DesignWare DW_apb_i2c Databook

DW_apb_i2c – Product Code

2.02a
July 2018

Copyright Notice and Proprietary Information

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.

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.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com

Contents

Revision History 9

Preface 15
Organization 15
Related Documentation 16
Web Resources 16
Customer Support 16
Product Code 17

Chapter 1
Product Overview 19
1.1 DesignWare System Overview 19
1.2 General Product Description 21
1.2.1 DW_apb_i2c Block Diagram 21
1.3 Features 22
1.3.1 I2C Features 22
1.3.2 DesignWare APB Slave Interface 23
1.4 Standards Compliance 23
1.5 Verification Environment Overview 23
1.6 Licenses 23
1.7 Where To Go From Here 23

Chapter 2
Functional Description 25
2.1 Overview 26
2.2 I2C Terminology 28
2.2.1 I2C Bus Terms 28
2.2.2 Bus Transfer Terms 29
2.3 I2C Behavior 30
2.3.1 START and STOP Generation 30
2.3.2 Combined Formats 31
2.4 I2C Protocols 31
2.4.1 START and STOP Conditions 31
2.4.2 Addressing Slave Protocol 32
2.4.3 Transmitting and Receiving Protocol 33
2.4.4 START BYTE Transfer Protocol 35
2.5 Tx FIFO Management and START, STOP and RESTART Generation 36
2.5.1 Tx FIFO Management When IC_EMPTYFIFO_HOLD_MASTER_EN = 0 36
2.5.2 Tx FIFO Management When IC_EMPTYFIFO_HOLD_MASTER_EN = 1 38
2.6 Multiple Master Arbitration 41

2.7 Clock Synchronization 43
2.8 Operation Modes 43
2.8.1 Slave Mode Operation 44
2.8.2 Master Mode Operation 47
2.8.3 Disabling DW_apb_i2c 50
2.8.4 Aborting I2C Transfers 51
2.9 Spike Suppression 51
2.10 Fast Mode Plus Operation 53
2.11 Bus Clear Feature 53
2.11.1 SDA Line Stuck at LOW Recovery 53
2.11.2 SCL Line is Stuck at LOW 54
2.12 Device ID 54
2.13 Ultra-Fast Speed Mode 55
2.14 SMBus/PMBus 56
2.14.1 tTimeout,MIN Parameter 56
2.14.2 Master Device Clock Extension 57
2.14.3 Slave Device Clock Extension 57
2.14.4 SMBDAT Low Timeout 57
2.14.5 Bus Protocols 57
2.14.6 SMBUS Address Resolution Protocol 59
2.14.7 SMBUS Additional Slave Address 64
2.14.8 SMBUS Optional Signals 65
2.15 IC_CLK Frequency Configuration 66
2.15.1 Minimum High and Low Counts in SS, FS, FM+ and high speed Modes With IC_CLK_FREQ_OPTIMIZATION = 0. 67
2.15.2 Minimum High and Low Counts in SS, FS, FM+ and high speed Modes With IC_CLK_FREQ_OPTIMIZATION = 1 69
2.15.3 Minimum High and Low counts in Ultra-Fast mode (IC_ULTRA_FAST_MODE = 1) 69
2.15.4 Minimum IC_CLK Frequency 69
2.16 SDA Hold Time 77
2.16.1 SDA Hold Timings in Receiver 78
2.16.2 SDA Hold Timings in Transmitter 79
2.17 DMA Controller Interface 80
2.17.1 Enabling the DMA Controller Interface 81
2.17.2 Overview of Operation 81
2.17.3 Transmit Watermark Level and Transmit FIFO Underflow 83
2.17.4 Choosing the Transmit Watermark Level 83
2.17.5 Selecting DEST_MSIZE and Transmit FIFO Overflow 85
2.17.6 Receive Watermark Level and Receive FIFO Overflow 85
2.17.7 Choosing the Receive Watermark level 86
2.17.8 Selecting SRC_MSIZE and Receive FIFO Underflow 86
2.17.9 Handshaking Interface Operation 86
2.18 APB Interface 90
2.18.1 APB 3.0 Support 90
2.18.2 APB 4.0 Support 91
2.19 I/O Connections 91
2.20 DW_apb_i2c Registers 92
2.20.1 Registers and Field Descriptions 92
2.20.2 Operation of Interrupt Registers 92

2.21 UDID Feature 94

Chapter 3
Parameter Descriptions 95
3.1 Top Level Parameters 96
3.2 I2C Version 3.0 Features Parameters 112
3.3 SMBus Features Parameters 114
3.4 I2C Version 6.0 Features Parameters 117

Chapter 4
Signal Descriptions 119
4.1 Interrupts Signals 121
4.2 I2C Interface (Master/Slave) Signals 129
4.3 APB Slave Interface Signals 132
4.4 DMA Interface Signals 135
4.5 SMBus Interface Signals 137
4.6 I2C Debug Signals 138

Chapter 5
Register Descriptions 141
5.1 DW_apb_i2c_mem_map/DW_apb_i2c_addr_block1 Registers 144
5.1.1 IC_CON 148
5.1.2 IC_TAR 156
5.1.3 IC_SAR 160
5.1.4 IC_HS_MADDR 161
5.1.5 IC_DATA_CMD 162
5.1.6 IC_SS_SCL_HCNT 166
5.1.7 IC_UFM_SCL_HCNT 168
5.1.8 IC_SS_SCL_LCNT 170
5.1.9 IC_UFM_SCL_LCNT 172
5.1.10 IC_FS_SCL_HCNT 174
5.1.11 IC_UFM_TBUF_CNT 176
5.1.12 IC_FS_SCL_LCNT 178
5.1.13 IC_HS_SCL_HCNT 180
5.1.14 IC_HS_SCL_LCNT 182
5.1.15 IC_INTR_STAT 184
5.1.16 IC_INTR_MASK 189
5.1.17 IC_RAW_INTR_STAT 193
5.1.18 IC_RX_TL 202
5.1.19 IC_TX_TL 203
5.1.20 IC_CLR_INTR 204
5.1.21 IC_CLR_RX_UNDER 205
5.1.22 IC_CLR_RX_OVER 206
5.1.23 IC_CLR_TX_OVER 207
5.1.24 IC_CLR_RD_REQ 208
5.1.25 IC_CLR_TX_ABRT 209
5.1.26 IC_CLR_RX_DONE 210
5.1.27 IC_CLR_ACTIVITY 211
5.1.28 IC_CLR_STOP_DET 212
5.1.29 IC_CLR_START_DET 213

5.1.30 IC_CLR_GEN_CALL 214
5.1.31 IC_ENABLE 215
5.1.32 IC_STATUS 220
5.1.33 IC_TXFLR 227
5.1.34 IC_RXFLR 228
5.1.35 IC_SDA_HOLD 229
5.1.36 IC_TX_ABRT_SOURCE 231
5.1.37 IC_SLV_DATA_NACK_ONLY 240
5.1.38 IC_DMA_CR 242
5.1.39 IC_DMA_TDLR 244
5.1.40 IC_DMA_RDLR 245
5.1.41 IC_SDA_SETUP 246
5.1.42 IC_ACK_GENERAL_CALL 248
5.1.43 IC_ENABLE_STATUS 249
5.1.44 IC_FS_SPKLEN 253
5.1.45 IC_UFM_SPKLEN 254
5.1.46 IC_HS_SPKLEN 256
5.1.47 IC_CLR_RESTART_DET 258
5.1.48 IC_SCL_STUCK_AT_LOW_TIMEOUT 259
5.1.49 IC_SDA_STUCK_AT_LOW_TIMEOUT 260
5.1.50 IC_CLR_SCL_STUCK_DET 261
5.1.51 IC_DEVICE_ID 262
5.1.52 IC_SMBUS_CLK_LOW_SEXT 263
5.1.53 IC_SMBUS_CLK_LOW_MEXT 264
5.1.54 IC_SMBUS_THIGH_MAX_IDLE_COUNT 265
5.1.55 IC_SMBUS_INTR_STAT 267
5.1.56 IC_SMBUS_INTR_MASK 271
5.1.57 IC_SMBUS_RAW_INTR_STAT 275
5.1.58 IC_CLR_SMBUS_INTR 280
5.1.59 IC_OPTIONAL_SAR 283
5.1.60 IC_SMBUS_UDID_LSB 284
5.1.61 IC_SMBUS_UDID_WORD0 285
5.1.62 IC_SMBUS_UDID_WORD1 286
5.1.63 IC_SMBUS_UDID_WORD2 287
5.1.64 IC_SMBUS_UDID_WORD3 288
5.1.65 REG_TIMEOUT_RST 289
5.1.66 IC_COMP_PARAM_1 291
5.1.67 IC_COMP_VERSION 294
5.1.68 IC_COMP_TYPE 295

Chapter 6
Programming the DW_apb_i2c 297
6.1 Software Registers 297
6.2 Software Drivers 297
6.3 Programming Example 298
6.4 Programming Flow for SCL and SDA Bus Recovery 304
6.5 Programming Flow for Reading the Device ID 305
6.6 Programming Flow for SMBUS Timeout in Master Mode 306
6.7 Programming Flow for SMBUS Timeout in Slave Mode 307

6.8 ARP Master Programming Flow 307
6.9 ARP Slave Programming Flow 308
6.10 SMBus SUSPEND Programming Flow in Host Mode 311
6.11 SMBus SUSPEND Programming Flow in Device Mode 312
6.12 SMBus ALERT Programming Flow in Host Mode 313
6.13 SMBus ALERT Programming Flow in Device Mode 314
6.14 Programming Flow Of DW_apb_i2c in Ultra-Fast Mode 315
6.14.1 DW_apb_i2c Master Mode 315
6.14.2 DW_apb_i2c Slave Mode 316

Chapter 7
Verification 317
7.1 Vera Testbench Environment 317
7.1.1 Overview of Vera Tests 317
7.1.2 APB Slave Interface 317
7.1.3 DW_apb_i2c Master Operation 318
7.1.4 DW_apb_i2c Slave Operation 318
7.1.5 DW_apb_i2c Interrupts 319
7.1.6 DMA Handshaking Interface 319
7.1.7 DW_apb_i2c Dynamic IC_TAR and IC_10BITADDR_MASTER Update 319
7.1.8 Generate NACK as a Slave-Receiver 319
7.1.9 SCL Held Low for Duration Specified in IC_SDA_SETUP 319
7.1.10 Generate ACK/NACK for General Call 320

Chapter 8
Integration Considerations 321
8.1 Accessing Top-level Constraints 321
8.2 Performance 321
8.2.1 Power Consumption, Frequency, and Area Results 321

Appendix A
Synchronizer Methods 325
A.1 Synchronizers Used in DW_apb_i2c 326
A.2 Synchronizer 1: Simple Double Register Synchronizer 327
A.3 Synchronizer 2: Simple Double Register Synchronizer with Configurable Polarity Reset 327

Chapter B
Internal Parameter Descriptions 329

Appendix C
Glossary 331

Index 335

Revision History

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

Version	Date	Description
2.02a	July 2018	**Added**: ■ “APB 3.0 Support” on page 90 ■ “APB 4.0 Support” on page 91 ■ “UDID Feature” on page 94 **Removed**: ■ Chapter 2, “Building and Verifying a Component or Subsystem” and added the contents in the newly created user guide. ■ Section “Running Leda on Generated Code with coreConsultant” ■ Removed all instances of Leda **Updated:** ■ Version number changed for 2018.07a release ■ “Performance” on page 321 ■ “Parameter Descriptions” on page 95, “Signal Descriptions” on page 119, “Register Descriptions” on page 141 and “Internal Parameter Descriptions” on page 329 are auto extracted with change bars from the RTL.

