS_VL_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_RXDS==0) ? \"read-only\": \"read-write\""</p>
22	RSVD_SPI_CTRLR0_22	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
21	SSIC_XIP_CONT_XFER_EN	* Varies	<p>Enable continuous transfer in XIP mode. If this bit is set to 1 then continuous transfer mode in XIP will be enabled, in this mode DWC_ssi will keep target selected until a non-XIP transfer is detected on the AHB interface.</p> <p><b>Value After Reset:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"0x0\": \"SSIC_DFLT_XIP_CONT_XFER_EN\""</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_CONT_XFER_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"read-only\": \"read-write\""</p>
20	XIP_INST_EN	* Varies	<p>XIP instruction enable bit. If this bit is set to 1 then XIP transfers will also have instruction phase. The instruction op-codes will be chosen from XIP_INCR_INST or XIP_WRAP_INST registers bases on AHB transfer type.</p> <p><b>Value After Reset:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"0x0\": \"SSIC_DFLT_XIP_INST_EN\""</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"read-only\": \"read-write\""</p>

Bits	Name	Memory Access	Description
19	XIP_DFS_HC	* Varies	<p>Fix DFS for XIP transfers. If this bit is set to 1 then data frame size for XIP transfers will be fixed to the programmed value in CTRLR0.DFS. The number of data frames to fetch will be determined by HSIZE and HBURST signals. If this bit is set to 0 then data frame size and number of data frames to fetch will be determined by HSIZE and HBURST signals</p> <p><b>Value After Reset:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"0x0\" : \"SSIC_DFLT_DFS_HC\""</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"read-only\" : \"read-write\""</p>
18	SPI_RXDS_EN	* Varies	<p>Read data strobe enable bit. Once this bit is set to 1 DWC_ssi will use Read data strobe (rxds) to capture read data.</p> <p><b>Value After Reset:</b> SSIC_DFLT_RXDS_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_RXDS==0) ? \"read-only\" : \"read-write\""</p>
17	INST_DDR_EN	* Varies	<p>Instruction DDR Enable bit. This will enable Dual-data rate transfer for Instruction phase.</p> <p><b>Value After Reset:</b> SSIC_DFLT_INST_DDR_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DDR==0) ? \"read-only\" : \"read-write\""</p>
16	SPI_DDR_EN	* Varies	<p>SPI DDR Enable bit. This will enable Dual-data rate transfers in Enhanced SPI frame formats of SPI.</p> <p><b>Value After Reset:</b> SSIC_DFLT_DDR_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DDR==0) ? \"read-only\" : \"read-write\""</p>
15:11	WAIT_CYCLES	R/W	<p>Wait cycles in Enhanced SPI mode between control frames transmit and data reception. Specified as number of SPI clock cycles.</p> <p><b>Value After Reset:</b> SSIC_DFLT_WAIT_CYCLES</p> <p><b>Exists:</b> Always</p>
10	RSVD_SPI_CTRLR0_10	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
9:8	INST_L	R/W	<p>Enhanced SPI mode instruction length in bits.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (INST_L0): No Instruction</li> <li>■ 0x1 (INST_L4): 4 bit Instruction length</li> <li>■ 0x2 (INST_L8): 8 bit Instruction length</li> <li>■ 0x3 (INST_L16): 16 bit Instruction length</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_INST_L</p> <p><b>Exists:</b> Always</p>
7	XIP_MD_BIT_EN	* Varies	<p>Mode bits enable in XIP mode. If this bit is set to 1, then in XIP mode of operation DWC_ssi will insert mode bits after the address phase. These bits are set in register XIP_MODE_BITS register.</p> <p><b>Value After Reset:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"0x0\": \"SSIC_DFLT_MD_BITS_EN\""</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_EN==0    SSIC_CONCURRENT_XIP_EN==1) ? \"read-only\": \"read-write\""</p>
6	RSVD_SPI_CTRLR0_6	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
5:2	ADDR_L	R/W	<p>This bit defines Length of Address to be transmitted. Only after this much bits are programmed in to the FIFO the transfer can begin.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (ADDR_L0): No Address</li> <li>■ 0x1 (ADDR_L4): 4 bit Address length</li> <li>■ 0x2 (ADDR_L8): 8 bit Address length</li> <li>■ 0x3 (ADDR_L12): 12 bit Address length</li> <li>■ 0x4 (ADDR_L16): 16 bit Address length</li> <li>■ 0x5 (ADDR_L20): 20 bit Address length</li> <li>■ 0x6 (ADDR_L24): 24 bit Address length</li> <li>■ 0x7 (ADDR_L28): 28 bit Address length</li> <li>■ 0x8 (ADDR_L32): 32 bit Address length</li> <li>■ 0x9 (ADDR_L36): 36 bit Address length</li> <li>■ 0xa (ADDR_L40): 40 bit Address length</li> <li>■ 0xb (ADDR_L44): 44 bit Address length</li> <li>■ 0xc (ADDR_L48): 48 bit Address length</li> <li>■ 0xd (ADDR_L52): 52 bit Address length</li> <li>■ 0xe (ADDR_L56): 56 bit Address length</li> <li>■ 0xf (ADDR_L60): 60 bit Address length</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_ADDR_L</p> <p><b>Exists:</b> Always</p>
1:0	TRANS_TYPE	R/W	<p>Address and instruction transfer format. Selects whether DWC_ssi will transmit instruction/address either in Standard SPI mode or the SPI mode selected in CTRLR0.SPI_FRF field.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (TT0): Instruction and Address will be sent in Standard SPI Mode.</li> <li>■ 0x1 (TT1): Instruction will be sent in Standard SPI Mode and Address will be sent in the mode specified by CTRLR0.SPI_FRF.</li> <li>■ 0x2 (TT2): Both Instruction and Address will be sent in the mode specified by SPI_FRF.</li> <li>■ 0x3 (TT3): Dual Octal mode, the address and instruction are transferred in octal mode and data is transferred on 16 data lines.</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_TRANS_TYPE</p> <p><b>Exists:</b> Always</p>

### 6.1.30 DDR\_DRIVE\_EDGE

- **Name:** DDR\_DRIVE\_EDGE - Transmit Drive Edge Register
- **Description:** This Register is valid only when SSIC\_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 DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0xf8
- **Exists:** SSIC\_HAS\_DDR != 0



**Table 6-34 Fields for Register: DDR\_DRIVE\_EDGE**

Bits	Name	Memory Access	Description
31:8	RSVD_DDR_DRIVE_EDGE	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
7:0	TDE	R/W	TXD Drive edge register which decided the driving edge of transmit data. The maximum value of this register is = (BAUDR/2) -1. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always

### 6.1.31 XIP\_MODE\_BITS

- **Name:** eXecute in Place - Mode bits
- **Description:** This register carries the mode bits which are sent in the XIP mode of operation after address phase. This is a 8 bit register and can only be written when SSIENR register is set to 0.
- **Size:** 32 bits
- **Offset:** 0xfc
- **Exists:** SSIC\_XIP\_EN == 1



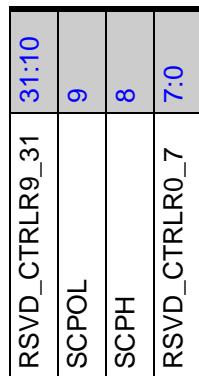
**Table 6-35 Fields for Register: XIP\_MODE\_BITS**

Bits	Name	Memory Access	Description
31:16	RSVD_XIP_MD_BITS	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
15:0	XIP_MD_BITS	R/W	XIP mode bits to be sent after address phase of XIP transfer. <b>Value After Reset:</b> SSIC_DFLT_MD_BITS <b>Exists:</b> Always <b>Volatile:</b> true

## 6.2 DWC\_ssi\_bridge\_block Registers

## 6.2.1 CTRLR0

- **Name:** Control Register 0
- **Description:** This register controls the serial data transfer. It is impossible to write to this register when the DWC\_ssi is enabled.
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always



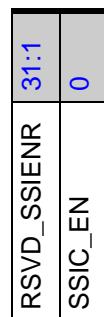
**Table 6-36 Fields for Register: CTRLR0**

Bits	Name	Memory Access	Description
31:10	RSVD_CTRLR9_31	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
9	SCPOL	R/W	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 DWC_ssi controller is not actively transferring data on the serial bus. <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x0 (INACTIVE_HIGH): Inactive state of serial clock is low</li> <li>■ 0x1 (INACTIVE_LOW): Inactive state of serial clock is high</li> </ul> <b>Value After Reset:</b> SSIC_DFLT_SCPOL <b>Exists:</b> Always

Bits	Name	Memory Access	Description
8	SCPH	R/W	<p>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 chip 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 chip select line is activated, and data are captured on the second edge of the serial clock.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (START_BIT): Serial clock toggles at start of first bit</li> <li>■ 0x0 (MIDDLE_BIT): Serial clock toggles in middle of first bit</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_SCPH  <b>Exists:</b> Always</p>
7:0	RSVD_CTRLR0_7	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>

## 6.2.2 SSIENR

- **Name:** SSI Enable Register
- **Description:** This register enables and disables the DWC\_ssi.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always



**Table 6-37 Fields for Register: SSIENR**

Bits	Name	Memory Access	Description
31:1	RSVD_SSIENR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
0	SSIC_EN	R/W	SSI Enable. Enables and disables all DWC_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 DWC_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. <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (ENABLED): Enables DWC_ssi</li> <li>■ 0x0 (DISABLE): Disables DWC_ssi</li> </ul> <b>Value After Reset:</b> SSIC_BOOT_MODE_EN <b>Exists:</b> Always

### 6.2.3 RXFBTR

- **Name:** Receive FIFO Burst Threshold Register
- **Description:** This register is used to control the burst threshold value for AHB interface from RX FIFO
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** Always



**Table 6-38 Fields for Register: RXFBTR**

Bits	Name	Memory Access	Description
31:y	RSVD_RXFBTR	R	<p>Reserved bits - read as zero  <b>Value After Reset:</b> 0x0  <b>Exists:</b> Always  <b>Range Variable[y]:</b> RX_ABW</p>
x:0	RXFBTL	R/W	<p>Receive FIFO Burst Threshold  For Write operations on SPI target interface this register will be used to build a burst on AHB Manager interface. The data will be send in bursts equal to the level specified in this register. If the RXFBTL value is 3 then a burst length of 4 will be used for every write transfer.  If DWC_ssi receives a burst smaller than the programmed value, then DWC_ssi will issue a smaller burst on AHB interface and complete the transaction.  <b>Value After Reset:</b> SSIC_DFLT_RXFBTL  <b>Exists:</b> Always  <b>Range Variable[x]:</b> RX_ABW - 1</p>

### 6.2.4 TXFTLR

- **Name:** Transmit FIFO Threshold Level
- **Description:** This register controls the threshold value for the transmit FIFO memory.
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** Always



**Table 6-39 Fields for Register: TXFTLR**

Bits	Name	Memory Access	Description
31:y	RSVD_TXFTLR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[y]:</b> TX_ABW</p>
x:0	TFT	R/W	<p>Transmit FIFO Threshold.</p> <p>Controls the level of entries (or below) at which the transmit FIFO controller triggers an interrupt. The FIFO depth is configurable in the range 8-256; 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.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[x]:</b> TX_ABW - 1</p>

### 6.2.5 RXFTLR

- **Name:** Receive FIFO Threshold Level
- **Description:** This register controls the threshold value for the receive FIFO memory.
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** Always

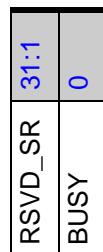


**Table 6-40 Fields for Register: RXFTLR**

Bits	Name	Memory Access	Description
31:y	RSVD_RXFTLR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[y]:</b> RX_ABW</p>
x:0	RFT	R/W	<p>Receive FIFO Threshold.</p> <p>Controls the level of entries (or above) at which the receive FIFO controller triggers an interrupt. The FIFO depth is configurable in the range 8-256. 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.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[x]:</b> RX_ABW - 1</p>

## 6.2.6 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.
- **Size:** 32 bits
- **Offset:** 0x28
- **Exists:** Always

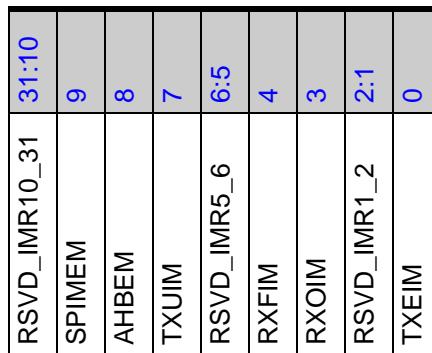


**Table 6-41 Fields for Register: SR**

Bits	Name	Memory Access	Description
31:1	RSVD_SR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	BUSY	R	<p>SSI Busy Flag. When set, indicates that a serial transfer is in progress; when cleared indicates that the DWC_ssi is idle or disabled.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): DWC_ssi is actively transferring data</li> <li>■ 0x0 (INACTIVE): DWC_ssi is idle or disabled</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.2.7 IMR