(Continued)

Version	Date	Description
2.01a	October 2016	**Added**: ■ "Running VCS XPROP Analyzer" ■ Entry for the xprop directory in Table 2-1 and Table 2-4. ■ Added note in "Overview" on page 26 and "Tx FIFO Management When IC_EMPTYFIFO_HOLD_MASTER_EN = 0" on page 36 ■ Parameter Descriptions and Register Descriptions auto-extracted from the RTL **Removed**: ■ Removed the "Running Leda on Generated Code with coreConsultant" section, and reference to Leda directory in Table 2-1 ■ Removed the "Running Leda on Generated Code with coreAssembler" section, and reference to Leda directory in Table 2-4 **Updated:** ■ Version number changed for 2016.10a release ■ Modified Table 2-2 on page 58 ■ Updated area and power numbers in sections "Area" and "Power Consumption" ■ Modified "APB Interface" on page 90 ■ Updated description for SMBus ■ Updated the ic_smbalert_oe signal description ■ Moved Internal Parameter Descriptions to Appendix
2.00a	June 2015	**Added**: ■ "Running SpyGlass® Lint and SpyGlass® CDC" ■ "Running SpyGlass on Generated Code with coreAssembler" ■ "Internal Parameter Descriptions" on page 329 ■ New features: - "Bus Clear Feature" on page 53 - "Device ID" on page 54 - "SMBus/PMBus" on page 56 - "Ultra-Fast Speed Mode" on page 55 - New parameter "IC_CLK_FREQ_OPTIMIZATION" - Synchronizer Methods ■ Included a note regarding tBUF timing and setup/hold time. **Updated**: - "IC_CLK Frequency Configuration" on page 66 updated for IC_CLK_FREQ_OPTIMIZATION and IC_ULTRA_FAST_MODE Configurations - "Signal Descriptions" on page 119 auto-extracted from the RTL

(Continued)

Version	Date	Description
1.22a	June 2014	**Added:** ■ New features: - Blocking the Tx FIFO commands using IC_TX_CMD_BLOCK field in IC_ENABLE register - Indication for first data byte received after the address in IC_DATA_CMD register - Detection of STOP interrupt only if master is active ■ coreConsultant parameter (IC_AVOID_RX_FIFO_FLUSH_ON_TX_ABRT) introduced to avoid flushing of RX FIFO during TX Abort ■ New bits in IC_STATUS register for Indicating a reason for bus holding ■ Performance section in Integration considerations **Updated:** ■ Width of TX_FLUSH_CNT field in the IC_TX_ABRT_SOURCE register ■ Default Input/Output Delays in Signals chapter
1.21a	May 2013	**Added:** ■ Section on Fast Mode Plus ■ Configuration Parameters: - IC_RX_FULL_HLD_BUS_EN - IC_SLV_RESTART_DET_EN ■ Signals: - ic_restart_det_intr(_n) signal to enable restart detect in slave mode ■ Registers - RESTART_DET bit of IC_INTR_STAT, IC_INTR_MASK and IC_RAW_INTR_STAT registers Bit detects a repeated start when the DW_apb_i2c is the addressed slave - IC_CLR_RESTART_DET to clear the RESTART_DET interrupt - MST_ON_HOLD bit to the IC_INTR_STAT, IC_INTR_MASK and IC_RAW_INTR_STAT registers. This bit indicates whether a master is holding the bus and the Tx FIFO is empty. Added the signal ic_mst_on_hold_intr(_n) ■ Programming flow for DW_apb_i2c master with TAR update

(Continued)

Version	Date	Description
1.21a *Cont'd*	May 2013 *Cont'd*	*Continued* **Updated:** ■ References to Fast Mode Plus ■ Registers: - TX_FLUSH_CNT field of the IC_TX_ABRT_SOURCE register - TX_ABRT field of the IC_RAW_INTR_STAT register - IC_CON - IC_RAW_INTR_STAT - IC_SDA_HOLD **Signals:** ■ Active state of the ic_current_src_en signal ■ Programming flow for DW_apb_i2c as master in standard or fast mode ■ Method for deriving ic_clk values in high speed modes ■ Documentation template **Removed:** ■ Text stating that Fast Mode Plus is not supported ■ Note in the IC_TX_ABRT_SOURCE register description stating DW_apb_i2c can be a master and slave at the same time
1.20a	Oct 2012	Added the product code on the cover and in Table 1-1.
1.20a	June 2012	Edited calculations for driving SDA in "high speed Modes" section; updated IC_ENABLE and IC_TX_ABRT_SOURCE registers.
1.17a	Mar 2012	Enhanced DW_ahb_dmac and DW_apb_i2c programming example; updated definition of IC_FS_SPKLEN and IC_HS_SPKLEN register descriptions; corrected programming values for dma_tx_req and dma_rx_req signals.
1.16b	Dec 2011	Enhanced description of IC_ADD_ENCODED_PARAMS parameter.
1.16b	Nov 2011	Version change for 2011.11a release.
1.16a	Oct 2011	Version change for 2011.10a release.
1.15a	14 June 2011	Removed "Digital/Analog Domain Functional Partitioning" section (9.1) – irrelevant now with Spike Suppression functionality.
1.15a	June 2011	Updated system diagram in Figure 1-1; enhanced description of ic_rst_n signal; enhanced "Related Documents" section in Preface.
1.15a	21 Apr 2011	Clarified description of C_DEFAULT_SDA_HOLD parameter.
1.15a	12 Apr 2011	Corrected IC_DEFAULT_FS_SPKLEN and IC_DEFAULT_HS_SPKLEN default values.
1.15a	Apr 2011	Added spike suppression material; corrected R/W locations in timing diagrams in "Tx FIFO Management and START, STOP and RESTART Generation" section
1.14a	Dec 2010	Corrected subsection numbering in Registers chapter.

(Continued)

Version	Date	Description
1.13a	Oct 2010	Added information on calculating maximum value for IC_DEFAULT_SDA_HOLD parameter and IC_SDA_HOLD register; “SDA Hold Time” section, description of IC_DEFAULT_SDA_HOLD parameter, and IC_SDA_HOLD register updated
1.12a	7 Sep 2010	Corrected DW_ahb_dmac response in “Receive Watermark Level and Receive FIFO Overflow” section
1.12a	Sep 2010	Corrected names of include files and vcs command used for simulation
1.11a	Mar 2010	Corrected information regarding how DW_apb_i2c communicates with slaves when operating in master mode; corrected default value for IC_DEFAULT_SDA_SETUP parameter; added SDA hold time information; added IC_SDA_HOLD register description; removed references to 300ns hold time in integration considerations; removed DW_apb_i2c Application Notes appendix.
1.10a	Jan 2010	Removed reference to I2C protocol created by Philips (NXP).
1.10a	Dec 2009	Corrected dependencies for IC_SS_SCL_HIGH_COUNT, IC_SS_SCL_LOW_COUNT, IC_FS_SCL_HIGH_COUNT, and IC_FS_SCL_LOW_COUNT parameters; corrected IC_RESTART_EN parameter description; modified description of IC_SDA_SETUP register; updated databook to new template for consistency with other IIP/VIP/PHY databooks.
1.10a	Jul 2009	Corrected equations for avoiding underflow when programming a source burst transaction.
1.10a	Jun 2009	Corrected name of IC_10BITADDR_SLAVE parameter in “Parameters” chapter.
1.10a	May 2009	Removed references to QuickStarts, as they are no longer supported.
1.10a	24 Apr 2009	Enhanced IC_CON description with table for IC_SLAVE_DISABLE and MASTER_MODE combinations that result in configuration errors.
1.10a	23 Apr 2009	Enhanced “Master Transmit and Master Receive” subsection to clarify reads for multiple bytes.
1.10a	Oct 2008	IC_RX_FULL_GEN_NACK parameter removed; IC_INTR_MASK is active low; dependency changed for IC_HS_MASTER_CODE parameter; IC_SLAVE_DISABLE default changed to 1; values for high speed mode corrected in Table 8; debug_* signal default values corrected; version change for 2008.10a release.
1.09a	Jul 2008	Removed IC_RX_FULL_GEN_NACK configuration parameter and its conditional text. Changed reference to non-existent table for IC_*S_SCL_*CNT registers to link to “IC_CLK Frequency Configuration” section. Removed USE_FOUNDATION parameter.
1.09a	Jun 2008	Removed Synchronous value from IC_CLK_TYPE parameter; clarified that putting data into the FIFO generates a START and emptying the FIFO generates a STOP; clarified description of I2C_DYNAMIC_TAR_UPDATE parameter; clarification of IC_TAR description.
1.08b	11 Feb 2008	Modified note on restriction; page 47.

Preface

This databook provides information that you need to interface the DW_apb_i2c to the Advanced Peripheral Bus (APB). The DW_apb_i2c conforms to the *AMBA Specification, Revision 2.0* from Arm®.

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

Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Functional Description" describes the functional operation of the DW_apb_i2c.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_apb_i2c.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_apb_i2c signals.
- Chapter 5, "Register Descriptions" describes the programmable registers of the DW_apb_i2c.
- Chapter 6, "Programming the DW_apb_i2c" provides information needed to program the configured DW_apb_i2c.
- Chapter 7, "Verification" provides information on verifying the configured DW_apb_i2c.
- Chapter 8, "Integration Considerations" includes information you need to integrate the configured DW_apb_i2c into your design.
- Chapter A, "Synchronizer Methods" documents the synchronizer methods (blocks of synchronizer functionality) used in DW_apb_i2c to cross clock boundaries.
- Appendix B, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals chapter.
- Appendix C, "Glossary" provides a glossary of general terms.

Related Documentation

- *DW_apb_i2c Driver Kit User Guide* – Contains information on the Driver Kit for the DW_apb_i2c; requires source code license (DWC-APB-Periph-Source)
- *Using DesignWare Library IP in coreAssembler* – Contains information on getting started with using DesignWare SIP components for AMBA 2 and AMBA 3 AXI components within coreTools
- *coreAssembler User Guide* – Contains information on using coreAssembler
- *coreConsultant User Guide* – Contains information on using coreConsultant

To see a complete listing of documentation within the DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI, see the https://www.synopsys.com/dw/doc.php/doc/amba/latest/intro.pdf.

Web Resources

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

Customer Support

To obtain support for your product:

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

    File > Build Debug Tar-file

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

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

- **Product:** DesignWare Library IP
- **Sub Product:** AMBA
- **Tool Version:** <*product version number*>
- **Problem Type:**
- **Priority:**
- **Title:** DW_apb_i2c
- **Description:** For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

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

- Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
  - Include the Product name, Sub Product name, and Tool Version number in your e-mail (as identified earlier) so it can be routed correctly.
  - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
  - Attach any debug files you created in the previous step.
- Or, telephone your local support center:
  - North America:

    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - All other countries:

    https://www.synopsys.com/support/global-support-centers.html

Product Code

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

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

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

1 Product Overview

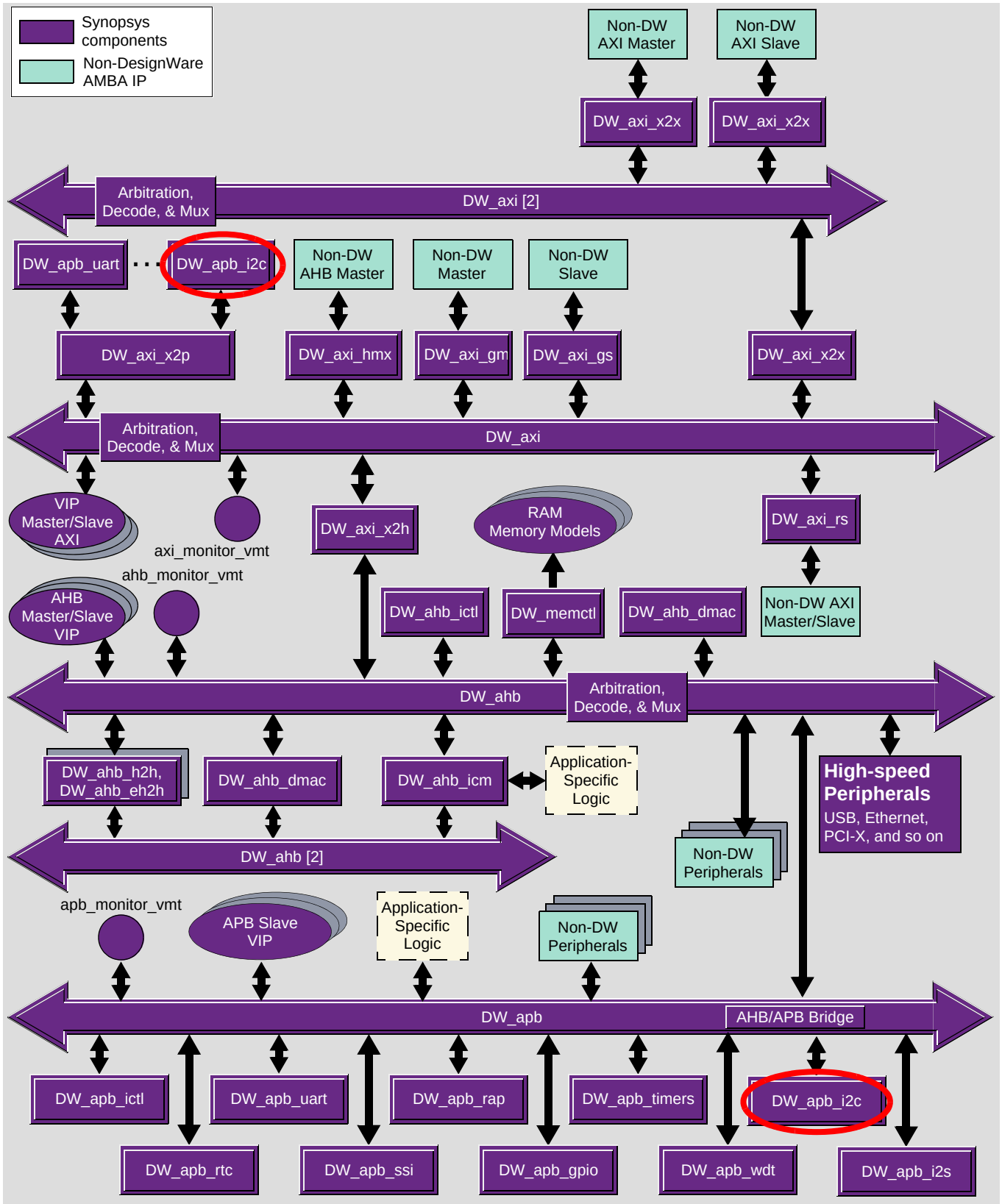
This chapter describes the DesignWare APB I^2C Interface Peripheral, referred to as DW_apb_i2c. The DW_apb_i2c component is an AMBA 2.0-compliant Advanced Peripheral Bus (APB) slave device and is part of the family of DesignWare Synthesizable Components.

1.1 DesignWare System Overview

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

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

Figure 1-1 Example of DW_apb_i2c in a Complete System



You can connect, configure, synthesize, and verify the DW_apb_i2c within a DesignWare subsystem using coreAssembler, documentation for which is available on the web in the *coreAssembler User Guide*.

If you want to configure, synthesize, and verify a single component such as the DW_apb_i2c component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

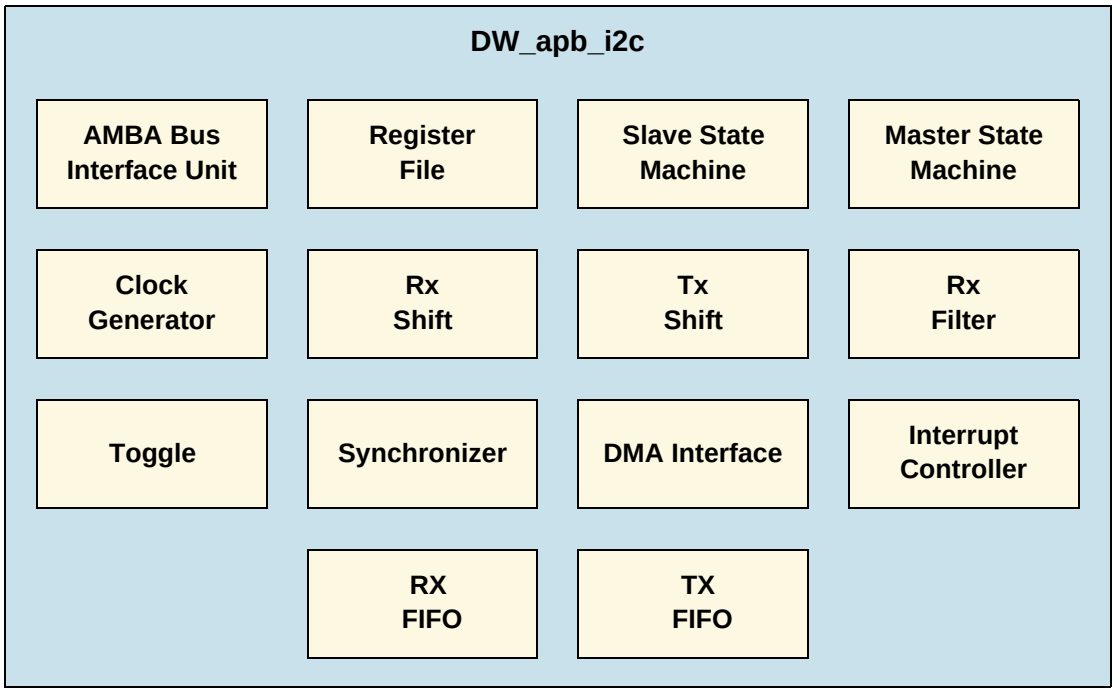
1.2 General Product Description

The DW_apb_i2c is a configurable, synthesizable, and programmable control bus that provides support for the communications link between integrated circuits in a system. It is a simple two-wire bus with a software-defined protocol for system control, which is used in temperature sensors and voltage level translators to EEPROMs, general-purpose I/O, A/D and D/A converters, CODECs, and many types of microprocessors.

1.2.1 DW_apb_i2c Block Diagram

Figure 1-2 illustrates a simple block diagram of DW_apb_i2c. For a more detailed block diagram and description of the component, see "Functional Description" on page 25.

Figure 1-2 Block Diagram of DW_apb_i2c



1.3 Features

DW_apb_i2c has the following features:

1.3.1 I²C Features

- Two-wire I²C serial interface – consists of a serial data line (SDA) and a serial clock (SCL)
- Three speeds:
  - Standard mode (0 to 100 Kb / s)
  - Fast mode (≤ 400 Kb / s) or fast mode plus (≤ 1000 Kb / s)[1]
  - High-speed mode (≤ 3.4 Mb / s)
- Clock synchronization
- Master OR slave I²C operation
- 7- or 10-bit addressing
- 7- or 10-bit combined format transfers
- Bulk transmit mode
- Ignores CBUS addresses (an older ancestor of I²C that used to share the I²C bus)
- Transmit and receive buffers
- Interrupt or polled-mode operation
- Handles Bit and Byte waiting at all bus speeds
- Simple software interface consistent with DesignWare APB peripherals
- Component parameters for configurable software driver support
- DMA handshaking interface compatible with the DW_ahb_dmac handshaking interface
- Programmable SDA hold time (tHD;DAT)
- Bus clear feature
- Device ID feature
- SMBus / PMBus support
- SMBus Slave detects and responds to ARP commands.
- Ultra-Fast mode support
- UDID feature support

The DW_apb_i2c requires external hardware components as support in order to be compliant in an I²C system. The descriptions are detailed later in this document.

1. In this document, references to fast mode also apply to fast mode plus, unless specifically stated otherwise.

It must also be noted that the DW_apb_i2c should only be operated either as (but not both):

- A master in an I^2C system and programmed only as a Master; OR
- A slave in an I^2C system and programmed only as a Slave.

1.3.2 DesignWare APB Slave Interface

- Support for APB data bus widths of 8, 16, and 32 bits
- Source code for this component is available on a per-project basis as a DesignWare Core; contact your local sales office for the details.

1.4 Standards Compliance

The DW_apb_i2c component conforms to the *AMBA Specification, Revision 2.0* from Arm®. Readers are assumed to be familiar with this specification.

The DW_apb_i2c is designed for the following specifications:

- *I2C Bus Specification, Version 6.0*, dated April 2014
- *SMBus specification Version 3.0*, dated January 2015
- *PMBus Specification Version 1.2*, dated September 2010

1.5 Verification Environment Overview

The DW_apb_i2c includes an extensive verification environment, which sets up and invokes your selected simulation tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation.

The "Verification" on page 317 chapter discusses the specific procedures for verifying the DW_apb_i2c.

1.6 Licenses

Before you begin using the DW_apb_i2c, you must have a valid license. For more information, see the "Licenses" section in *DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI Installation Guide*.

1.7 Where To Go From Here

At this point, you may want to get started working with the DW_apb_i2c component within a subsystem or by itself. Synopsys provides several tools within its coreTools suite of products for the purposes of configuration, synthesis, and verification of single or multiple synthesizable IP components—coreConsultant and coreAssembler. For information on the different coreTools, see *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_apb_i2c component, see "Overview of the coreConsultant Configuration and Integration Process" in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

For more information about implementing your DW_apb_i2c component within a DesignWare subsystem using coreAssembler, see "Overview of the coreAssembler Configuration and Integration Process" in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

2 Functional Description

This chapter describes the functional behavior of DW_apb_i2c in more detail. Following topics are covered in tis chapter:

- "Overview" on page 26
- "I²C Terminology" on page 28
- "I²C Behavior" on page 30
- "I²C Protocols" on page 31
- "Tx FIFO Management and START, STOP and RESTART Generation" on page 36
- "Multiple Master Arbitration" on page 41
- "Clock Synchronization" on page 43
- "Operation Modes" on page 43
- "Spike Suppression" on page 51
- "Fast Mode Plus Operation" on page 53
- "Bus Clear Feature" on page 53
- "Device ID" on page 54
- "Ultra-Fast Speed Mode" on page 55"SMBus/PMBus" on page 56
- "IC_CLK Frequency Configuration" on page 66
- "SDA Hold Time" on page 77
- "DMA Controller Interface" on page 80
- "APB Interface" on page 90
- "I/O Connections" on page 91
- "DW_apb_i2c Registers" on page 92
- "UDID Feature" on page 94

2.1 Overview

The I^2C bus is a two-wire serial interface, consisting of a serial data line (SDA) and a serial clock (SCL). These wires carry information between the devices connected to the bus. Each device is recognized by a unique address and can operate as either a “transmitter” or “receiver,” depending on the function of the device. Devices can also be considered as masters or slaves when performing data transfers. A master is a device that initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

Note The DW_apb_i2c must only be programmed to operate in either master OR slave mode only. Operating as a master and slave simultaneously is not supported.

The DW_apb_i2c module can operate in standard mode (with data rates 0 to 100 Kb/s), fast mode (with data rates less than or equal to 400 Kb/s), fast mode plus (with data rates less than or equal to 1000 Kb/s), high speed mode (with data rates less than or equal to 3.4 Mb/s), and Ultra-Fast Speed Mode (with data rates less than or equal to 5 Mb/s).

Note In this document, references to fast mode also apply to fast mode plus, unless specifically stated otherwise.