- **Name:** Interrupt Mask Register
- **Description:** This read/ write register masks or enables all interrupts generated by the DWC\_ssi.
- **Size:** 32 bits
- **Offset:** 0x2c
- **Exists:** Always



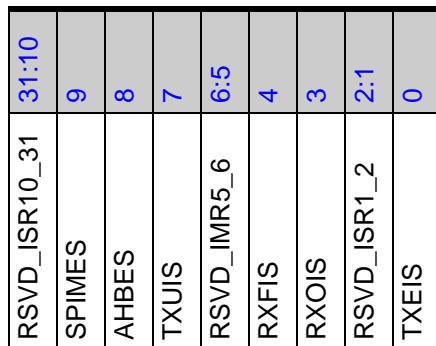
**Table 6-42 Fields for Register: IMR**

Bits	Name	Memory Access	Description
31:10	RSVD_IMR10_31	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
9	SPIMEM	R/W	SPI Manager Error Interrupt Mask <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_spime_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_spime_intr interrupt is masked</li> </ul> <b>Value After Reset:</b> 0x1 <b>Exists:</b> Always
8	AHBEM	R/W	AHB Error Interrupt Mask <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_ahbme_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_ahbme_intr interrupt is masked</li> </ul> <b>Value After Reset:</b> 0x1 <b>Exists:</b> Always
7	TXUIM	R/W	Transmit FIFO Underflow Mask <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_txu_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_txu_intr interrupt is masked</li> </ul> <b>Value After Reset:</b> 0x1 <b>Exists:</b> Always

Bits	Name	Memory Access	Description
6:5	RSVD_IMR5_6	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
4	RXFIM	R/W	<p>Receive FIFO Full Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_rxf_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_rxf_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
3	RXOIM	R/W	<p>Receive FIFO Overflow Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_rxo_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_rxo_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> 0x1</p> <p><b>Exists:</b> Always</p>
2:1	RSVD_IMR1_2	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
0	TXEIM	R/W	<p>Transmit FIFO Empty Interrupt Mask</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (UNMASKED): ssi_txe_intr interrupt is not masked</li> <li>■ 0x0 (MASKED): ssi_txe_intr interrupt is masked</li> </ul> <p><b>Value After Reset:</b> SSIC_IS_DFLT_TXEIM</p> <p><b>Exists:</b> Always</p>

### 6.2.8 ISR

- **Name:** Interrupt Status Register
- **Description:** This register reports the status of the DWC\_ssi interrupts after they have been masked.
- **Size:** 32 bits
- **Offset:** 0x30
- **Exists:** Always



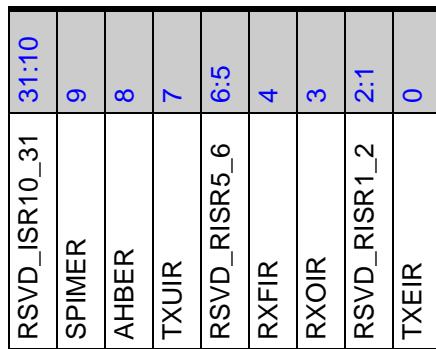
**Table 6-43 Fields for Register: ISR**

Bits	Name	Memory Access	Description
31:10	RSVD_ISR10_31	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
9	SPIMES	R	SPI Manager Error Interrupt Status <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_ahbe_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_ahbe_intr interrupt is not active after masking</li> </ul> <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
8	AHBES	R	AHB Error Interrupt Status <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_spime_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_spime_intr interrupt is not active after masking</li> </ul> <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

Bits	Name	Memory Access	Description
7	TXUIS	R	<p>Transmit FIFO Underflow Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txu_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_txu_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
6:5	RSVD_IMR5_6	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
4	RXFIS	R	<p>Receive FIFO Full Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxf_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_rxf_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
3	RXOIS	R	<p>Receive FIFO Overflow Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxo_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_rxo_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
2:1	RSVD_ISR1_2	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	TXEIS	R	<p>Transmit FIFO Empty Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txe_intr interrupt is active after masking</li> <li>■ 0x0 (INACTIVE): ssi_txe_intr interrupt is not active after masking</li> </ul> <p><b>Value After Reset:</b> "(SSIC_IS_DFLT_TXEIM == 1 &amp;&amp; SSIC_BOOT_MODE_EN == 1)"</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

## 6.2.9 RISR

- **Name:** Raw Interrupt Status Register
- **Description:** Raw Interrupt Status Register
- **Size:** 32 bits
- **Offset:** 0x34
- **Exists:** Always



**Table 6-44 Fields for Register: RISR**

Bits	Name	Memory Access	Description
31:10	RSVD_ISR10_31	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
9	SPIMER	R	<p>SPI Controller Error Raw Interrupt status.</p> <p>This bit will be set to 1 under following conditions:</p> <ul style="list-style-type: none"> <li>- If SPI Target receives a command which does not fall into command set.</li> <li>- If SPI Target receives an out of order read command (for example read data before read request in polling mode).</li> <li>- If SPI Controller tries to write more data than specified in instruction phase.</li> </ul> <p>This bit is cleared when the status register is read.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_ahbe_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_ahbe_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
8	AHBER	R	<p>AHB Error Raw Interrupt Status. In SPI-Bridge configuration this bit indicates whether AHB Manager interface received an error response. This bit is cleared when the status register is read.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_spime_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_spime_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>
7	TXUIR	R	<p>Transmit FIFO Underflow Raw Interrupt status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txu_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_txu_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>
6:5	RSVD_RISR5_6	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>
4	RXFIR	R	<p>Receive FIFO Full Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxf_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_rxf_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>
3	RXOIR	R	<p>Receive FIFO Overflow Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_rxo_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_rxo_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
2:1	RSVD_RISR1_2	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	TXEIR	R	<p>Transmit FIFO Empty Raw Interrupt Status</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x1 (ACTIVE): ssi_txe_intr interrupt is active prior to masking</li> <li>■ 0x0 (INACTIVE): ssi_txe_intr interrupt is not active prior to masking</li> </ul> <p><b>Value After Reset:</b> SSIC_BOOT_MODE_EN</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.2.10 TXUICR

- **Name:** Transmit FIFO Underflow Interrupt Clear Register
- **Description:** Transmit FIFO Underflow Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x38
- **Exists:** Always



**Table 6-45 Fields for Register: TXUICR**

Bits	Name	Memory Access	Description
31:1	RSVD_TXUICR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
0	TXUICR	R	Clear Transmit FIFO Underflow Interrupt This register will reflect the status of the interrupt. A read from this register clears the ssi_txu_intr interrupt. Writing to this register has no effect. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

### 6.2.11 RXOICR

- **Name:** Receive FIFO Overflow Interrupt Clear Register
- **Description:** Receive FIFO Overflow Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x3c
- **Exists:** Always



**Table 6-46 Fields for Register: RXOICR**

Bits	Name	Memory Access	Description
31:1	RSVD_RXOICR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> 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. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

### 6.2.12 SPIMECR

- **Name:** SPI Controller Error interrupt Clear Register
- **Description:** SPI Controller Error interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x40
- **Exists:** Always



**Table 6-47 Fields for Register: SPIMECR**

Bits	Name	Memory Access	Description
31:1	RSVD_SPIMECR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	SPIMECR	R	<p>Clear SPI Controller Error interrupt</p> <p>This register will reflect the status of the interrupt.</p> <p>A read from this register clears the ssi_spime_intr interrupt.</p> <p>Writing to this register has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.2.13 AHBECCR

- **Name:** AHB Error Clear Register
- **Description:** AHB Error Clear Register
- **Size:** 32 bits
- **Offset:** 0x44
- **Exists:** Always

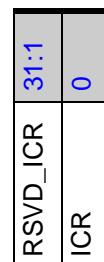


**Table 6-48 Fields for Register: AHBECCR**

Bits	Name	Memory Access	Description
31:1	RSVD_AHBECCR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	AHBECCR	R	<p>Clear AHB Error Interrupt</p> <p>This register will reflect the status of the interrupt.</p> <p>A read from this register clears the ssi_ahbe_intr interrupt.</p> <p>Writing to this register has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.2.14 ICR

- **Name:** Interrupt Clear Register
- **Description:** Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x48
- **Exists:** Always



**Table 6-49 Fields for Register: ICR**

Bits	Name	Memory Access	Description
31:1	RSVD_ICR	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	ICR	R	<p>Clear Interrupts.</p> <p>This register is set if any of the interrupts below are active. A read clears the ssi_spime_intr, ssi_ahbe_intr, ssi_txu_intr and ssi_rxo_intr interrupts. Writing to this register has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.2.15 IDR

- **Name:** Identification Register
- **Description:** This register contains the peripherals identification code, which is written into the register at configuration time using coreConsultant.
- **Size:** 32 bits
- **Offset:** 0x58
- **Exists:** Always



**Table 6-50 Fields for Register: IDR**

Bits	Name	Memory Access	Description
31:0	IDCODE	R	<p>Identification code. The register contains the identification code of the peripheral, which is written into the register at configuration time using CoreConsultant.</p> <p><b>Value After Reset:</b> SSIC_ID</p> <p><b>Exists:</b> Always</p>

## 6.2.16 SSIC\_VERSION\_ID

- **Name:** DWC\_ssi component version
- **Description:** This read-only register stores the specific DWC\_ssi component version.
- **Size:** 32 bits
- **Offset:** 0x5c
- **Exists:** Always



**Table 6-51 Fields for Register: SSIC\_VERSION\_ID**

Bits	Name	Memory Access	Description
31:0	SSIC_COMP_VERSION	R	<p>Contains the hex representation of the Synopsys component version. Consists of ASCII value for each number in the version, followed by *. For example 31_30_33_2A represents the version 1.03*.</p> <p><b>Value After Reset:</b> SSIC_VERSION_ID</p> <p><b>Exists:</b> Always</p>

## 6.3 DWC\_ssi\_address\_block2 Registers

### 6.3.1 XIP\_INCR\_INST

- **Name:** XIP\_INCR\_INST - XIP INCR transfer opcode
- **Description:** This Register is valid only when SSIC\_XIP\_EN is equal to 1. This register is used to store the instruction op-code to be used in INCR transactions when the same is requested on AHB interface. It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0x100
- **Exists:** SSIC\_XIP\_EN == 1



**Table 6-52 Fields for Register: XIP\_INCR\_INST**

Bits	Name	Memory Access	Description
31:16	RSVD_INCR_INST	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
15:0	INCR_INST	R/W	<p>XIP INCR transfer opcode. When SPI_CTRLR0.XIP_INST_EN bit is set to 1, DWC_ssi sends instruction for all XIP transfers, this register field stores the instruction op-code to be sent when an INCR type transfer is requested on AHB bus. The number of bits to be send in instruction phase is determined by SPI_CTRLR0.INST_L field.</p> <p><b>Value After Reset:</b> SSIC_DFLT_INCR_INST</p> <p><b>Exists:</b> Always</p>

### 6.3.2 XIP\_WRAP\_INST

- **Name:** XIP\_WRAP\_INST - XIP WRAP transfer opcode
- **Description:** This Register is valid only when SSIC\_XIP\_EN is equal to 1. This register is used to store the instruction op-code to be used in WRAP transactions when the same is requested on AHB interface. It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0x104
- **Exists:** SSIC\_XIP\_EN == 1



**Table 6-53 Fields for Register: XIP\_WRAP\_INST**

Bits	Name	Memory Access	Description
31:16	RSVD_WRAP_INST	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
15:0	WRAP_INST	R/W	XIP WRAP transfer opcode. When SPI_CTRLR0.XIP_INST_EN bit is set to 1, DWC_ssi sends instruction for all XIP transfers, this register field stores the instruction op-code to be sent when an WRAP type transfer is requested on AHB bus. The number of bits to be send in instruction phase is determined by SPI_CTRLR0.INST_L field. <b>Value After Reset:</b> SSIC_DFLT_WRAP_INST <b>Exists:</b> Always

### 6.3.3 XIP\_CTRL

- **Name:** XIP\_CTRL - XIP Control Register
- **Description:** This Register is valid only when SSIC\_CONCURRENT\_XIP\_EN is equal to 1. This register is used to store the control information that the XIP transfer will be using in the concurrent mode.  
It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0x108
- **Exists:** SSIC\_CONCURRENT\_XIP\_EN == 1

RSVD_XIP_CTRL	31
RXDS_VL_EN	30
XIP_PREFETCH_EN	29
RSVD_XIP_CTRL_28	28
XIP_MBL	27:26
RXDS_SIG_EN	25
XIP_HYPERBUS_EN	24
CONT_XFER_EN	23
INST_EN	22
RXDS_EN	21
INST_DDR_EN	20
DDR_EN	19
DFS_HC	18
WAIT_CYCLES	17:13
MD_BITS_EN	12
RSVD_SPI_CTRLR0_11	11
INST_L	10:9
RSVD_XIP_CTRL_8	8
ADDR_L	7:4
TRANS_TYPE	3:2
FRF	1:0