The DW_apb_i2c can communicate with devices only of these modes as long as they are attached to the bus. Additionally, high speed mode and fast mode devices are downward compatible. For instance, high speed mode devices can communicate with fast mode and standard mode devices in a mixed-speed bus system; fast mode devices can communicate with standard mode devices in 0 to 100 Kb/s I^2C bus system. However:

1. Standard mode devices are not upward compatible and should not be incorporated in a fast-mode I^2C bus system as they cannot follow the higher transfer rate and unpredictable states would occur.
2. Ultra-Fast mode devices are not downward compatible and should not be incorporated in traditional I2C speeds (High Speed, Fast/Fast Mode Plus speed, Standard mode speed) as Ultra-Fast mode follows the higher transfer rate (up to 5Mb/s) with only write transfers and there is no acknowledgment from the slave.

An example of high speed mode devices are LCD displays, high-bit count ADCs, and high capacity EEPROMs. These devices typically need to transfer large amounts of data. Most maintenance and control applications, the common use for the I²C bus, typically operate at 100 kHz (in standard and fast modes).

An example of Ultra-Fast speed mode devices are LED controllers and other devices that do not need feedback. These devices typically need to transfer large amounts of data greater than 1Mhz.

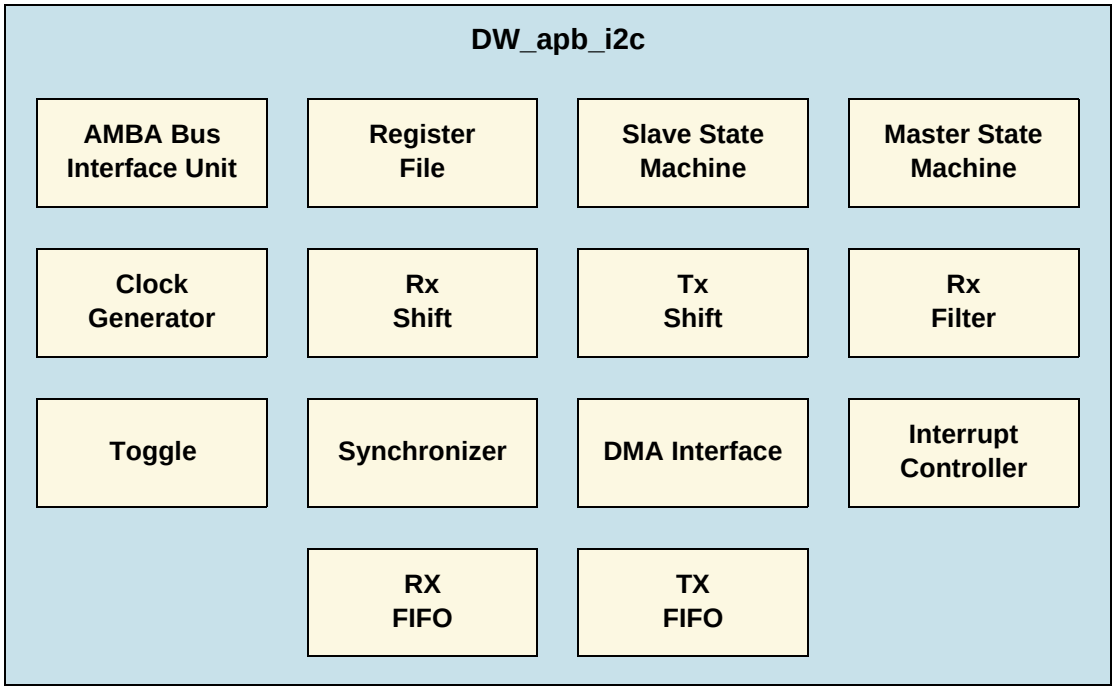
Any DW_apb_i2c device can be attached to an I²C-bus and every device can talk with any master, passing information back and forth. There needs to be at least one master (such as a microcontroller or DSP) on the bus but there can be multiple masters, which require them to arbitrate for ownership. Multiple masters and arbitration are explained later in this chapter.

The DW_apb_i2c also supports SMBus and PMBus protocols for System Management and Power management.

Note In this databook, any reference to SMBus implicitly refers to PMBus also and vice versa.

The DW_apb_i2c is made up of an AMBA APB slave interface, an I^2C interface, and FIFO logic to maintain coherency between the two interfaces. A simplified block diagram of the component is illustrated in Figure 2-1.

Figure 2-1 DW_apb_i2c Block Diagram



The following define the file names and functions of the blocks in Figure 2-1:

- AMBA Bus Interface Unit—DW_apb_i2c_biu.v—Takes the APB interface signals and translates them into a common generic interface that allows the register file to be bus protocol-agnostic.
- Register File—DW_apb_i2c_regfile—Contains configuration registers and is the interface with software.
- Slave State Machine—DW_apb_i2c_slvfsm—Follows the protocol for a slave and monitors bus for address match.
- Master State Machine—DW_apb_i2c_mstfsm—Generates the I^2C protocol for the master transfers.
- Clock Generator—DW_apb_i2c_clk_gen.v—Calculates the required timing to do the following:
  - Generate the SCL clock when configured as a master
  - Check for bus idle
  - Generate a START and a STOP
  - Setup the data and hold the data

- Rx Shift—DW_apb_i2c_rx_shift—Takes data into the design and extracts it in byte format.
- Tx Shift—DW_apb_i2c_tx_shift—Presents data supplied by CPU for transfer on the I^2C bus.
- Rx Filter—DW_apb_i2c_rx_filter—Detects the events in the bus; for example, start, stop and arbitration lost.
- Toggle—DW_apb_i2c_toggle—Generates pulses on both sides and toggles to transfer signals across clock domains.
- Synchronizer—DW_apb_i2c_sync—Transfers signals from one clock domain to another.
- DMA Interface—DW_apb_i2c_dma—Generates the handshaking signals to the central DMA controller in order to automate the data transfer without CPU intervention.
- Interrupt Controller—DW_apb_i2c_intctl—Generates the raw interrupt and interrupt flags, allowing them to be set and cleared.
- RX FIFO / TX FIFO—DW_apb_i2c_fifo—Holds the RX FIFO and TX FIFO register banks and controllers, along with their status levels.

Note

If PCLK and IC_CLK are asynchronous (IC_CLK_TYPE=ASYNC) then the following condition must be met for DW_apb_i2c to function properly:

- When IC_HAS_ASYNC_FIFO = 0,

  (SCL_LOW_COUNT*ic_clk_period) > (3*pclk_period + 3*ic_clk_period)))

  Where,
  SCL_LOW_COUNT Specifies the low count value in terms of ic_clk for the respective speed mode.
- When IC_HAS_ASYNC_FIFO = 1,

  pclk_period < (9 * scl_period)/2

  Where,
  pclk_period Specifies the clock period of the application clock.
  scl_period Specifies the SCL period.

2.2 I^2C Terminology

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

2.2.1 I^2C Bus Terms

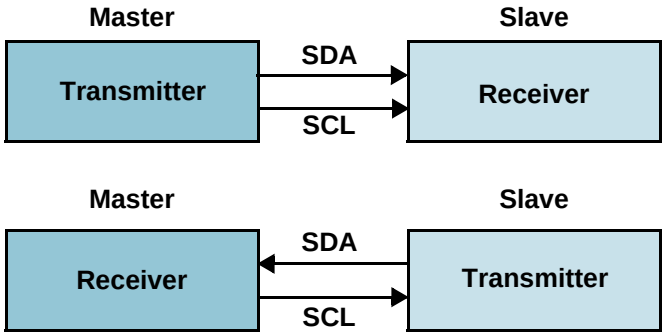
The following terms relate to how the role of the I^2C device and how it interacts with other I^2C devices on the bus.

- **Transmitter** – the device that sends data to the bus. A transmitter can either be a device that initiates the data transmission to the bus (a *master-transmitter*) or responds to a request from the master to send data to the bus (a *slave-transmitter*).

- **Receiver** – the device that receives data from the bus. A receiver can either be a device that receives data on its own request (a *master-receiver*) or in response to a request from the master (a *slave-receiver*).
- **Master** -– the component that initializes a transfer (START command), generates the clock (SCL) signal and terminates the transfer (STOP command). A master can be either a transmitter or a receiver.
- **Slave** – the device addressed by the master. A slave can be either receiver or transmitter.

  These concepts are illustrated in Figure 2-2.

Figure 2-2 Master/Slave and Transmitter/Receiver Relationships



- **Multi-master** – the ability for more than one master to co-exist on the bus at the same time without collision or data loss.
- **Arbitration** – the predefined procedure that authorizes only one master at a time to take control of the bus. For more information about this behavior, see "Multiple Master Arbitration" on page 41.
- **Synchronization** – the predefined procedure that synchronizes the clock signals provided by two or more masters. For more information about this feature, see "Clock Synchronization" on page 43.
- **SDA** – data signal line (Serial DAta)
- **SCL** – clock signal line (Serial CLock)

2.2.2 Bus Transfer Terms

The following terms are specific to data transfers that occur to / from the I^2C bus.

- **START** (**RESTART**) – data transfer begins with a START or RESTART condition. The level of the SDA data line changes from high to low, while the SCL clock line remains high. When this occurs, the bus becomes busy.

Note START and RESTART conditions are functionally identical.

- **STOP** – data transfer is terminated by a STOP condition. This occurs when the level on the SDA data line passes from the low state to the high state, while the SCL clock line remains high. When the data transfer has been terminated, the bus is free or idle once again. The bus stays busy if a RESTART is generated instead of a STOP condition.

2.3 I²C Behavior

The DW_apb_i2c can be controlled through software to be either:

- An I²C master only, communicating with other I²C slaves; OR
- An I²C slave only, communicating with one more I²C masters.

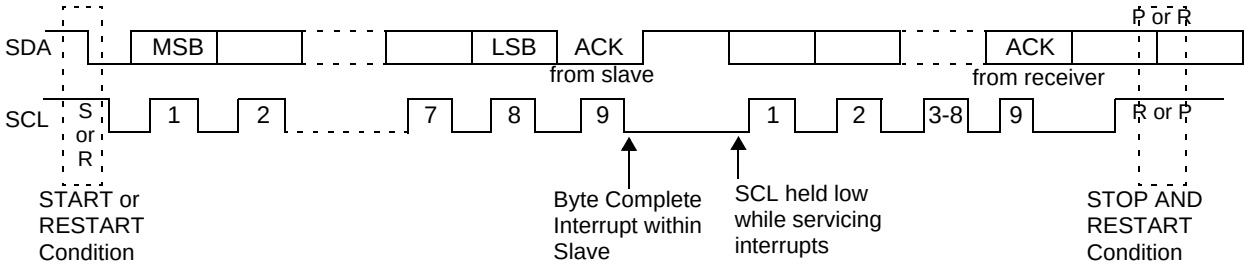
The master is responsible for generating the clock and controlling the transfer of data. The slave is responsible for either transmitting or receiving data to/from the master. The acknowledgement of data is sent by the device that is receiving data, which can be either a master or a slave. As mentioned previously, the I²C protocol also allows multiple masters to reside on the I²C bus and uses an arbitration procedure to determine bus ownership.

Each slave has a unique address that is determined by the system designer. When a master wants to communicate with a slave, the master transmits a START/RESTART condition that is then followed by the slave's address and a control bit (R/W) to determine if the master wants to transmit data or receive data from the slave. The slave then sends an acknowledge (ACK) pulse after the address.

If the master (master-transmitter) is writing to the slave (slave-receiver), the receiver gets one byte of data. This transaction continues until the master terminates the transmission with a STOP condition. If the master is reading from a slave (master-receiver), the slave transmits (slave-transmitter) a byte of data to the master, and the master then acknowledges the transaction with the ACK pulse. This transaction continues until the master terminates the transmission by not acknowledging (NACK) the transaction after the last byte is received, and then the master issues a STOP condition or addresses another slave after issuing a RESTART condition. This behavior is illustrated in Figure 2-3.

In Ultra-Fast Speed Mode, the master can issue only the write transfers to the slaves with always not acknowledging (NACK) from the slaves. Read transfers are not allowed in this mode.

Figure 2-3 Data transfer on the I2C Bus



The DW_apb_i2c is a synchronous serial interface. The SDA line is a bidirectional signal and changes only while the SCL line is low, except for STOP, START, and RESTART conditions. The output drivers are open-drain or open-collector to perform wire-AND functions on the bus. The maximum number of devices on the bus is limited by only the maximum capacitance specification of 400 pF. Data is transmitted in byte packages.

The I²C protocols implemented in DW_apb_i2c are described in more details in "I²C Protocols" on page 31.

2.3.1 START and STOP Generation

When operating as an I²C master, putting data into the transmit FIFO causes the DW_apb_i2c to generate a START condition on the I²C bus. If the IC_EMPTYFIFO_HOLD_MASTER_EN parameteris set to 0,

allowing the transmit FIFO to empty causes the DW_apb_i2c to generate a STOP condition on the I^2C bus. If IC_EMPTYFIFO_HOLD_MASTER_EN is set to 1, then writing a 1 to IC_DATA_CMD[9] causes the DW_apb_i2c to generate a STOP condition on the I^2C bus; a STOP condition is not issued if this bit is not set, even if the transmit FIFO is empty.

When operating as a slave, the DW_apb_i2c does not generate START and STOP conditions, as per the protocol. However, if a read request is made to the DW_apb_i2c, it holds the SCL line low until read data has been supplied to it. This stalls the I^2C bus until read data is provided to the slave DW_apb_i2c, or the DW_apb_i2c slave is disabled by writing a 0 to bit 0 of the IC_ENABLE register.

2.3.2 Combined Formats

The DW_apb_i2c supports mixed read and write combined format transactions in both 7-bit and 10-bit addressing modes.

The DW_apb_i2c does not support mixed address and mixed address format—that is, a 7-bit address transaction followed by a 10-bit address transaction or vice versa—combined format transactions.

To initiate combined format transfers, IC_CON.IC_RESTART_EN should be set to 1. With this value set and operating as a master, when the DW_apb_i2c completes an I2C transfer, it checks the transmit FIFO and executes the next transfer. If the direction of this transfer differs from the previous transfer, the combined format is used to issue the transfer. If the transmit FIFO is empty when the current I2C transfer completes—depending on the value of IC_EMPTYFIFO_HOLD_MASTER_EN:

- Either a STOP is issued or,
- IC_DATA_CMD[9] is checked *and*:
  - If set to 1, a STOP bit is issued.
  - If set to 0, the SCL is held low until the next command is written to the transmit FIFO.

For more details, see "Tx FIFO Management and START, STOP and RESTART Generation" on page 36.

Mixed write and read transactions in both 7-bit and 10-bit addressing modes are not applicable for Ultra-Fast Mode (IC_ULTRA_FAST_MODE=1) as read transfers are not supported in Ultra-Fast Mode.

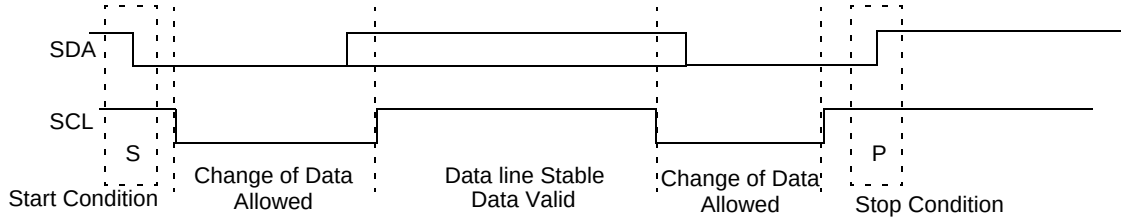
2.4 I^2C Protocols

The DW_apb_i2c has the protocols discussed in this section.

2.4.1 START and STOP Conditions

When the bus is idle, both the SCL and SDA signals are pulled high through external pull-up resistors on the bus. When the master wants to start a transmission on the bus, the master issues a START condition. This is defined to be a high-to-low transition of the SDA signal while SCL is 1. When the master wants to terminate the transmission, the master issues a STOP condition. This is defined to be a low-to-high transition of the SDA line while SCL is 1. Figure 2-4 shows the timing of the START and STOP conditions. When data is being transmitted on the bus, the SDA line must be stable when SCL is 1.

Figure 2-4 START and STOP Condition



Note The signal transitions for the START/STOP conditions, as depicted in Figure 2-4, reflect those observed at the output signals of the Master driving the I^2C bus. Care should be taken when observing the SDA/SCL signals at the input signals of the Slave(s), because unequal line delays may result in an incorrect SDA/SCL timing relationship.

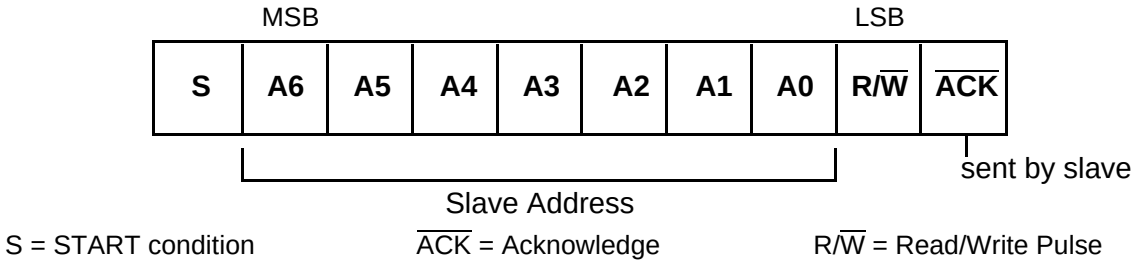
2.4.2 Addressing Slave Protocol

There are two address formats: the 7-bit address format and the 10-bit address format.

2.4.2.1 7-bit Address Format

During the 7-bit address format, the first seven bits (bits 7:1) of the first byte set the slave address and the LSB bit (bit 0) is the R / W bit as shown in Figure 2-5. When bit 0 (R / W) is set to 0, the master writes to the slave. When bit 0 (R / W) is set to 1, the master reads from the slave.

Figure 2-5 7-bit Address Format



2.4.2.2 10-bit Address Format

During 10-bit addressing, two bytes are transferred to set the 10-bit address. The transfer of the first byte contains the following bit definition. The first five bits (bits 7:3) notify the slaves that this is a 10-bit transfer followed by the next two bits (bits 2:1), which set the slaves address bits 9:8, and the LSB bit (bit 0) is the

R/W bit. The second byte transferred sets bits 7:0 of the slave address. Figure 2-6 shows the 10-bit address format.

Figure 2-6 10-bit Address Format

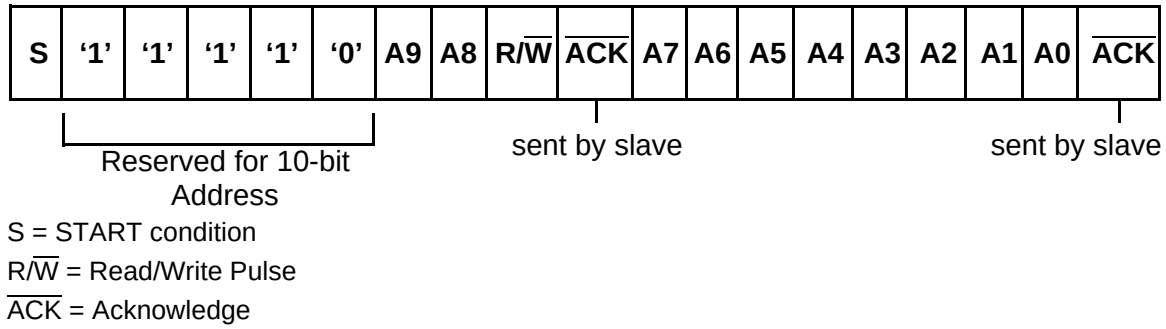


Table 2-1 defines the special purpose and reserved first byte addresses.

Table 2-1 I^2C/SMBus Definition of Bits in First Byte

Slave Address	R/W Bit	Description
0000 000	0	General Call Address. DW_apb_i2c places the data in the receive buffer and issues a General Call interrupt.
0000 000	1	START byte. For more details, see "START BYTE Transfer Protocol" on page 35.
0000 001	X	CBUS address. DW_apb_i2c ignores these accesses.
0000 010	X	Reserved.
0000 011	X	Reserved.
0000 1XX	X	High-speed master code (for more information, see "Multiple Master Arbitration" on page 41).
1111 1XX	X	Reserved.
1111 0XX	X	10-bit slave addressing.
0001 000	X	SMbus Host
0001 100	X	SMBus Alert Response Address
1100 001	X	SMBus Device Default Address

DW_apb_i2c does not restrict you from using these reserved addresses. However, if you use these reserved addresses, you may run into incompatibilities with other I^2C components.

2.4.3 Transmitting and Receiving Protocol

The master can initiate data transmission and reception to/from the bus, acting as either a master-transmitter or master-receiver. A slave responds to requests from the master to either transmit data or receive data to/from the bus, acting as either a slave-transmitter or slave-receiver, respectively.

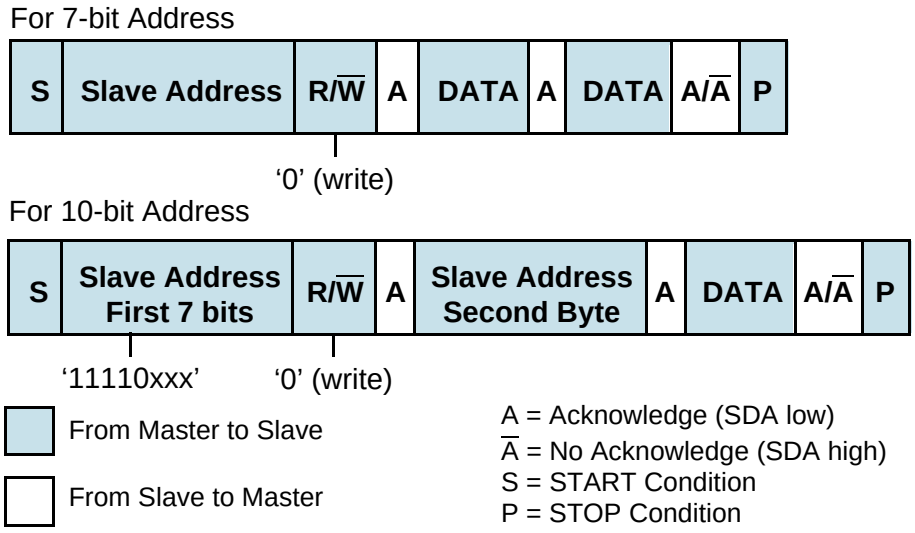
2.4.3.1 Master-Transmitter and Slave-Receiver

All data is transmitted in byte format, with no limit on the number of bytes transferred per data transfer. After the master sends the address and R/W bit or the master transmits a byte of data to the slave, the slave-receiver must respond with the acknowledge signal (ACK). When a slave-receiver does not respond with an ACK pulse, the master aborts the transfer by issuing a STOP condition. The slave must leave the SDA line high so that the master can abort the transfer.

If the master-transmitter is transmitting data as shown in Figure 2-7, then the slave-receiver responds to the master-transmitter with an acknowledge pulse after every byte of data is received.

Note In Ultra-Fast Mode, the slave-receiver always responds with the No Acknowledge signal (NACK) for the Address and the write data from the Master.