Table 6-54 Fields for Register: XIP\_CTRL

Bits	Name	Memory Access	Description
31	RSVD_XIP_CTRL	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
30	RXDS_VL_EN	* Varies	RXDS variable latency enable bit. When this bit is set DWC_ssi waits for all the data to be sampled before stopping the SCLK_OUT clock. This enables the support for the devices which support variable latencies within the transfers for RXDS transfers. <b>Value After Reset:</b> SSIC_DFLT_XIP_RXDS_VL_EN <b>Exists:</b> Always <b>Memory Access:</b> "(SSIC_HAS_RXDS==0) ? \"read-only\": \"read-write\""

Bits	Name	Memory Access	Description
29	XIP_PREFETCH_EN	R/W	<p>Enables XIP pre-fetch functionality in DWC_ssi. Once enabled DWC_ssi will pre-fetch data frames from next contiguous location, to reduce the latency for the upcoming contiguous transfer. If the next XIP request is not contiguous then pre-fetched bits will be discarded.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_PREFETCH</p> <p><b>Exists:</b> Always</p>
28	RSVD_XIP_CTRL_28	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
27:26	XIP_MBL	R/W	<p>XIP Mode bits length. Sets the length of mode bits in XIP mode of operation. These bits are valid only when XIP_CTRL.XIP_MD_BIT_EN is set to 1.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (MBL_2): Mode bits length equal to 2</li> <li>■ 0x1 (MBL_4): Mode bits length equal to 4</li> <li>■ 0x2 (MBL_8): Mode bits length equal to 8</li> <li>■ 0x3 (MBL_16): Mode bits length equal to 16</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_MBL</p> <p><b>Exists:</b> Always</p>
25	RXDS_SIG_EN	* Varies	<p>Enable rxds signaling during address and command phase of Hyperbus transfer.</p> <p>This bit enables rxds signaling by Hyperbus target devices during Command-Address (CA) phase. If the rxds signal is set to 1 during the CA phase of transfer, DWC_ssi transmits (2*SPI_CTRLR0.WAIT_CYCLES-1) wait cycles after the address phase is complete.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_RXDS_SIG_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HYPERBUS_EN==0) ? \"read-only\" : \"read-write\""</p>
24	XIP_HYPERBUS_EN	* Varies	<p>SPI Hyperbus Frame format enable for XIP transfers.</p> <p>Selects if data frame format for XIP transfers is in Hyperbus mode. This field is effective only when CTRLR0.FRF is set to SPI frame format.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_HYPERBUS_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HYPERBUS_EN==0) ? \"read-only\" : \"read-write\""</p>

Bits	Name	Memory Access	Description
23	CONT_XFER_EN	* Varies	<p>Enable continuous transfer in XIP mode. If this bit is set to 1 then continuous transfer mode in XIP will be enabled, in this mode DWC_ssi will keep target selected until a non-XIP transfer is detected on the AHB interface.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_CONT_XFER_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_CONT_XFER_EN==0) ? \"read-only\": \"read-write\""</p>
22	INST_EN	* Varies	<p>XIP instruction enable bit. If this bit is set to 1 then XIP transfers will also have instruction phase. The instruction op-codes will be chosen from XIP_INCR_INST or XIP_WRAP_INST registers bases on AHB transfer type.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_INST_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_EN==0) ? \"read-only\": \"read-write\""</p>
21	RXDS_EN	* Varies	<p>Read data strobe enable bit. Once this bit is set to 1 DWC_ssi will use Read data strobe (rxds) to capture read data.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_RXDS_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_RXDS==0) ? \"read-only\": \"read-write\""</p>
20	INST_DDR_EN	* Varies	<p>Instruction DDR Enable bit. This will enable Dual-data rate transfer for Instruction phase.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_INST_DDR_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DDR==0) ? \"read-only\": \"read-write\""</p>
19	DDR_EN	* Varies	<p>SPI DDR Enable bit. This will enable Dual-data rate transfers in Enhanced SPI frame formats of SPI.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_DDR_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DDR==0) ? \"read-only\": \"read-write\""</p>

Bits	Name	Memory Access	Description
18	DFS_HC	* Varies	<p>Fix DFS for XIP transfers. If this bit is set to 1 then data frame size for XIP transfers will be fixed to the programmed value in CTRLR0.DFS. The number of data frames to fetch will be determined by HSIZE and HBURST signals. If this bit is set to 0 then data frame size and number of data frames to fetch will be determined by HSIZE and HBURST signals</p> <p><b>Value After Reset:</b> SSIC_DFLT_DFS_HC</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_XIP_EN==0) ? \"read-only\": \"read-write\""</p>
17:13	WAIT_CYCLES	R/W	<p>Wait cycles in Enhanced SPI mode between control frames transmit and data reception. Specified as number of SPI clock cycles.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_WAIT_CYCLES</p> <p><b>Exists:</b> Always</p>
12	MD_BITS_EN	R/W	<p>Mode bits enable in XIP mode. If this bit is set to 1, then in XIP mode of operation DWC_ssi will insert mode bits after the address phase. These bits are set in register XIP_MODE_BITS register.</p> <p><b>Value After Reset:</b> SSIC_DFLT_MD_BITS_EN</p> <p><b>Exists:</b> Always</p>
11	RSVD_SPI_CTRLR0_11	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
10:9	INST_L	R/W	<p>Enhanced SPI mode instruction length in bits.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (INST_L0): No Instruction</li> <li>■ 0x1 (INST_L4): 4 bit Instruction length</li> <li>■ 0x2 (INST_L8): 8 bit Instruction length</li> <li>■ 0x3 (INST_L16): 16 bit Instruction length</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_INST_L</p> <p><b>Exists:</b> Always</p>
8	RSVD_XIP_CTRL_8	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
7:4	ADDR_L	R/W	<p>This bit defines Length of Address to be transmitted. Only after this much bits are programmed in to the FIFO the transfer can begin.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (ADDR_L0): No Address</li> <li>■ 0x1 (ADDR_L4): 4 bit Address length</li> <li>■ 0x2 (ADDR_L8): 8 bit Address length</li> <li>■ 0x3 (ADDR_L12): 12 bit Address length</li> <li>■ 0x4 (ADDR_L16): 16 bit Address length</li> <li>■ 0x5 (ADDR_L20): 20 bit Address length</li> <li>■ 0x6 (ADDR_L24): 24 bit Address length</li> <li>■ 0x7 (ADDR_L28): 28 bit Address length</li> <li>■ 0x8 (ADDR_L32): 32 bit Address length</li> <li>■ 0x9 (ADDR_L36): 36 bit Address length</li> <li>■ 0xa (ADDR_L40): 40 bit Address length</li> <li>■ 0xb (ADDR_L44): 44 bit Address length</li> <li>■ 0xc (ADDR_L48): 48 bit Address length</li> <li>■ 0xd (ADDR_L52): 52 bit Address length</li> <li>■ 0xe (ADDR_L56): 56 bit Address length</li> <li>■ 0xf (ADDR_L60): 60 bit Address length</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_ADDR_L</p> <p><b>Exists:</b> Always</p>
3:2	TRANS_TYPE	R/W	<p>Address and instruction transfer format. Selects whether DWC_ssi will transmit instruction/address either in Standard SPI mode or the SPI mode selected in CTRLR0.SPI_FRF field.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (TT0): Instruction and Address will be sent in Standard SPI Mode.</li> <li>■ 0x1 (TT1): Instruction will be sent in Standard SPI Mode and Address will be sent in the mode specified by XIP_CTRL.SPI_FRF.</li> <li>■ 0x2 (TT2): Both Instruction and Address will be sent in the mode specified by XIP_CTRL.SPI_FRF.</li> <li>■ 0x3 (TT3): Dual Octal mode, the address and instruction are transferred in octal mode and data is transferred on 16 data lines.</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_TRANS_TYPE</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
1:0	FRF	R/W	<p>SPI Frame Format Selects data frame format for Transmitting/Receiving the data.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (SPI_STANDARD): Standard SPI Format</li> <li>■ 0x1 (SPI_DUAL): Dual SPI Format</li> <li>■ 0x2 (SPI_QUAD): Quad SPI Format</li> <li>■ 0x3 (SPI_OCTAL): Octal SPI Format</li> </ul> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_SPI_FRF <b>Exists:</b> Always</p>

### 6.3.4 XIP\_SER

- **Name:** Target Enable Register
- **Description:** This register is valid only when the SSIC\_CONCURRENT\_XIP\_EN is equal to 1 and SSIC\_XIP\_WRITE\_EN is equal to 0. The register enables the individual chip select output lines from the DWC\_ssi controller for XIP mode of operation. Up to 16 chip-select output pins are available on the DWC\_ssi controller. You cannot write to this register when DWC\_ssi is busy or when SSIC\_EN = 1.
- **Size:** 32 bits
- **Offset:** 0x10c
- **Exists:** SSIC\_CONCURRENT\_XIP\_EN == 1 && SSIC\_XIP\_EXT\_SSEL\_EN == 0

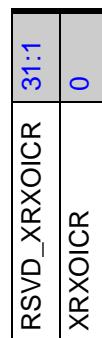


**Table 6-55 Fields for Register: XIP\_SER**

Bits	Name	Memory Access	Description
31:y	RSVD_SER	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[y]:</b> SSIC_NUM_SLAVES</p>
x:0	SER	R/W	<p>Chip Select Enable Flag.</p> <p>Each bit in this register corresponds to a chip select line (ss_x_n) from the DWC_ssi controller. When a bit in this register is set (1), the corresponding chip select line from the controller is activated when a XIP transfer begins. It should be noted that setting or clearing bits in this register have no effect on the corresponding chip select outputs until a XIP transfer is started. Before beginning a transfer, you should enable the bit in this register that corresponds to the target device with which the controller wants to communicate. When not operating in broadcast mode, only one bit in this field should be set.</p> <p><b>Value After Reset:</b> SSIC_DFLT_XIP_SLV</p> <p><b>Exists:</b> Always</p> <p><b>Range Variable[x]:</b> SSIC_NUM_SLAVES - 1</p>

### 6.3.5 XRxoICR

- **Name:** XIP Receive FIFO Overflow Interrupt Clear Register
- **Description:** XIP Receive FIFO Overflow Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x110
- **Exists:** SSIC\_CONCURRENT\_XIP\_EN == 1

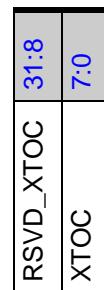


**Table 6-56 Fields for Register: XRxoICR**

Bits	Name	Memory Access	Description
31:1	RSVD_XRxoICR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
0	XRxoICR	R	Clear XIP Receive FIFO Overflow Interrupt. This register reflects the status of the interrupt. A read from this register clears the <i>ssi_xrxo_intr(_n)</i> interrupt; writing has no effect. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

### 6.3.6 XIP\_CNT\_TIME\_OUT

- **Name:** XIP time out register for continuous transfers
- **Description:** XIP count down register for continuous mode. The counter is used to de-select the target during continuous transfer mode. It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0x114
- **Exists:** SSIC\_CONT\_XFER\_TIMEOUT\_CNT\_EN == 1



**Table 6-57 Fields for Register: XIP\_CNT\_TIME\_OUT**

Bits	Name	Memory Access	Description
31:8	RSVD_XTOC	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
7:0	XTOC	R/W	XIP time out value in terms of hclk. Once target is selected in continuous XIP mode this counter will be used to de-select the target if there is no request for the time specified in the counter. <b>Value After Reset:</b> SSIC_DFLT_XIP_TOCNT <b>Exists:</b> Always <b>Volatile:</b> true

### 6.3.7 SPI\_CTRLR1

- **Name:** SPI Control 1 register
- **Description:** SPI Control register 1 is used to control serial transfers in SPI mode of operation. It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0x118
- **Exists:** SSIC\_SPI\_MODE != 0

RSVD_SPI_CTRLR1_20_31	31:20
CS_MIN_HIGH	19:16
RSVD_SPI_CTRLR1_12_15	15:12
MAX_WS	11:8
RSVD_SPI_CTRLR1_3_7	7:3
DYN_WS	2:0

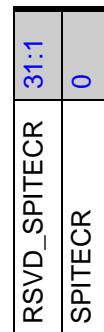
Table 6-58 Fields for Register: SPI\_CTRLR1

Bits	Name	Memory Access	Description
31:20	RSVD_SPI_CTRLR1_20_31	R	Reserved bits - Read Only <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
19:16	CS_MIN_HIGH	R/W	Chip-Select Minimum HIGH period. This field is used to set the minimum time period (in terms of ssi_clk) between two back to back SPI operations. <b>Value After Reset:</b> SSIC_DFLT_CS_MIN_HIGH <b>Exists:</b> Always <b>Volatile:</b> true
15:12	RSVD_SPI_CTRLR1_12_15	R	Reserved bits - Read Only <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