Figure 2-7 Master-Transmitter Protocol

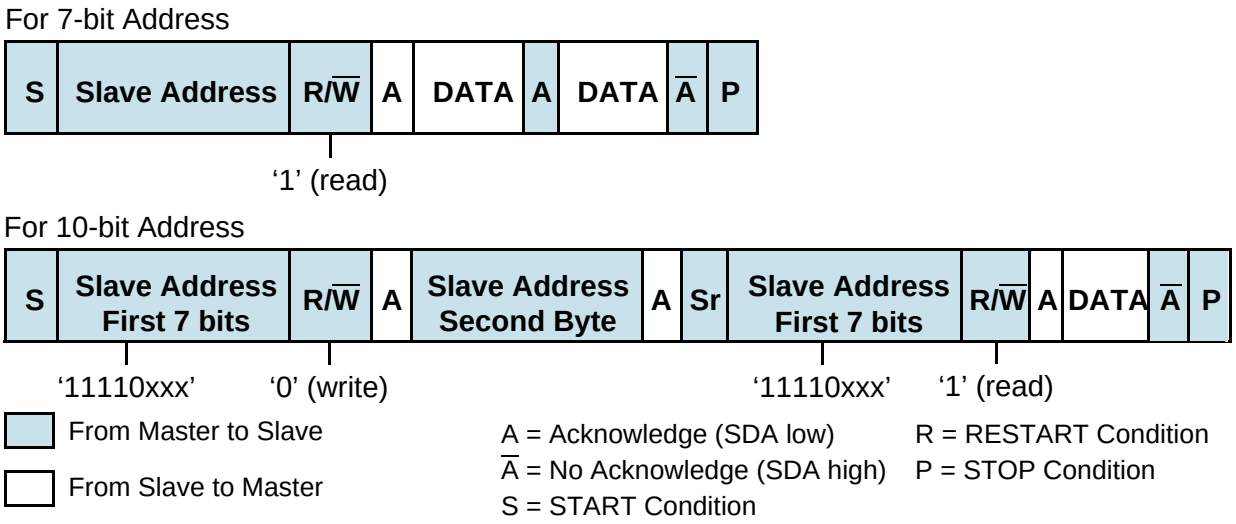


2.4.3.2 Master-Receiver and Slave-Transmitter

If the master is receiving data as shown in Figure 2-8, then the master responds to the slave-transmitter with an acknowledge pulse after a byte of data has been received, except for the last byte. This is the way the

master-receiver notifies the slave-transmitter that this is the last byte. The slave-transmitter relinquishes the SDA line after detecting the No Acknowledge (NACK) so that the master can issue a STOP condition.

Figure 2-8 Master-Receiver Protocol



When a master does not want to relinquish the bus with a STOP condition, the master can issue a RESTART condition. This is identical to a START condition except it occurs after the ACK pulse. Operating in master mode, the DW_apb_i2c can then communicate with the same slave using a transfer of a different direction. For a description of the combined format transactions that the DW_apb_i2c supports, see "Combined Formats" on page 31.

Note The DW_apb_i2c must be completely disabled—if I2C_DYNAMIC_TAR_UPDATE = 0—or inactive on the serial port—if I2C_DYNAMIC_TAR_UPDATE = 1—before the target slave address register (IC_TAR) can be reprogrammed.

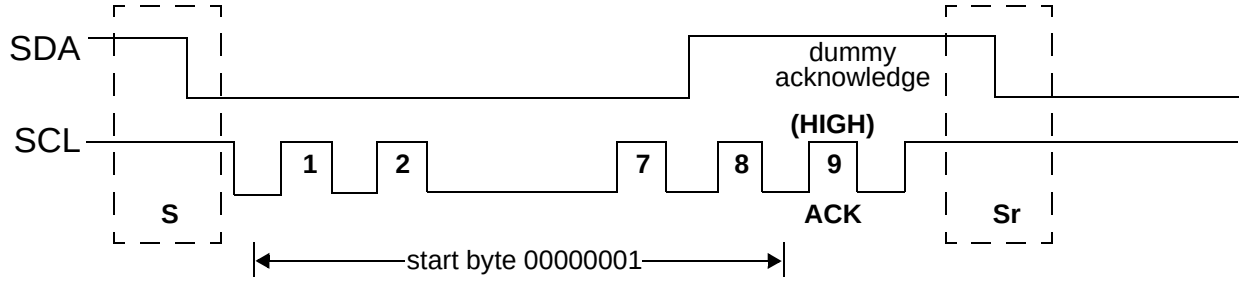
Note The DW_apb_i2c must be completely disabled—if I2C_DYNAMIC_TAR_UPDATE = 0—or inactive on the serial port—if I2C_DYNAMIC_TAR_UPDATE = 1—before the target slave address register (IC_TAR) can be reprogrammed.

2.4.4 START BYTE Transfer Protocol

The START BYTE transfer protocol is set up for systems that do not have an on-board dedicated I^2C hardware module. When the DW_apb_i2c is addressed as a slave, it always samples the I^2C bus at the highest speed supported so that it never requires a START BYTE transfer. However, when DW_apb_i2c is a master, it supports the generation of START BYTE transfers at the beginning of every transfer in case a slave device requires it.

This protocol consists of seven zeros being transmitted followed by a 1, as illustrated in Figure 2-9. This allows the processor that is polling the bus to under-sample the address phase until 0 is detected. Once the microcontroller detects a 0, it switches from the under sampling rate to the correct rate of the master.

Figure 2-9 START BYTE Transfer



The START BYTE procedure is as follows:

1. Master generates a START condition.
2. Master transmits the START byte (0000 0001).
3. Master transmits the ACK clock pulse. (Present only to conform with the byte handling format used on the bus)
4. No slave sets the ACK signal to 0.
5. Master generates a RESTART (R) condition.

A hardware receiver does not respond to the START BYTE because it is a reserved address and resets after the RESTART condition is generated.

2.5 Tx FIFO Management and START, STOP and RESTART Generation

When operating as a master, the DW_apb_i2c component supports two modes of Tx FIFO management. You use the IC_EMPTYFIFO_HOLD_MASTER_EN parameter to select between these two modes:

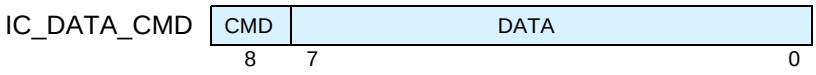
- IC_EMPTYFIFO_HOLD_MASTER_EN equals 0, illustrated in Figure 2-10
- IC_EMPTYFIFO_HOLD_MASTER_EN equals 1, illustrated in Figure 2-13

2.5.1 Tx FIFO Management When IC_EMPTYFIFO_HOLD_MASTER_EN = 0

When the value of IC_EMPTYFIFO_HOLD_MASTER_EN is 0, the component generates a STOP on the bus whenever the Tx FIFO becomes empty. If RESTART generation capability is enabled, the component generates a RESTART when the direction of the transfer in the Tx FIFO commands changes from Read to Write or vice-versa; if RESTART is not enabled, a STOP followed by a START is generated in this situation.

Figure 2-10 shows the bits in the IC_DATA_CMD register if IC_EMPTYFIFO_HOLD_MASTER_EN = 0.

Figure 2-10 IC_DATA_CMD Register if IC_EMPTYFIFO_HOLD_MASTER_EN = 0



DATA –Read/Write field; data retrieved from slave is read from this field; data to be sent to slave is written to this field.
CMD –Write-only field; this bit determines whether transfer to be carried out is Read (CMD=1) or Write (CMD=0)

Figure 2-11 shows a timing diagram that illustrates the behavior of the DW_apb_i2c when Tx FIFO becomes empty while operating as a master transmitter when IC_EMPTYFIFO_HOLD_MASTER_EN=0.

Figure 2-11 Master Transmitter — Tx FIFO Becomes Empty If IC_EMPTYFIFO_HOLD_MASTER_EN = 0

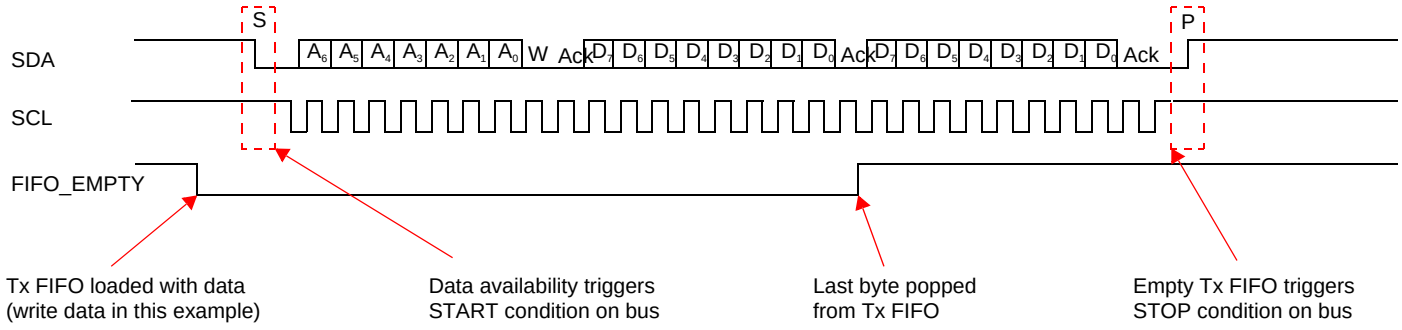
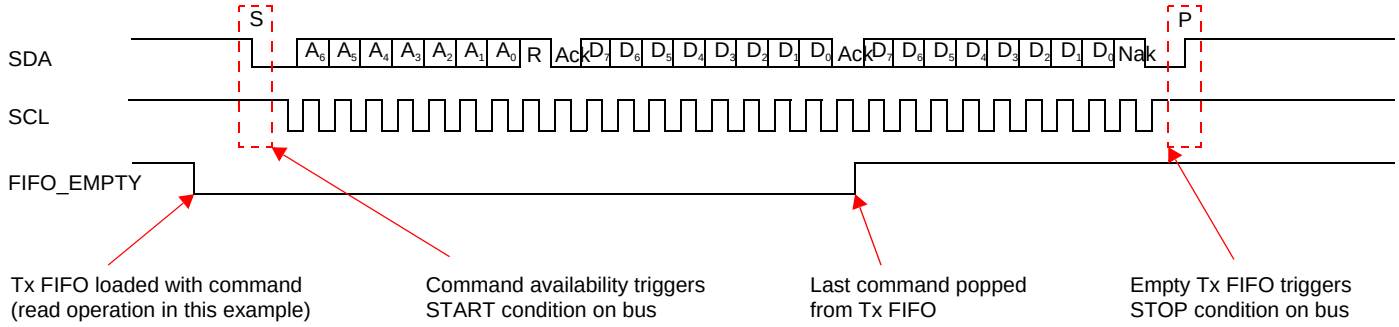


Figure 2-12 shows a timing diagram that illustrates the behavior of the DW_apb_i2c when Tx FIFO becomes empty while operating as a master receiver when IC_EMPTYFIFO_HOLD_MASTER_EN=0.

Figure 2-12 Master Receiver — Tx FIFO Becomes Empty If IC_EMPTYFIFO_HOLD_MASTER_EN = 0



Note

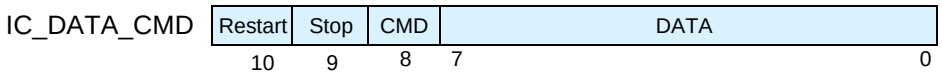
When IC_EMPTYFIFO_HOLD_MASTER_EN = 0, if the TX_EMPTY_CTRL bit of the IC_CON register is set to 1 and the transmit threshold (TX_THLD) is set to 0, then one I2C frame (consisting of multiple commands) always breaks into multiple frames based on the number of commands in each frame. This is because DW_apb_i2c issues a TX_EMPTY interrupt after the end of each data (TX_EMPTY_CTRL=1) is sent on the line and the data (TX_THLD=0) is pushed based on TX_EMPTY interrupt.

2.5.2 Tx FIFO Management When IC_EMPTYFIFO_HOLD_MASTER_EN = 1

When the value of IC_EMPTYFIFO_HOLD_MASTER_EN is 1, the component does not generate a STOP if the Tx FIFO becomes empty; in this situation the component holds the SCL line low, stalling the bus until a new entry is available in the Tx FIFO. A STOP condition is generated only when you specifically request it by setting bit 9 (Stop bit) of the command written to IC_DATA_CMD register.

Figure 2-13 shows the bits in the IC_DATA_CMD register if IC_EMPTYFIFO_HOLD_MASTER_EN = 1.

Figure 2-13 IC_DATA_CMD Register if IC_EMPTYFIFO_HOLD_MASTER_EN = 1



DATA –Read/Write field; data retrieved from slave is read from this field; data to be sent to slave is written to this field
CMD –Write-only field; this bit determines whether transfer to be carried out is Read (CMD=1) or Write (CMD=0)
Stop –Write-only field; this bit determines whether STOP is generated after data byte is sent or received
Restart – Write-only field; this bit determines whether RESTART (or STOP followed by START in case of restart capability is not enabled) is generated before data byte is sent of received

Figure 2-14 illustrates the behavior of the DW_apb_i2c when the Tx FIFO becomes empty while operating as a master transmitter, as well as showing the generation of a STOP condition when IC_EMPTYFIFO_HOLD_MASTER_EN=1.

Figure 2-14 Master Transmitter — Tx FIFO Empties/STOP Generation If IC_EMPTYFIFO_HOLD_MASTER_EN = 1

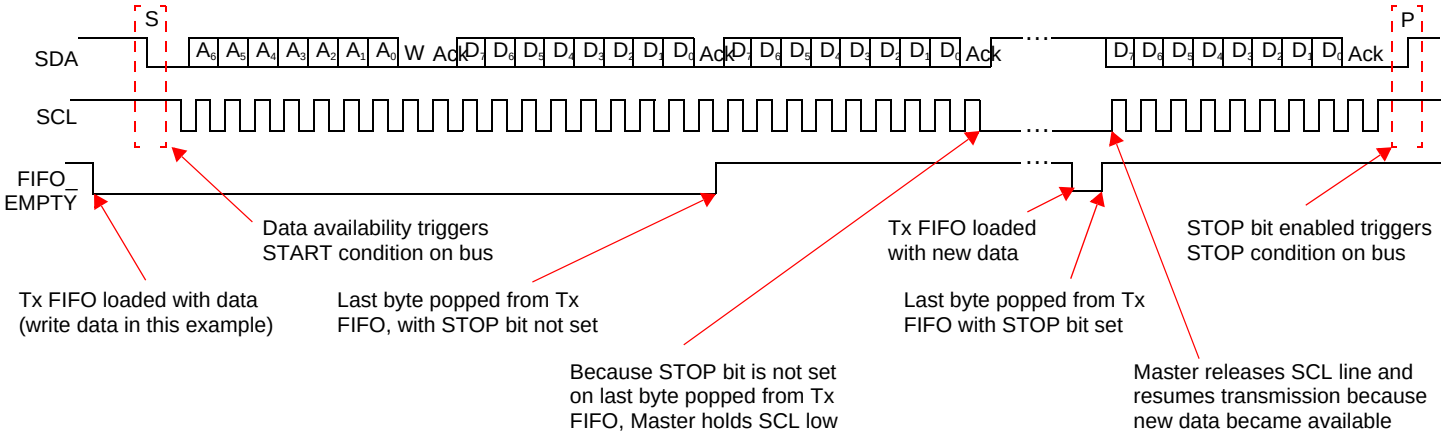


Figure 2-15 illustrates the behavior of the DW_apb_i2c when the Tx FIFO becomes empty while operating as a master receiver, as well as showing the generation of a STOP condition when IC_EMPTYFIFO_HOLD_MASTER_EN=1.

Figure 2-15 Master Receiver — Tx FIFO Empties/STOP Generation If IC_EMPTYFIFO_HOLD_MASTER_EN = 1

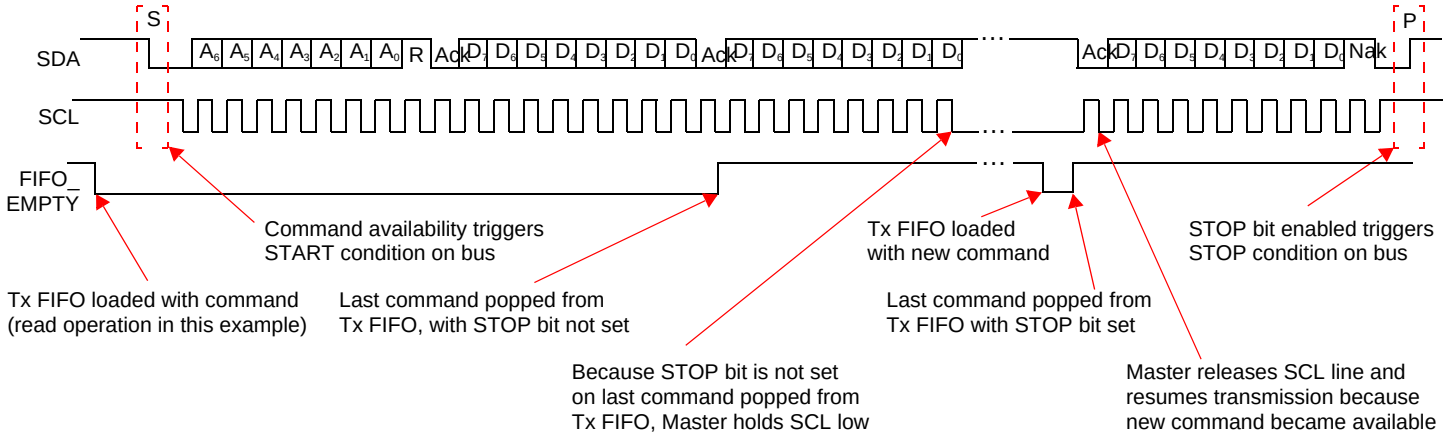


Figure 2-16 and Figure 2-17 illustrate configurations where you can control the generation of RESTART conditions on the I2C bus. If bit 10 (Restart) of the IC_DATA_CMD register is set and the restart capability is enabled (IC_RESTART_EN=1), a RESTART is generated before the data byte is written to or read from the slave. If the restart capability is not enabled a STOP followed by a START is generated in place of the RESTART. Figure 2-16 illustrates this situation during operation as a master transmitter.

Figure 2-16 Master Transmitter — Restart Bit of IC_DATA_CMD Is Set (IC_EMPTYFIFO_HOLD_MASTER_EN = 1)

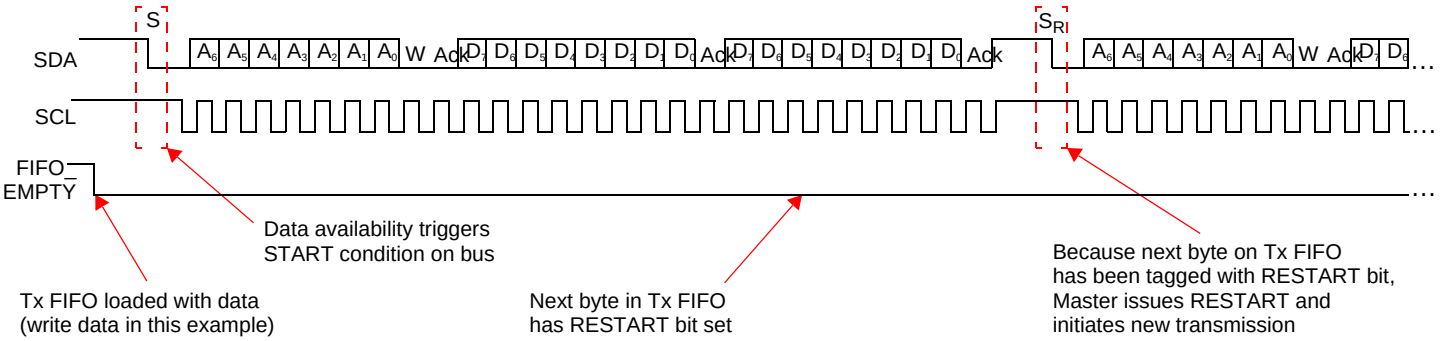


Figure 2-17 illustrates the same situation, but during operation as a master receiver.

Figure 2-17 Master Receiver — Restart Bit of IC_DATA_CMD Is Set (IC_EMPTYFIFO_HOLD_MASTER_EN = 1)

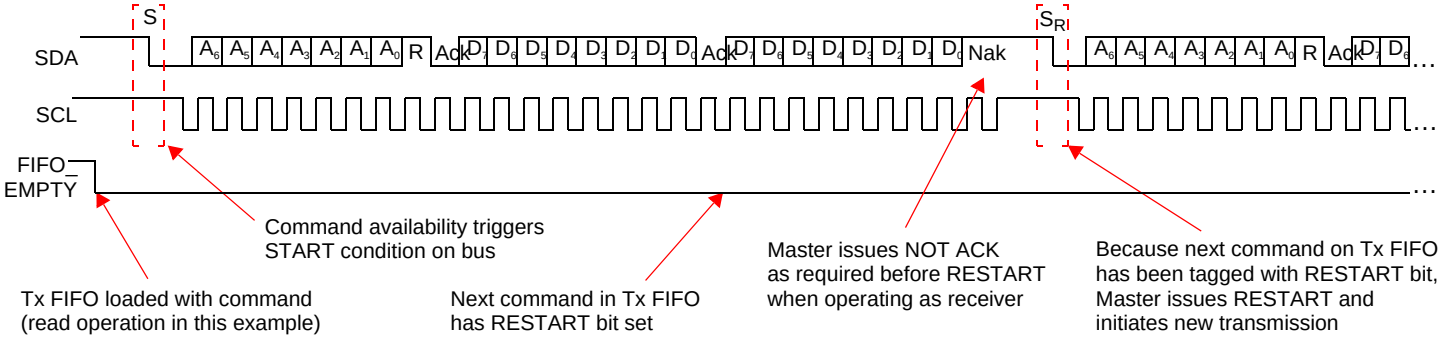


Figure 2-18 illustrates operation as a master transmitter where the Stop bit of the IC_DATA_CMD register is set and the Tx FIFO is not empty (IC_EMPTYFIFO_HOLD_MASTER_EN=1).