Bits	Name	Memory Access	Description
11:8	MAX_WS	R/W	<p>Maximum wait cycles allowed per transaction. This field indicate, up to how many times SPI target could insert the wait states. The internal counter is incremented every time when DWC_ssi controller checks for the status from the target and receives wait response.</p> <p><b>Value After Reset:</b> SSIC_DFLT_MAX_WS</p> <p><b>Exists:</b> Always <b>Volatile:</b> true</p>
7:3	RSVD_SPI_CTRLR1_3_7	R	<p>Reserved bits - Read Only</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always <b>Volatile:</b> true</p>
2:0	DYN_WS	R/W	<p>SPI Dynamic Wait states field. This field is used to set the value for wait states which will be introduced when SPI target sends BUSY status to the Controller. The programmed value of wait States will be introduced before checking status again. Number of wait states = DYN_WS+1</p> <p><b>Value After Reset:</b> SSIC_DFLT_DYN_WS</p> <p><b>Exists:</b> Always <b>Volatile:</b> true</p>

### 6.3.8 SPITECR

- **Name:** SPI Transmit Error Interrupt Clear Register
- **Description:** SPI Transmit Error Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x11c
- **Exists:** SSIC\_SPI\_DYN\_WS\_EN == 1

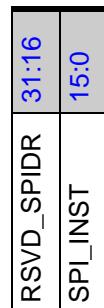


**Table 6-59 Fields for Register: SPITECR**

Bits	Name	Memory Access	Description
31:1	RSVD_SPITECR	R	<p>Reserved bits - Read Only</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	SPITECR	R	<p>Clear SPI Transmit Error interrupt.</p> <p>This register will reflect the status of the interrupt.</p> <p>A read from this register clears the ssi_spitie_intr interrupt.</p> <p>Writing to this register has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.3.9 SPIDR

- **Name:** SPI Device Register
- **Description:** SPI Device Register
- **Size:** 32 bits
- **Offset:** 0x120
- **Exists:** SSIC\_HAS\_DMA == 2



**Table 6-60 Fields for Register: SPIDR**

Bits	Name	Memory Access	Description
31:16	RSVD_SPIDR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
15:0	SPI_INST	R/W	SPI Instruction code. This instruction code will be used for SPI operation <b>Value After Reset:</b> SSIC_DFLT_SPI_INST <b>Exists:</b> Always

### 6.3.10 SPIAR

- **Name:** SPI Device Address Register
- **Description:** SPI Device Address Register
- **Size:** 32 bits
- **Offset:** 0x124
- **Exists:** SSIC\_HAS\_DMA == 2



**Table 6-61 Fields for Register: SPIAR**

Bits	Name	Memory Access	Description
31:0	SDAR	R/W	<p>SPI Device Address Register. This address will be used during read/write on SPI interface for DMA transfer</p> <p><b>Value After Reset:</b> SSIC_DFLT_SPIAR</p> <p><b>Exists:</b> Always</p>

### 6.3.11 AXIAR0

- **Name:** AXI Address Register 0
- **Description:** AXI Address Register 0
- **Size:** 32 bits
- **Offset:** 0x128
- **Exists:** SSIC\_HAS\_DMA == 2



**Table 6-62 Fields for Register: AXIAR0**

Bits	Name	Memory Access	Description
31:0	AXIAR_0_31	R/W	LSB for AXI address for DMA operation. <b>Value After Reset:</b> SSIC_DFLT_AXIAR0 <b>Exists:</b> Always

### 6.3.12 AXIAR1

- **Name:** AXI Address Register 1
- **Description:** AXI Address Register 1
- **Size:** 32 bits
- **Offset:** 0x12c
- **Exists:** SSIC\_HAS\_DMA == 2 && SSIC\_AXI\_AW > 32



**Table 6-63 Fields for Register: AXIAR1**

Bits	Name	Memory Access	Description
31:y	RSVD_AXIAR1	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Range Variable[y]:</b> SSIC_AXI_AW - 32
x:0	AXIAR_32_63	R/W	MSB for source address for DMA operation. <b>Value After Reset:</b> SSIC_DFLT_AXIAR1 <b>Exists:</b> Always <b>Range Variable[x]:</b> SSIC_AXI_AW - 33

### 6.3.13 AXIECR

- **Name:** AXI Manager Error Interrupt Clear Register
- **Description:** AXI Manager Error Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x130
- **Exists:** SSIC\_HAS\_DMA == 2



**Table 6-64 Fields for Register: AXIECR**

Bits	Name	Memory Access	Description
31:1	RSVD_AXIECR	R	Reserved bit <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
0	AXIECR	R	Clear AXI Error Interrupt. This register will reflect the status of the interrupt. A read from this register clears the ssi_axie_intr interrupt. Writing to this register has no effect. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true

### 6.3.14 DONECR

- **Name:** Transfer Done Clear Interrupt Clear Register
- **Description:** Transfer Done Clear Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x134
- **Exists:** SSIC\_HAS\_DMA == 2

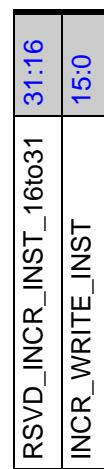


**Table 6-65 Fields for Register: DONECR**

Bits	Name	Memory Access	Description
31:1	RSVD_DONECR	R	<p>Reserved bit</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
0	DONECR	R	<p>Clear Transfer Done Interrupt. This register will reflect the status of the interrupt. A read from this register clears the ssi_done_intr interrupt. Writing to this register has no effect.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.3.15 XIP\_WRITE\_INCR\_INST

- **Name:** XIP\_WRITE\_INCR\_INST - XIP Write INCR transfer opcode
- **Description:** This Register is valid only when SSIC\_XIP\_WRITE\_REG\_EN is set to 1. This register is used to store the instruction op-code to be used in INCR transactions for XIP Write when the same is requested on AHB interface. It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0x140
- **Exists:** SSIC\_XIP\_WRITE\_REG\_EN == 1



**Table 6-66 Fields for Register: XIP\_WRITE\_INCR\_INST**

Bits	Name	Memory Access	Description
31:16	RSVD_INCR_INST_16to31	R	Reserved bits - Read Only <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
15:0	INCR_WRITE_INST	R/W	XIP Write INCR transfer opcode. When XIP_WRITE_CTRL.INST_L is not equal to 0, DWC_ssi sends instruction for all XIP write transfers, this register field stores the instruction op-code to be sent when an INCR type XIP Write transfer is requested on AHB bus. The number of bits to be send in instruction phase is determined by XIP_WRITE_CTRL.INST_L field. <b>Value After Reset:</b> SSIC_XIPWR_DFLT_INCR_INST <b>Exists:</b> Always

### 6.3.16 XIP\_WRITE\_WRAP\_INST

- **Name:** XIP\_WRITE\_WRAP\_INST - XIP Write WRAP transfer opcode
- **Description:** This Register is valid only when SSIC\_XIP\_WRITE\_REG\_EN is set to 1. This register is used to store the instruction op-code to be used in WRAP transactions for XIP Write when the same is requested on AHB interface. It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0x144
- **Exists:** SSIC\_XIP\_WRITE\_REG\_EN == 1



**Table 6-67 Fields for Register: XIP\_WRITE\_WRAP\_INST**

Bits	Name	Memory Access	Description
31:16	RSVD_WRAP_INST_16to31	R	Reserved bits - Read Only <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
15:0	WRAP_WRITE_INST	R/W	XIP Write WRAP transfer opcode. When XIP_WRITE_CTRL.INST_L is not equal to 0, DWC_ssi sends instruction for all XIP write transfers, this register field stores the instruction op-code to be sent when an WRAP type XIP Write transfer is requested on AHB bus. The number of bits to be send in instruction phase is determined by XIP_WRITE_CTRL.INST_L field. <b>Value After Reset:</b> SSIC_XIPWR_DFLT_WRAP_INST <b>Exists:</b> Always

### 6.3.17 XIP\_WRITE\_CTRL

- **Name:** XIP\_WRITE\_CTRL - XIP Write Control Register
- **Description:** This Register is valid only when SSIC\_XIP\_WRITE\_REG\_EN is equal to 1. This register is used to store the control information that the XIP write transfer will be using in the XIP mode. It is not possible to write to this register when the DWC\_ssi is enabled (SSIC\_EN=1).
- **Size:** 32 bits
- **Offset:** 0x148
- **Exists:** SSIC\_XIP\_WRITE\_REG\_EN == 1

RSVD_XIP_WRITECTRL_22to31	31:22	
XIPWR_DFS_HC	21	
XIPWR_WAIT_CYCLES	20:16	
RSVD_XIP_WRITECTRL_15	15	
XIPWR_DM_EN	14	
XIPWR_RXDS_SIG_EN	13	
XIPWR_HYPERBUS_EN	12	
WR_INST_DDR_EN	11	
WR_SPI_DDR_EN	10	
WR_INST_L	9:8	
WR_ADDR_L	7:4	
WR_TRANS_TYPE	3:2	
WR_FRF	1:0	

**Table 6-68 Fields for Register: XIP\_WRITE\_CTRL**

Bits	Name	Memory Access	Description
31:22	RSVD_XIP_WRITECTRL_22to31	R	Reserved bits - Read Only <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
21	XIPWR_DFS_HC	R/W	Fix DFS for XIP transfers. If this bit is set to 1 then data frame size for XIP transfers will be fixed to the programmed value in CTRLR0.DFS. If this bit is set to 0 then data frame size will be determined by HSIZE signal. <b>Value After Reset:</b> SSIC_XIPWR_DFLT_DFS_HC <b>Exists:</b> Always
20:16	XIPWR_WAIT_CYCLES	R/W	Wait cycles in Enhanced SPI mode between control frames transmit and data reception. Specified as number of SPI clock cycles. <b>Value After Reset:</b> SSIC_XIPWR_DFLT_XIP_WAIT_CYCLES <b>Exists:</b> Always

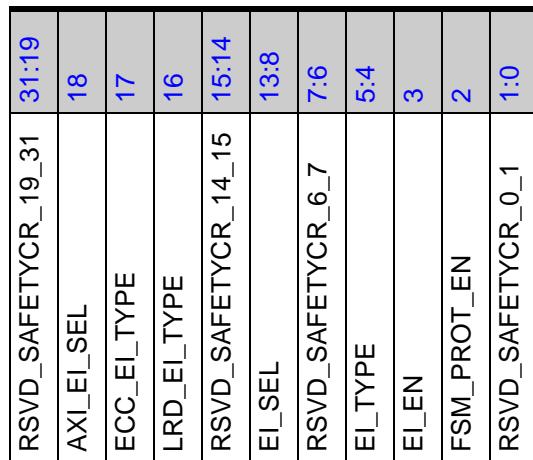
Bits	Name	Memory Access	Description
15	RSVD_XIP_WRITECTRL_15	R	<p>Reserved bits - Read Only</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p>
14	XIPWR_DM_EN	* Varies	<p>SPI data mask enable bit.</p> <p>When this bit is enabled, the txd_dm signal is used to mask the data on the txd data line.</p> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_DM_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_SPI_DM_EN==0) ? \"read-only\": \"read-write\""</p>
13	XIPWR_RXDS_SIG_EN	* Varies	<p>Enable rxds signaling during address and command phase of Hyperbus transfer.</p> <p>This bit enables rxds signaling by Hyperbus target devices during Command-Address (CA) phase. If the rxds signal is set to 1 during the CA phase of transfer, DWC_ssi transmits (2*XIP_WRITE_CTRL.WAIT_CYCLES-1) wait cycles after the address phase is complete.</p> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_RXDS_SIG_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HYPERBUS_EN==0) ? \"read-only\": \"read-write\""</p>
12	XIPWR_HYPERBUS_EN	* Varies	<p>SPI Hyperbus Frame format enable for XIP Write transfers. Selects if data frame format for XIP Write transfers is in Hyperbus mode. This field is effective only when CTRLR0.FRF is set to SPI frame format.</p> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_HYPERBUS_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HYPERBUS_EN==0) ? \"read-only\": \"read-write\""</p>
11	WR_INST_DDR_EN	* Varies	<p>Instruction DDR Enable bit. This will enable Dual-data rate transfer for Instruction phase.</p> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_INST_DDR_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DDR==0) ? \"read-only\": \"read-write\""</p>
10	WR_SPI_DDR_EN	* Varies	<p>SPI DDR Enable bit. This will enable Dual-data rate transfers in Enhanced SPI frame formats of SPI.</p> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_DDR_EN</p> <p><b>Exists:</b> Always</p> <p><b>Memory Access:</b> "(SSIC_HAS_DDR==0) ? \"read-only\": \"read-write\""</p>

Bits	Name	Memory Access	Description
9:8	WR_INST_L	R/W	<p>Enhanced SPI mode instruction length in bits.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (INST_L0): No Instruction</li> <li>■ 0x1 (INST_L4): 4 bit Instruction length</li> <li>■ 0x2 (INST_L8): 8 bit Instruction length</li> <li>■ 0x3 (INST_L16): 16 bit Instruction length</li> </ul> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_INST_L</p> <p><b>Exists:</b> Always</p>
7:4	WR_ADDR_L	R/W	<p>This bit defines Length of Address to be transmitted. Only after this much bits are programmed in to the FIFO the transfer can begin.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (ADDR_L0): Reserved</li> <li>■ 0x1 (ADDR_L4): 4 bit Address length</li> <li>■ 0x2 (ADDR_L8): 8 bit Address length</li> <li>■ 0x3 (ADDR_L12): 12 bit Address length</li> <li>■ 0x4 (ADDR_L16): 16 bit Address length</li> <li>■ 0x5 (ADDR_L20): 20 bit Address length</li> <li>■ 0x6 (ADDR_L24): 24 bit Address length</li> <li>■ 0x7 (ADDR_L28): 28 bit Address length</li> <li>■ 0x8 (ADDR_L32): 32 bit Address length</li> </ul> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_ADDR_L</p> <p><b>Exists:</b> Always</p>
3:2	WR_TRANS_TYPE	R/W	<p>Address and instruction transfer format.</p> <p>Selects whether DWC_ssi will transmit instruction/address either in Standard SPI mode or the SPI mode selected in XIP_WRITE_CTRL.FRF field.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (TT0): Instruction and Address will be sent in Standard SPI Mode.</li> <li>■ 0x1 (TT1): Instruction will be sent in Standard SPI Mode and Address will be sent in the mode specified by XIP_WRITE_CTRL.FRF.</li> <li>■ 0x2 (TT2): Both Instruction and Address will be sent in the mode specified by XIP_WRITE_CTRL.FRF.</li> <li>■ 0x3 (TT3): Dual Octal mode, the address and instruction are transferred in octal mode and data is transferred on 16 data lines.</li> </ul> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_TRANS_TYPE</p> <p><b>Exists:</b> Always</p>

Bits	Name	Memory Access	Description
1:0	WR_FRF	R/W	<p>SPI Frame Format Selects data frame format for Transmitting the data.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (SPI_STANDARD): Standard SPI Format</li> <li>■ 0x1 (SPI_DUAL): Dual SPI Format</li> <li>■ 0x2 (SPI_QUAD): Quad SPI Format</li> <li>■ 0x3 (SPI_OCTAL): Octal SPI Format</li> </ul> <p><b>Value After Reset:</b> SSIC_XIPWR_DFLT_FRF <b>Exists:</b> Always</p>

### 6.3.18 SAFETYCR

- **Name:** Safety Control register
- **Description:** Safety Control Register
- **Size:** 32 bits
- **Offset:** 0x180
- **Exists:** SSIC\_SAFETY\_PKG\_EN > 0



**Table 6-69 Fields for Register: SAFETYCR**

Bits	Name	Memory Access	Description
31:19	RSVD_SAFETYCR_19_31	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
18	AXI_EI_SEL	* Varies	<p>Error injection selection for AXI data path. Selects whether the error is inserted for the lower bits or upper bits of AXI data</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (LOW): Insert Error in lower 32 bits</li> <li>■ 0x1 (HIGH): Insert Error in upper 32 bits</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p> <p><b>Memory Access:</b> "(SSIC_HAS_DMA==2) ? \"read-write\" : \"read-only\""</p>

Bits	Name	Memory Access	Description
17	ECC_EI_TYPE	* Varies	<p>ECC Error Insertion Type.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (CE): 1 bit Correctable Error</li> <li>■ 0x1 (UE): 2 bit Uncorrectable Error</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p> <p><b>Memory Access:</b> "(SSIC_SAFETY_PKG_EN==2) ? \"read-write\" : \"read-only\""</p>
16	LRD_EI_TYPE	R/W	<p>Logic/Register Duplication Error type.</p> <p>This bit selects whether the error should be inserted in the original or the duplicated instance of the protected logic.</p> <p>For the Register duplication it is used to select whether the error should be inserted in the duplicated or the original register.</p> <p>For the logic duplication this bit is used to select if the error should be inserted in the original or the duplicated instance.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (ORG): Error is inserted in the original logic</li> <li>■ 0x1 (DUP): Error is inserted in the duplicate logic</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
15:14	RSVD_SAFETYCR_14_15	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
13:8	EI_SEL	R/W	<p>Error Injection Selection.</p> <p>Used for error injection selection. For Data Path error injection, this field is used to select in which bit the error should be inserted.</p> <p>For the Register Duplication Error injection, this bit is used to select in which register the error should be inserted.</p> <p>For FSM Error injection, this bit is used to select in which FSM the error should be inserted.</p> <p>For Logic Duplication error insertion, this bit is used to select in which protected logic the error should be inserted.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
7:6	RSVD_SAFETYCR_6_7	R	<p>Reserved bits - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
5:4	EI_TYPE	R/W	<p>Error Insertion Type.</p> <p>Select the error injection type in which safety mechanism the error should be inserted</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (DP_EI): Insert Error in data path protection</li> <li>■ 0x1 (REGP_EI): Insert Error in Register Duplication Mechanism</li> <li>■ 0x2 (TO_EI): Insert Error in FSM time-out protection</li> <li>■ 0x2 (LD_EI): Insert Error in Logic Duplication mechanism</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
3	EI_EN	R/W	<p>Error Insertion Enable.</p> <p>Insert Error in Safety Mechanism chosen using EI_TYPE field. This bit will be cleared by hardware once the error interrupt is generated.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (ENABLE): Enable Error Insertion</li> <li>■ 0x1 (DISABLE): Disable Error Insertion</li> </ul> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
2	FSM_PROT_EN	* Varies	<p>FSM One-hot encoding check Enable.</p> <p>This bit enables one-hot encoding checker for all FSM state variables in DWC_ssi.</p> <p><b>Values:</b></p> <ul style="list-style-type: none"> <li>■ 0x0 (DISABLE): Disable FSM One-hot encoding Protection</li> <li>■ 0x1 (ENABLE): Enable FSM One-hot encoding Protection</li> </ul> <p><b>Value After Reset:</b> SNPS_RSVDPARAM_7</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p> <p><b>Memory Access:</b> "(SSIC_SAFETY_PKG_EN==1) ? \"read-write\" : \"read-only\""</p>
1:0	RSVD_SAFETYCR_0_1	R	<p>Reserved bit - read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

### 6.3.19 SLVIFFSMTOCR

- **Name:** Subordinate interface Clock domain FSM time out register
- **Description:** Subordinate interface clock domain FSM time out register. This register contains control bits to enable automotive safety features in DWC\_ssi.
- **Size:** 32 bits
- **Offset:** 0x184
- **Exists:** SSIC\_SAFETY\_PKG\_EN > 0 && SSIC\_SLVIF\_TYPE==1

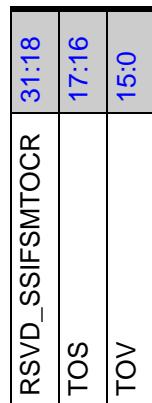


**Table 6-70 Fields for Register: SLVIFFSMTOCR**

Bits	Name	Memory Access	Description
31:16	RSVD_SLVIFFSMTOCR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
15:0	TOV	R/W	Time Out Value. This bit sets the time out value for AHB subordinate clock domain FSM. If FSM state does not reach an IDLE state in given amount of AHB clock cycles, the corresponding error interrupt is asserted. <b>Value After Reset:</b> SNPS_RSVDPARAM_4 <b>Exists:</b> Always

### 6.3.20 SSIFSMTOCR

- **Name:** SSI Clock domain FSM time out register
- **Description:** DWC\_ssi core clock domain FSM time out register. This register contains control bits to enable automotive safety features in DWC\_ssi.
- **Size:** 32 bits
- **Offset:** 0x188
- **Exists:** SSIC\_SAFETY\_PKG\_EN > 0



**Table 6-71 Fields for Register: SSIFSMTOCR**

Bits	Name	Memory Access	Description
31:18	RSVD_SSIFSMTOCR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
17:16	TOS	R/W	Time out scale. This bit is used to scale the time-out value for SSI clock domain. BAUDR = Baud Rate for Controller and SCLK_IN Sampling edge for Target mode <b>Values:</b> <ul style="list-style-type: none"> <li>■ 0x0 (DFLT): Each count tick happens after BAUDR*DFS ssi_clk cycles</li> <li>■ 0x1 (MUL2): Each count tick happens after 2*BAUDR*DFS ssi_clk cycles</li> <li>■ 0x2 (MUL4): Each count tick happens after 4*BAUDR*DFS ssi_clk cycles</li> <li>■ 0x3 (MUL8): Each count tick happens after 8*BAUDR*DFS ssi_clk cycles</li> </ul> <b>Value After Reset:</b> SNPS_RSVDPARAM_15 <b>Exists:</b> Always

Bits	Name	Memory Access	Description
15:0	TOV	R/W	<p>Time Out Value. This bit sets the time out value for DWC_ssi core clock domain FSMs. If FSM state does not reach an IDLE state in the given time period, the corresponding error interrupt is asserted.</p> <p><b>Value After Reset:</b> SNPS_RSVDPARAM_9</p> <p><b>Exists:</b> Always</p>

### 6.3.21 AXIFSMTOCR

- **Name:** AXI Clock domain FSM time out register
- **Description:** AXI clock domain FSM time out register. This register contains control bits to enable automotive safety features in DWC\_ssi.
- **Size:** 32 bits
- **Offset:** 0x18c
- **Exists:** SSIC\_SAFETY\_PKG\_EN > 0 && SSIC\_HAS\_DMA==2



**Table 6-72 Fields for Register: AXIFSMTOCR**

Bits	Name	Memory Access	Description
31:16	RSVD_AXIFSMTOCR	R	Reserved bits - read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
15:0	TOV	R/W	Time Out Value. This bit sets the time out value for AXI clock domain FSMs. If FSM state does not reach an IDLE state in given amount of AXI clock cycles, the corresponding error interrupt is asserted. <b>Value After Reset:</b> SNPS_RSVDPARAM_8 <b>Exists:</b> Always

### 6.3.22 SAFETYISR

- **Name:** Safety Interrupt Status Register
- **Description:** Safety Interrupt Status Register
- **Size:** 32 bits
- **Offset:** 0x1a0
- **Exists:** SSIC\_SAFETY\_PKG\_EN > 0

RSVD_25_31_SAFETYISR	31:25
TMRES	24
RSVD_22_23_SAFETYISR	23:22
LDFIFOES	21
LDDMAFSMES	20
LDXIPES	19
LDSHIFTES	18
LDCKES	17
LDSSIFS MES	16
LDLSVIFES	15
DPUES	14
DPCES	13
AXIUES	12
AXICES	11
AXIVRPES	10
SLVIFAPES	9
SLVIFDPES	8
REGERRS	7
DPPEs	6
AXIPES	5
RSVD_4_SAFETYISR	4
SSIIFSMTOS	3
SLVIFSMTOS	2
AXIFSMTOS	1
FSPMPEs	0

Table 6-73 Fields for Register: SAFETYISR

Bits	Name	Memory Access	Description
31:25	RSVD_25_31_SAFETYISR	R	RSVD25_31_SAFETYISR Reserved bits - Read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
24	TMRES	R	TMR error status register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2 <b>Volatile:</b> true
23:22	RSVD_22_23_SAFETYISR	R	RSVD22_23_SAFETYISR Reserved bits - Read as zero <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true
21	LDFIFOES	R	FIFO Packing/Unpacking Logic Duplication error status register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2 && SSIC_HAS_DMA==2 <b>Volatile:</b> true

Bits	Name	Memory Access	Description
20	LDDMAFSMES	R	<p>AXI DMA FSM Logic Duplication error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SAFETY_PKG_EN == 2 &amp;&amp; SSIC_HAS_DMA==2</p> <p><b>Volatile:</b> true</p>
19	LDXIPES	R	<p>XIP Read/Write Logic Duplication error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SAFETY_PKG_EN == 2 &amp;&amp; SSIC_XIP_EN==1</p> <p><b>Volatile:</b> true</p>
18	LDSHIFTES	R	<p>Shift Register Logic Duplication error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SAFETY_PKG_EN == 2</p> <p><b>Volatile:</b> true</p>
17	LDCKES	R	<p>Clock Generator Logic Duplication error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SAFETY_PKG_EN == 2</p> <p><b>Volatile:</b> true</p>
16	LDSSIFSMES	R	<p>SSI Controller/Target FSM Logic Duplication error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SAFETY_PKG_EN == 2</p> <p><b>Volatile:</b> true</p>
15	LDSLVIFES	R	<p>AHB Target FSM Logic Duplocation error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SAFETY_PKG_EN == 2</p> <p><b>Volatile:</b> true</p>
14	DPUES	R	<p>Data Path ECC uncorrectable error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SAFETY_PKG_EN == 2</p> <p><b>Volatile:</b> true</p>
13	DPCES	R	<p>Data Path ECC correctable error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SAFETY_PKG_EN == 2</p> <p><b>Volatile:</b> true</p>
12	AXIUES	R	<p>AXI Manager Interface ECC uncorrectable error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_AXI_SAFETY_EN == 2</p> <p><b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
11	AXICES	R	<p>AXI Manager Interface ECC correctable error status register.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_AXI_SAFETY_EN == 2</p> <p><b>Volatile:</b> true</p>
10	AXIVRPES	R	<p>AXI Valid/Ready Parity Error status register.</p> <p>This bit is set when AXI Interface Valid/Ready parity failure is detected.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_READY_VALID_PAR_EN == 1</p> <p><b>Volatile:</b> true</p>
9	SLVIFAPES	R	<p>AHB address parity failure interrupt status</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> SSIC_SLVIF_PAR_EN &gt; 1</p> <p><b>Volatile:</b> true</p>
8	SLVIFDPES	R	<p>AHB write data parity failure interrupt status</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
7	REGERRS	R	<p>Register space error status. This bit will be asserted when error is detected in register space protection logic.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
6	DPPES	R	<p>Data path Parity checker error.</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
5	AXIPES	R	<p>AXI parity failure interrupt status</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>
4	RSVD_4_SAFETYISR	R	<p>Reserved bit - Read as zero</p> <p><b>Value After Reset:</b> 0x0</p> <p><b>Exists:</b> Always</p> <p><b>Volatile:</b> true</p>

Bits	Name	Memory Access	Description
3	SSIFSMTOS	R	<p>SSI Core clock domain FSM time out error status. This register bit is set when SSI Core clock domain FSM time out occurs.</p> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>
2	SLVIFSMTOS	R	<p>AHB Subordinate Interface clock domain FSM time out error status. This register bit is set when AHB Subordinate Interface clock domain FSM time out occurs.</p> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>
1	AXIFSMTOS	R	<p>AXI clock domain FSM time out error status. This register bit is set when AXI clock domain FSM time out occurs.</p> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>
0	FSMPES	R	<p>FSM One-hot Error status. This register bit is set when FSM one-hot encoding failure occurs.</p> <p><b>Value After Reset:</b> 0x0 <b>Exists:</b> Always <b>Volatile:</b> true</p>

### 6.3.23 SAFETYICR

- **Name:** Safety Interrupt Clear Register
- **Description:** Safety Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x1a4
- **Exists:** SSIC\_SAFETY\_PKG\_EN > 0

RSVD_25_31_SAFETYICR	31:25
TMREC	24
RSVD_22_23_SAFETYICR	23:22
LDFIFOEC	21
LDDMAFSMEC	20
LDXIPEC	19
LDSHIFTEC	18
LDCKEC	17
LDSSIFSMEC	16
LDSLVIFEC	15
DPUEC	14
DPCEC	13
AXIUEC	12
AXICEC	11
AXIVRPEC	10
SLVIFAPEC	9
SLVIFDPEC	8
REGERRC	7
DPPEC	6
AXIPEC	5
RSVD_4_SAFETYICR	4
SSIFSMTOC	3
SLVIFSMTOC	2
AXIFSMTOC	1
FSMPEC	0

Table 6-74 Fields for Register: SAFETYICR

Bits	Name	Memory Access	Description
31:25	RSVD_25_31_SAFETYICR	W	RSVD_25_31_SAFETYICR Reserved bits <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
24	TMREC	W	TMR error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2
23:22	RSVD_22_23_SAFETYICR	W	RSVD_22_23_SAFETYICR Reserved bits <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
21	LDFIFOEC	W	FIFO Packing/Unpacking Logic Duplication error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2 && SSIC_HAS_DMA==2
20	LDDMAFSMEC	W	AXI DMA FSM Logic Duplication error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2 && SSIC_HAS_DMA==2

Bits	Name	Memory Access	Description
19	LDXIPEC	W	XIP Read/Write Logic Duplication error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2 && SSIC_XIP_EN==1
18	LDSHIFTEC	W	Shift Register Logic Duplication error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2
17	LDCKEC	W	Clock Generator Logic Duplication error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2
16	LDSSIFSMEC	W	SSI Controller/Target FSM Logic Duplication error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2
15	LDSLVIFEC	W	AHB Target FSM Logic Duplication error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2
14	DPUEC	W	Data Path ECC uncorrectable error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2
13	DPCEC	W	Data Path ECC correctable error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SAFETY_PKG_EN == 2
12	AXIUEC	W	AXI Manager Interface ECC uncorrectable error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_AXI_SAFETY_EN == 2
11	AXICEC	W	AXI Manager Interface ECC correctable error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_AXI_SAFETY_EN == 2
10	AXIVRPEC	W	AXI Valid/Ready Parity Error status clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_READY_VALID_PAR_EN == 1
9	SLVIFAPEC	W	AHB address parity failure interrupt clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> SSIC_SLVIF_PAR_EN > 1

Bits	Name	Memory Access	Description
8	SLVIFDPEC	W	AHB write data parity failure interrupt clear <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
7	REGERRRC	W	Register space error clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
6	DPPEC	W	Data path Parity checker error clear register <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
5	AXIPEC	W	AXI parity failure interrupt clear register. <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
4	RSVD_4_SAFETYICR	W	Reserved bit <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
3	SSIFSMTOC	W	SSI Core clock domain FSM time out error clear register <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
2	SLVIFSMTOC	W	AHB Subordinate Interface clock domain FSM time out error clear register <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
1	AXIFSMTOC	W	AXI clock domain FSM time out error clear register <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always
0	FSMPEC	W	FSM One-hot Error status clear register <b>Value After Reset:</b> 0x0 <b>Exists:</b> Always

# A

## Power Consumption and Area

---

This appendix discusses power and area requirements for the following configurations in DWC\_ssi:

- “Power and Area for typical configurations” on page [428](#)
- “Power and Area for Safety Prime Profiles” on page [431](#)

## A.1 Power and Area for typical configurations

[Table A-1](#) provides information about the synthesis results for typical configurations (power consumption, frequency, and area) and DFT coverage of the DWC\_ssi using the industry standard 7nm technology library.

Operating Frequencies:

aclk: 500MHz

hclk/pclk: 200MHz

ssi\_clk: 500MHz

**Table A-1 Power and Area Numbers for typical configurations**

<b>Configuration</b>	<b>Gate Count</b>	<b>Static Power (Watts)</b>	<b>Dynamic Power (Watts)</b>	<b>Tetramax (%)</b>		<b>SpyGlass StuckAtCov (%)</b>
				<b>StuckAtTest</b>	<b>Transition</b>	
Controller Max	59256	3.93e-4 W	1.94e-7 W	99.84	95.23	99.8
Controller with Internal DMA	50858	6.47e-4 W	1.70e-7 W	99.71	95.13	99.8
Target Max	22525	1.49e-4 W	7.29e-8 W	99.78	95.20	99.8
SPI Bridge	23174	2.15e-4 W	7.47e-8 W	99.68	95.60	99.9

[Table A-2](#) provides information about the configuration parameters used to get the area and power numbers.

**Table A-2 Configuration Settings for typical configurations**

<b>Parameter</b>	<b>Controller Max</b>	<b>Controller with Internal DMA</b>	<b>Target Max</b>	<b>SPI Bridge</b>
SSIC_SPI_BRIDGE	0	0	0	1
SSIC_IS_MASTER	1	1	0	0
SSIC_SLVIF_TYPE	1	1	1	0
APB_DATA_WIDTH	N/A	N/A	N/A	32
SSIC_ENH_CLK_RATIO	N/A	N/A	1	1
SSIC_NUM_SLAVES	4	2	N/A	N/A
SSIC_HAS_RX_SAMPLE_DELAY	0	0	N/A	N/A
SSIC_HAS_DMA	1	2	0	0
SSIC_TX_FIFO_DEPTH	32	16	32	32
SSIC_RX_FIFO_DEPTH	32	16	32	32
SSIC_SYNC_CLK	0	0	0	0

Parameter	Controller Max	Controller with Internal DMA	Target Max	SPI Bridge
SSIC_SPI_MODE	3	4	0	0
SSIC_AHB_M_SYNC_CLK	1	1	1	0
SSIC_AXIM_SYNC_CLK	1	0	1	1
SSIC_AXIM_SLVIF_SYNC_CLK	1	0	1	1
SSIC_HAS_CLK_LOOP_BACK	1	1	N/A	N/A
SSIC_SPI_DYN_WS_EN	0	0	N/A	N/A
SSIC_SLV_SPI_MODE	0	0	3	3
SSIC_INTR_POL	1	1	1	0
SSIC_INTR_IO	0	0	1	1
SSIC_IO_MAP_EN	1	1	N/A	N/A
SSIC_CLK_STRETCH_EN	1	1	N/A	N/A
SSIC_HAS_DDR	1	1	N/A	N/A
SSIC_HAS_RXDS	1	1	N/A	N/A
SSIC_XIP_EN	1	1	N/A	N/A
SSIC_XIP_WRITE_EN	1	0	N/A	N/A
SSIC_XIP_EXT_SSEL_EN	1	0	N/A	N/A
SSIC_CONCURRENT_XIP_EN	1	0	N/A	N/A
SSIC_XIP_WRITE_REG_EN	1	N/A	N/A	N/A
SSIC_XIP_RX_FIFO_DEPTH	32	N/A	N/A	N/A
SSIC_XIP_TX_FIFO_DEPTH	32	N/A	N/A	N/A
SSIC_XIP_CONT_XFER_EN	1	1	N/A	N/A
SSIC_XIP_PREFETCH_EN	1	0	N/A	N/A
SSIC_CONT_XFER_TIMEOUT_CNT_EN	1	1	N/A	N/A
SSIC_HYPERBUS_EN	1	1	N/A	N/A
SSIC_SPI_DM_EN	1	1	N/A	N/A
SSIC_AXI_INTF_TYPE	0	1	N/A	N/A
SSIC_AXI_DW	32	64	N/A	N/A
SSIC_AXI_AW	32	32	N/A	N/A

Parameter	Controller Max	Controller with Internal DMA	Target Max	SPI Bridge
SSIC_AXI_BLW	4	8	N/A	N/A
SSIC_AXI_IDW	6	12	N/A	N/A

## A.2 Power and Area for Safety Prime Profiles

[Table A-3](#) provides information about the synthesis results for safety prime profiles (power consumption, frequency, and area) and DFT coverage of the DWC\_ssi using the industry standard 7nm technology library.

Operating Frequencies:

aclk: 500MHz

hclk: 200MHz

ssi\_clk: 500MHz

**Table A-3 Power and Area Numbers for Safety Prime Profiles**

Configuration	Gate Count	Static Power (Watts)	Dynamic Power (Watts)	Tetramax (%)		SpyGlass StuckAtCov (%)
				StuckAtTest	Transition	
DWC_ssi Safety Package 1						
Controller Configuration	62891	8.90e-4 W	2.08e-7 W	99.65	95.10	99.9
DWC_ssi Safety Package 2						
Controller Configuration	50858	6.47e-4 W	1.70e-7 W	99.71	95.13	99.8
Target Configuration	22525	1.49e-4 W	7.29e-8 W	99.78	95.20	99.8
Controller/Target Programmable Configuration	23174	2.15e-4 W	7.47e-8 W	99.68	95.60	99.9

[Table A-6](#) provides information about the configuration parameters used to get area and power numbers. Please note, only the highlighted parameters can be changed during the configuration.

**Table A-4 Configuration Settings for Safety Prime Profiles**

Parameter Name	Controller	Safety Profile 2			Programmable Controller / Target
		Controller	Target		
SSIC_SAFETY_PKG_EN	1	2	2		2
SSIC_IS_MASTER	1	1	0		2
SSIC_ENH_CLK_RATIO	N/A	N/A	1		1
SSIC_NUM_SLAVES	2	4	N/A		N/A
SSIC_HAS_RX_SAMPLE_DELAY	0	0	N/A		2

**Table A-4 Configuration Settings for Safety Prime Profiles**

Parameter Name	Safety Profile 1 Controller	Safety Profile 2		
		Controller	Target	Programmable Controller / Target
SSIC_RX_DLY_SR_DEPTH	4	4	N/A	4
SSIC_HAS_DMA	2	2	0	0
SSIC_TX_FIFO_DEPTH	16	32	32	32
SSIC_RX_FIFO_DEPTH	16	32	32	32
SSIC_SYNC_CLK	0	0	0	0
SSIC_SPI_MODE	3	3	N/A	N/A
SSIC_AXIM_SYNC_CLK	0	0	N/A	N/A
SSIC_AXIM_SLVIF_SYNC_CLK	0	0	N/A	N/A
SSIC_BOOT_MODE_EN	1	1	1	1
SSIC_HAS_CLK_LOOP_BACK	1	1	N/A	N/A
SSIC_SPI_DYN_WS_EN	0	0	N/A	N/A
SSIC_SLV_SPI_MODE	N/A	N/A	0	0
SSIC_INTR_POL	1	1	1	1
SSIC_INTR_IO	0	0	0	0
SSIC_IO_MAP_EN	1	1	N/A	N/A
SSIC_CLK_STRETCH_EN	1	1	N/A	N/A
SSIC_HAS_DDR	1	1	N/A	N/A
SSIC_HAS_RXDS	1	1	N/A	N/A
SSIC_XIP_EN	1	1	N/A	N/A
SSIC_XIP_WRITE_EN	0	1	N/A	N/A
SSIC_XIP_EXT_SSEL_EN	0	0	N/A	N/A
SSIC_CONCURRENT_XIP_EN	0	0	N/A	N/A
SSIC_XIP_WRITE_REG_EN	0	0	N/A	N/A
SSIC_XIP_CONT_XFER_EN	1	0	N/A	N/A
SSIC_XIP_PREFETCH_EN	0	0	N/A	N/A
SSIC_CONT_XFER_TIMEOUT_CNT_EN	1	0	N/A	N/A

**Table A-4 Configuration Settings for Safety Prime Profiles**

Parameter Name	Safety Profile 1 Controller	Safety Profile 2		
		Controller	Target	Programmable Controller / Target
SSIC_HYPERBUS_EN	1	0	N/A	N/A
SSIC_SPI_DM_EN	1	0	N/A	N/A
SSIC_AXI_INTF_TYPE	1	1	0	0
SSIC_AXI_DW	64	64	N/A	N/A
SSIC_AXI_AW	32	64	N/A	N/A
SSIC_AXI_BLW	8	8	N/A	N/A
SSIC_AXI_IDW	4	6	6	6
SSIC_SLVIF_PAR_EN	2	2	2	2
SSIC_AXI_SAFETY_EN	2	2	0	0
SSIC_READY_VALID_PAR_EN	1	1	0	0
SSIC_PAR_TYPE	0	0	0	0
SSIC_ECC_GRAN_TYPE	1	2	1	1
SSIC_AR_REGSLICE_EN	1	1	0	0
SSIC_AW_REGSLICE_EN	1	1	0	0
SSIC_W_REGSLICE_EN	1	1	0	0
SSIC_R_REGSLICE_EN	1	1	0	0
SSIC_B_REGSLICE_EN	1	1	0	0
SSIC_H_2_S_SYNC_DEPTH	2	2	2	2
SSIC_S_2_H_SYNC_DEPTH	2	2	2	2
SSIC_A_2_S_SYNC_DEPTH	2	2	2	2
SSIC_S_2_A_SYNC_DEPTH	2	2	2	2
SSIC_A_2_H_SYNC_DEPTH	2	2	2	2
SSIC_H_2_A_SYNC_DEPTH	2	2	2	2



# B

## Serial Interface Timings

---

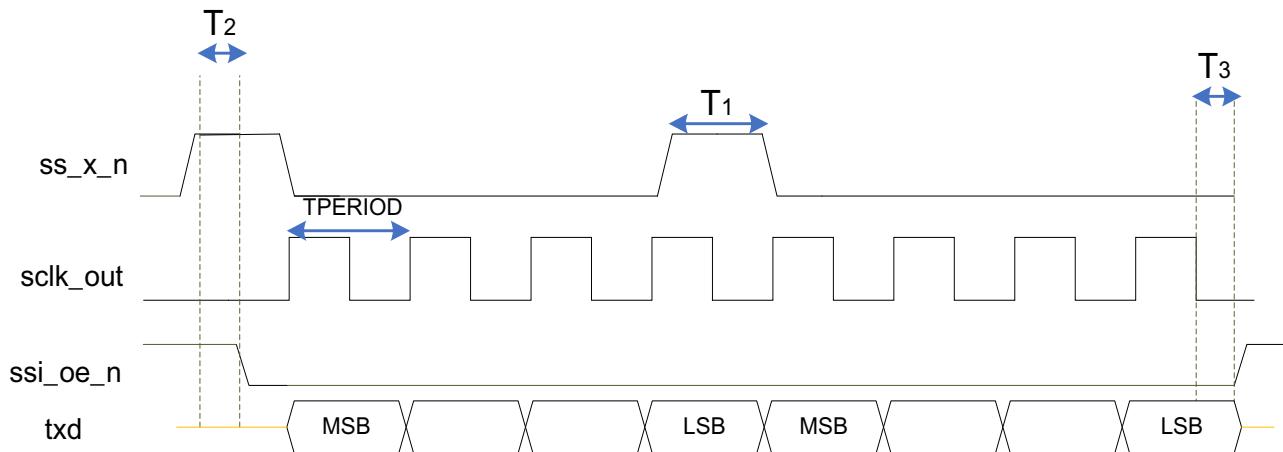
This section describes the serial interface timings of DWC\_ssi. This chapter includes following topics:

- “[SSP Protocol Timings](#)” on page [436](#)
- “[Microwire Protocol Timings](#)” on page [438](#)
- “[SPI Protocol Timings](#)” on page [440](#)

## B.1 SSP Protocol Timings

### B.1.1 Serial Controller Mode

**Figure B-1 SSP Serial Format Transfer Serial Controller Mode**

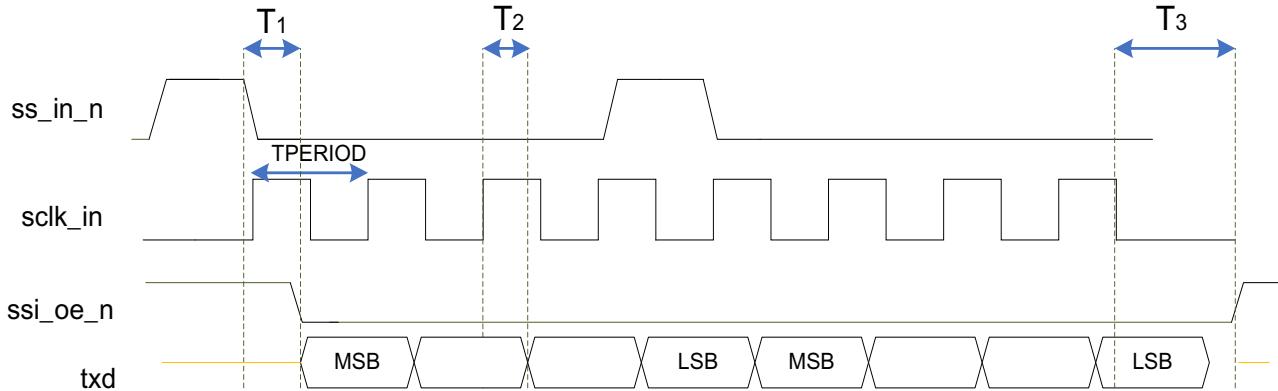


The following table displays the SSP Protocol Timings in Serial Controller mode:

Parameter	Symbol	Timing
Serial Clock Period	$T_{\text{PERIOD}}$	Baud rate *{ssi_clk period}
Chip Select Toggle Period	$T_1$	$T_{\text{PERIOD}}$
Chip Select rising edge to output enable assertion	$T_2$	$T_{\text{PERIOD}} / 2$
Last Clock edge to output enable de-assertion	$T_3$	$T_{\text{PERIOD}} / 2$

## B.1.2 Serial Target Mode

**Figure B-2 SSP Serial Format Transfer Serial Controller Mode**



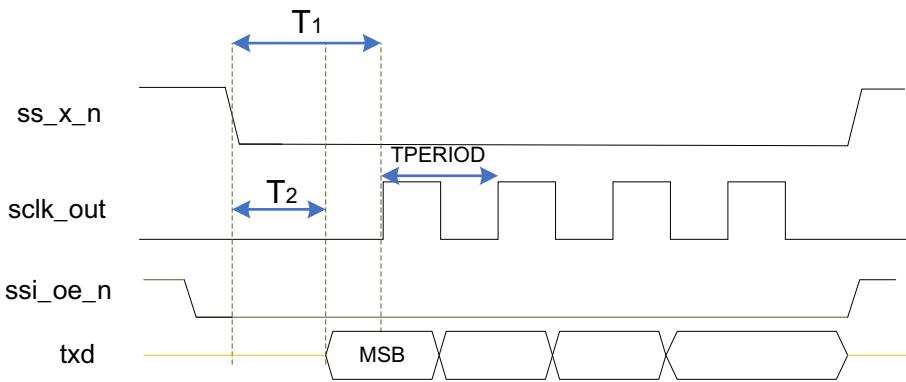
The following table describes the SSP Protocol Timings in Serial Target mode:

Parameter	Symbol	Timing(min)
<b>SSIC_ENH_CLK_RATIO = 0</b>		
Serial Clock Period	$T_{PERIOD}$	6 *{ssi_clk period} for Receive only 8*{ssi_clk period} otherwise
First rising edge of clock to output enable assertion	$T_1$	2*{ssi_clk period}
Rising edge of clock to data transmission	$T_2$	2*{ssi_clk period}
Last Clock edge to output enable de-assertion	$T_3$	3*{ssi_clk period}
<b>SSIC_ENH_CLK_RATIO = 1</b>		
Serial Clock Period	$T_{PERIOD}$	4 *{ssi_clk period}
First rising edge of clock to output enable assertion	$T_1$	1*{ssi_clk period}
Rising edge of clock to data transmission	$T_2$	1*{ssi_clk period}
Last Clock edge to output enable de-assertion	$T_3$	1/*{ssi_clk period}

## B.2 Microwire Protocol Timings

### B.2.1 Serial Controller Mode

**Figure B-3** Microwire Transfer in Serial Controller Mode

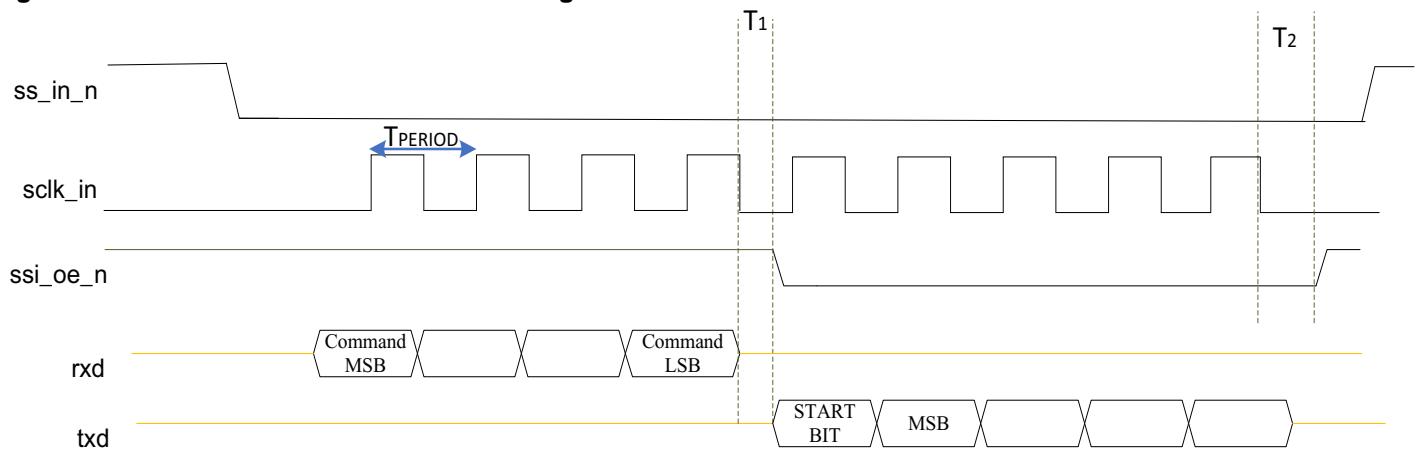


The following table describes the Microwire Protocol Timings Serial Controller mode:

Parameter	Symbol	Timing
Serial Clock Period	$T_{PERIOD}$	Baud rate *{ssi_clk period}
Serial Select to first clock edge	$T_1$	$1.5 * T_{PERIOD}$
Chip Select to first data transmission	$T_2$	$T_{PERIOD}$

### B.2.2 Serial Target Mode

**Figure B-4** Microwire Transfer in Serial Target Mode



The following table describes the Microwire Protocol Timings in Serial Target mode:

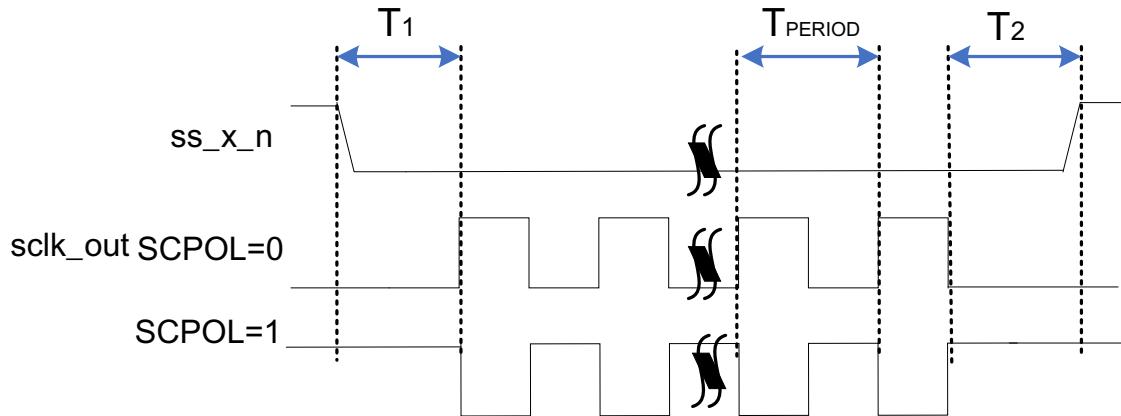
Parameter	Symbol	Timing(min)
<b>SSIC_ENH_CLK_RATIO = 0</b>		
Serial Clock Period	$T_{\text{PERIOD}}$	6 *{ssi_clk period} for Receive only 8*{ssi_clk period} otherwise
First Transmit edge to valid data and ssi_oe_n assertion	$T_1$	2*{ssi_clk period}
Last Sampling edge to ssi_oe_n de-assertion	$T_2$	2*{ssi_clk period}
<b>SSIC_ENH_CLK_RATIO = 1</b>		
Serial Clock Period	$T_{\text{PERIOD}}$	4 *{ssi_clk period}
First Transmit edge to valid data and ssi_oe_n assertion	$T_1$	1*{ssi_clk period}
Last Sampling edge to ssi_oe_n de-assertion	$T_2$	1*{ssi_clk period}

## B.3 SPI Protocol Timings

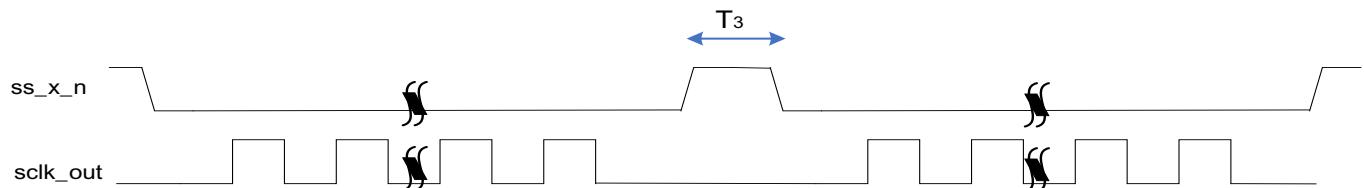
### B.3.1 Serial Controller Mode

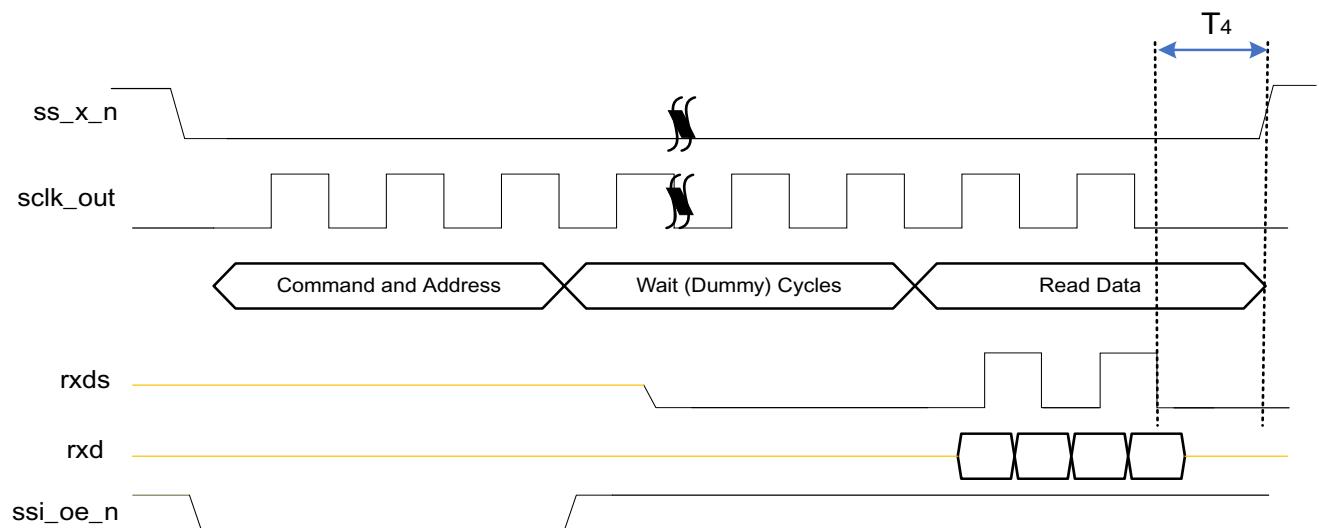
This section describes the interface timings for SPI serial interface in DWC\_ssi.

**Figure B-5 Clock and Chip Select Timings in Serial Controller Mode**



**Figure B-6 SPI Transfer with SCPH0 and CTRLR0.SSTE=1**



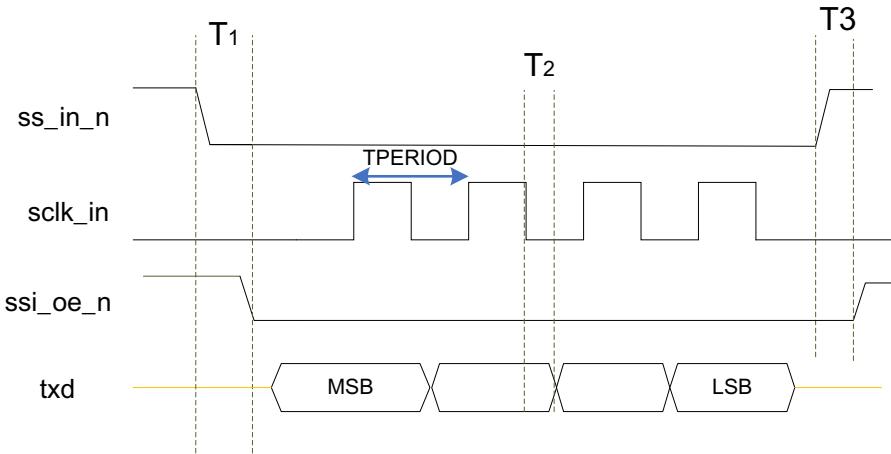
**Figure B-7 SPI DDR Read with Read Data Strobe**

The following table describes SPI Protocol Timings in Serial Controller mode:

Parameter	Symbol	Timing
Serial Clock Period	$T_{\text{PERIOD}}$	Baud rate *{ssi_clk period}
Chip select LOW to clock first edge	$T_1$	$T_{\text{PERIOD}}$ (when SCPH=1) $1.5 * T_{\text{PERIOD}}$ (when SCPH=0)
Clock Last edge to chip select HIGH	$T_2$	$T_{\text{PERIOD}}$
Chip Select toggle when CTLRLR0 . SSTE=1	$T_3$	$T_{\text{PERIOD}}$
RXDS Last edge to Chip select HIGH	$T_4$	$T_{\text{PERIOD}}$

### B.3.2 Serial Target Mode

**Figure B-8 SPI Transfer in Serial Target mode with SCPH=0 and SCPOL=0**



The following table describes the SPI Protocol Timings in Serial Target mode:

Parameter	Symbol	Timing (min)
<b>SSIC_ENH_CLK_RATIO = 0</b>		
Serial Clock Period	$T_{\text{PERIOD}}$	$6 \times \{\text{ssi\_clk period}\}$ for Receive only $8 \times \{\text{ssi\_clk period}\}$ otherwise
Chip Select to <b>ssi_oe_n</b> assertion (and data transmission in case of <b>SCPH=0</b> )	$T_1$	$2 \times \{\text{ssi\_clk period}\}$
Serial Clock driving edge to data transmission	$T_2$	$2 \times \{\text{ssi\_clk period}\}$
Chip Select de-assertion to <b>ssi_oe_n</b> de-assertion	$T_3$	$2 \times \{\text{ssi\_clk period}\}$
<b>SSIC_ENH_CLK_RATIO = 1</b>		
Serial Clock Period	$T_{\text{PERIOD}}$	$4 \times \{\text{ssi\_clk period}\}$
Chip Select to <b>ssi_oe_n</b> assertion (and data transmission in case of <b>SCPH=0</b> )	$T_1$	$1 \times \{\text{ssi\_clk period}\}$
Serial Clock driving edge to data transmission	$T_2$	$1 \times \{\text{ssi\_clk period}\}$

Parameter	Symbol	Timing (min)
Chip Select de-assertion to ssi_oe_n de-assertion	T <sub>3</sub>	1*{ssi_clk period}



## 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
AR_PYLD0_USER_WIDTH	(SSIC_AXI_AW/8 + ((SSIC_AXI_AW%8) ? 1 : 0))
AR_PYLD1_USER_WIDTH	4
AW_PYLD0_USER_WIDTH	(SSIC_AXI_AW/8 + ((SSIC_AXI_AW%8) ? 1 : 0))
AW_PYLD1_USER_WIDTH	4
B_PYLD0_USER_WIDTH	2
DM_W	(SSIC_SPI_MODE == 4 ? 2 : 1)
LOG2_SSIC_NUM_SLAVES	[function_of: SSIC_NUM_SLAVES]
MIOW	(SSIC_SPI_MODE == 4 ? 16 : (SSIC_SPI_MODE == 3 ? 8 : (SSIC_SPI_MODE == 1 ? 2 : (SSIC_SPI_MODE == 2 ? 4 : 1))))
R_PYLD0_USER_WIDTH	((SSIC_AXI_SAFETY_EN==2) ? ((SSIC_AXI_DW==32) ? 7 : (SSIC_ECC_GRAN_TYPE==2 ? 8 : 14)) : (SSIC_AXI_DW/8))
R_PYLD1_USER_WIDTH	2
RX_ABW	[function_of: SSIC_RX_FIFO_DEPTH]

Parameter Name	Equals To
SERIOW	(SSIC_IS_MASTER == 0 ? SIOW : MIOW)
SIOW	(SSIC_SLV_SPI_MODE == 3 ? 8 : (SSIC_SLV_SPI_MODE == 1 ? 2 : (SSIC_SLV_SPI_MODE == 2 ? 4 : 1)))
SNPS_RSVDPARAM_15	0x0
SNPS_RSVDPARAM_21	SSIC_PAR_MODE==0 && SSIC_AXI_SAFETY_EN!=0
SNPS_RSVDPARAM_4	0x0
SNPS_RSVDPARAM_7	(SSIC_SAFETY_PKG_EN == 1)
SNPS_RSVDPARAM_8	0x0
SNPS_RSVDPARAM_9	0x0
SPI_W	(SSIC_SPI_MODE == 4 ? 3 : 2)
SSIC_AXI_ARUW	((SNPS_RSVDPARAM_21 == 1) ? ([function_of: SSIC_AXI_PAR_ARW]) : (AR_PYLD1_USER_WIDTH + AR_PYLD0_USER_WIDTH))
SSIC_AXI_AWUW	((SNPS_RSVDPARAM_21 == 1) ? ([function_of: SSIC_AXI_PAR_AWW]) : (AW_PYLD1_USER_WIDTH + AW_PYLD0_USER_WIDTH))
SSIC_AXI_BUW	((SNPS_RSVDPARAM_21 == 1) ? ([function_of: SSIC_AXI_PAR_BW]) : (B_PYLD0_USER_WIDTH))
SSIC_AXI_INTF_AXI4	((SSIC_AXI_INTF_TYPE == 1) ? 1 : 0)
SSIC_AXI_LTW	((SSIC_AXI_INTF_TYPE == 0) ? 2 : 1)
SSIC_AXI_PAR_ARW	[function_of: SSIC_AXI_AW SSIC_AXI_IDW SSIC_AXI_BLW SSIC_AXI_LTW]
SSIC_AXI_PAR_AWW	[function_of: SSIC_AXI_AW SSIC_AXI_IDW SSIC_AXI_BLW SSIC_AXI_LTW]
SSIC_AXI_PAR_BW	[function_of: SSIC_AXI_IDW]
SSIC_AXI_PAR_RW	[function_of: SSIC_AXI_DW SSIC_AXI_IDW]
SSIC_AXI_PAR_WW	[function_of: SSIC_AXI_DW SSIC_AXI_IDW SSIC_AXI_WSW SSIC_AXI_INTF_AXI4]
SSIC_AXI_RUW	((SNPS_RSVDPARAM_21 == 1) ? ([function_of: SSIC_AXI_PAR_RW]) : (R_PYLD1_USER_WIDTH + R_PYLD0_USER_WIDTH))
SSIC_AXI_WSW	( (SSIC_AXI_DW == 32) ? 4 : (SSIC_AXI_DW == 64) ? 8 : 4 )

Parameter Name	Equals To
SSIC_AXI_WUW	((SNPS_RSVDPARAM_21 == 1) ? ([function_of: SSIC_AXI_PAR_WW]) : (W_PYLD1_USER_WIDTH + W_PYLD0_USER_WIDTH))
SSIC_COMBINED	1
SSIC_INDIVIDUAL	0
SSIC_PAR_MODE	(SSIC_AXI_SAFETY_EN == 2) ? 1: 0
SSIC_RAM_AW	[function_of: SSIC_RAM_DEPTH]
SSIC_VERSION_ID	0x3230302a
TX_ABW	[function_of: SSIC_TX_FIFO_DEPTH]
TXFTHR_W	((SSIC_HAS_DMA == 2) ? TX_ABW+3 : TX_ABW)
W_PYLD0_USER_WIDTH	((SSIC_AXI_SAFETY_EN==2) ? ((SSIC_AXI_DW==32) ? 7 : (SSIC_ECC_GRAN_TYPE==2 ? 8 : 14)) : (SSIC_AXI_DW/8))
W_PYLD1_USER_WIDTH	((SSIC_AXI_INTF_TYPE == 0) ? 3 : 2)