Figure 2-18 Master Transmitter — Stop Bit of IC_DATA_CMD Set/Tx FIFO Not Empty (IC_EMPTYFIFO_HOLD_MASTER_EN=1)

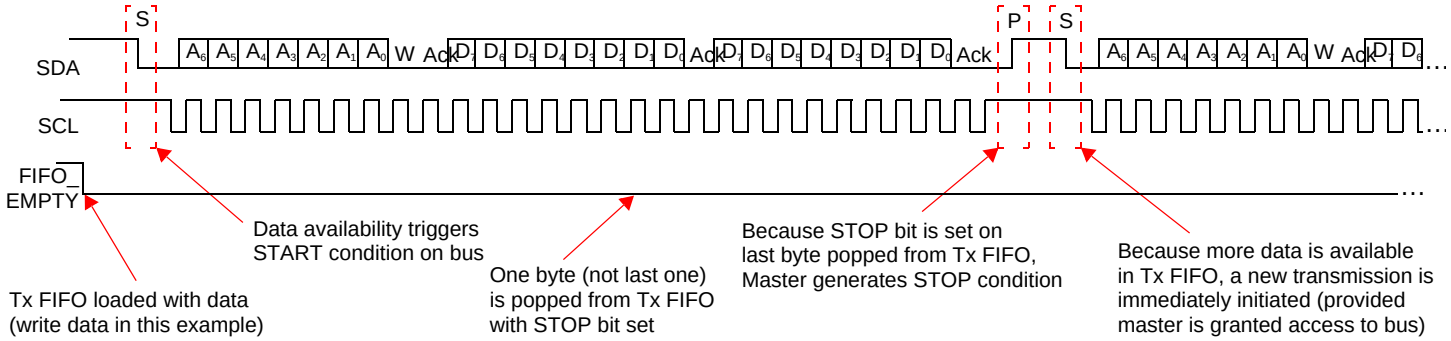


Figure 2-19 illustrates operation as a master transmitter where the first byte loaded into the Tx FIFO is allowed to go empty with the Restart bit set (IC_EMPTYFIFO_HOLD_MASTER_EN=1).

Figure 2-19 Master Transmitter — First Byte Loaded Into Tx FIFO Allowed to Empty, Restart Bit Set (IC_EMPTYFIFO_HOLD_MASTER_EN=1)

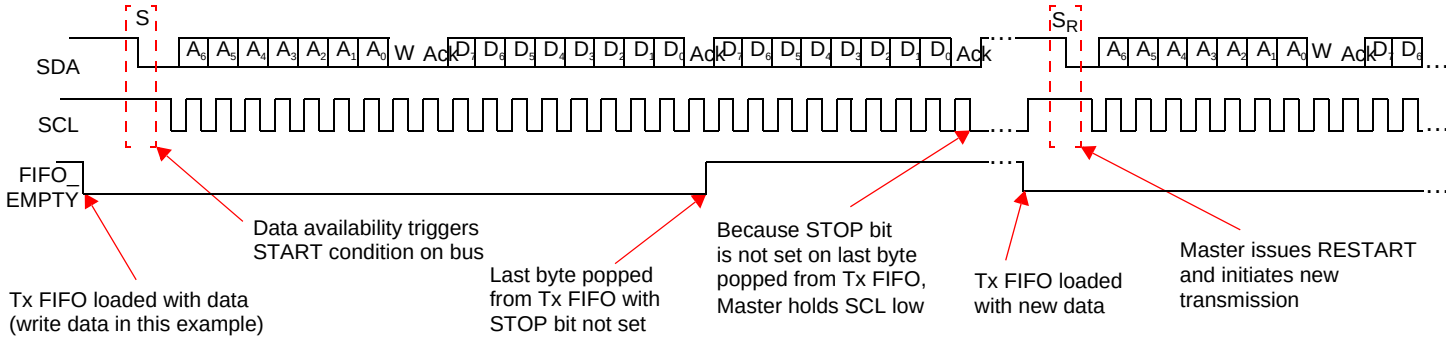


Figure 2-20 illustrates operation as a master receiver where the Stop bit of the IC_DATA_CMD register is set and the Tx FIFO is not empty (IC_EMPTYFIFO_HOLD_MASTER_EN=1).

Figure 2-20 Master Receiver — Stop Bit of IC_DATA_CMD Set/Tx FIFO Not Empty (IC_EMPTYFIFO_HOLD_MASTER_EN=1 and IC_ULTRA_FAST_MODE=0)

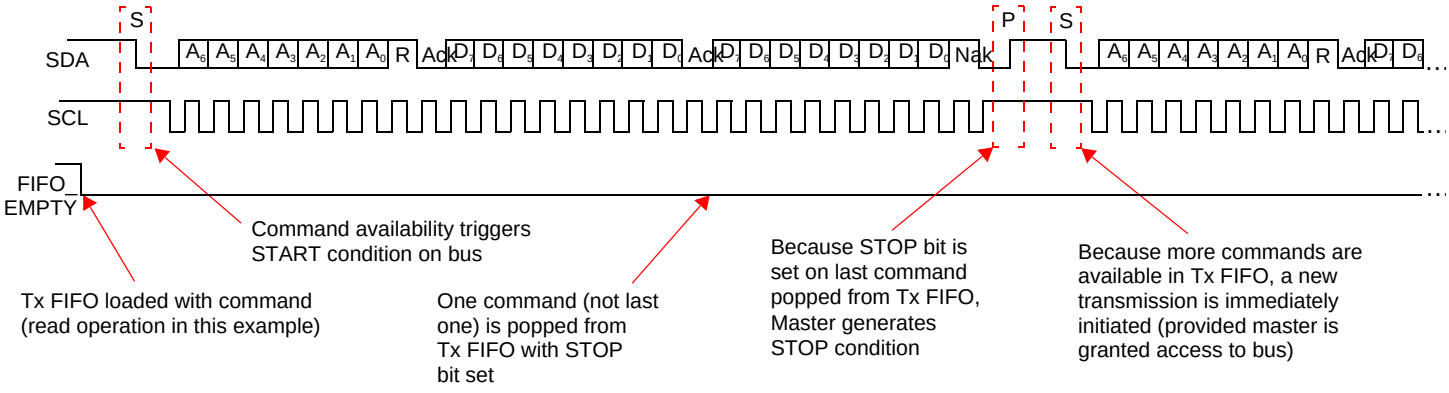
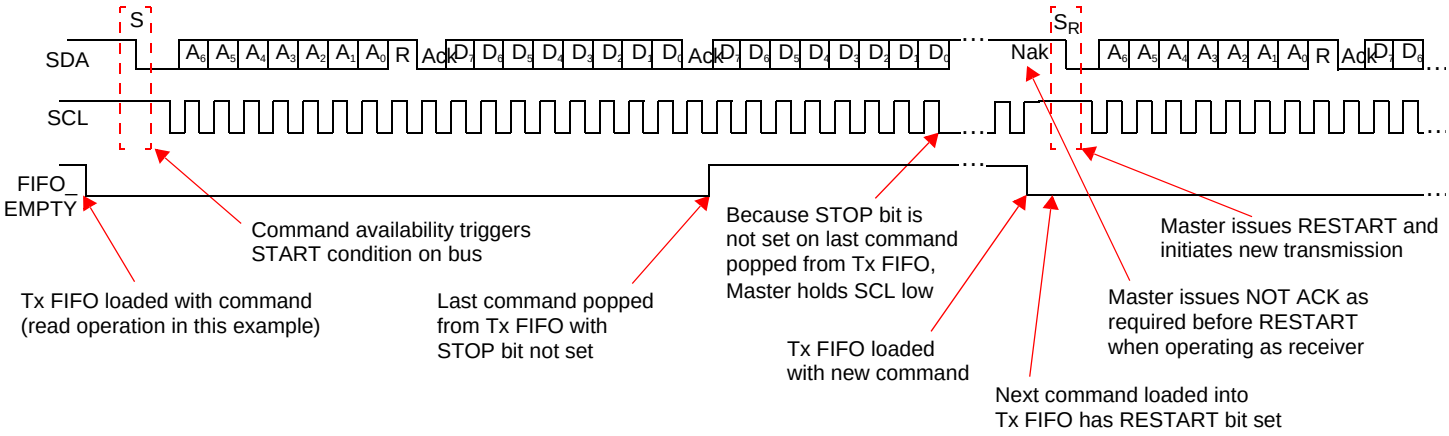


Figure 2-21 illustrates operation as a master receiver where the first command loaded after the Tx FIFO is allowed to empty and the Restart bit is set (IC_EMPTYFIFO_HOLD_MASTER_EN=1).

Figure 2-21 Master Receiver — First Command Loaded After Tx FIFO Allowed to Empty/Restart Bit Set (IC_EMPTYFIFO_HOLD_MASTER_EN=1 and IC_ULTRA_FAST_MODE=0)



2.6 Multiple Master Arbitration

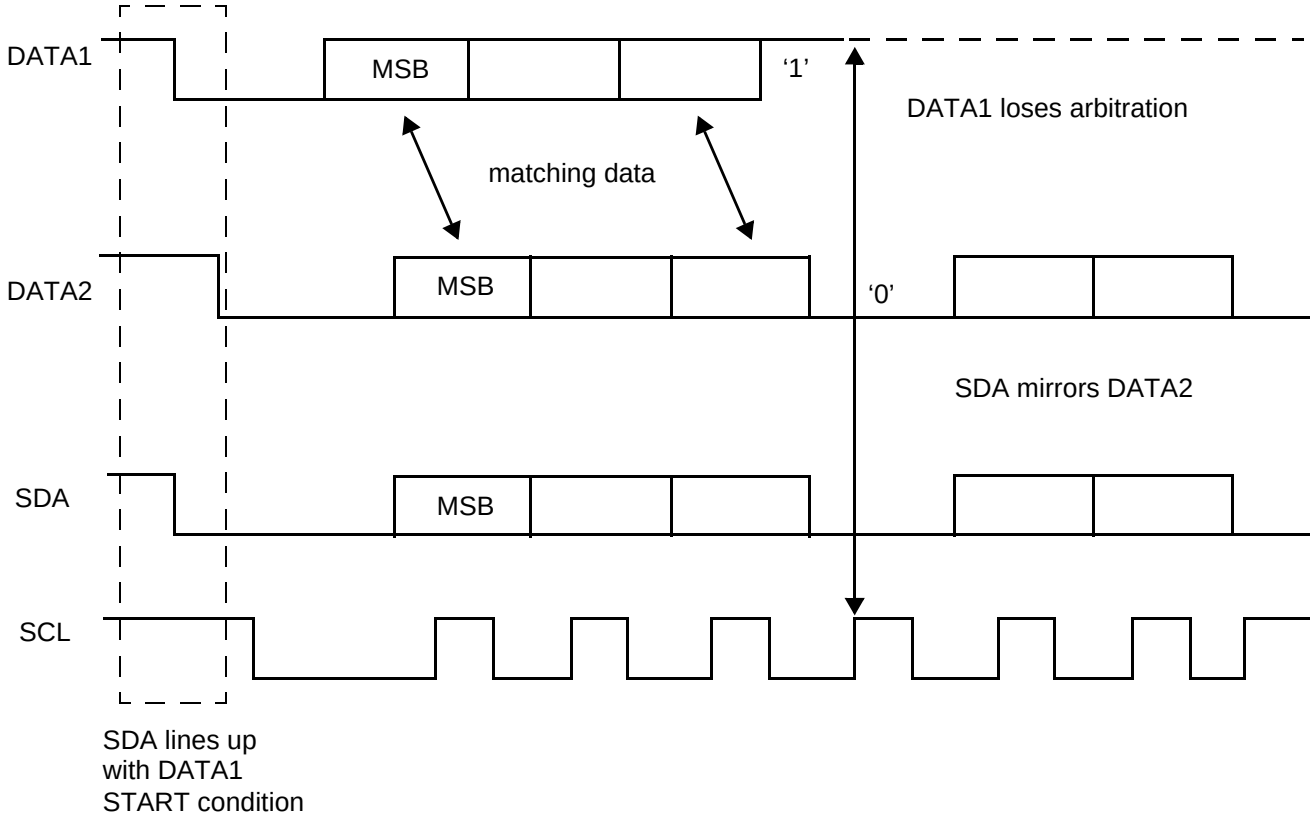
The DW_apb_i2c bus protocol allows multiple masters to reside on the same bus. If there are two masters on the same I²C-bus, there is an arbitration procedure if both try to take control of the bus at the same time by generating a START condition at the same time. Once a master (for example, a microcontroller) has control of the bus, no other master can take control until the first master sends a STOP condition and places the bus in an idle state.

Arbitration takes place on the SDA line, while the SCL line is 1. The master, which transmits a 1 while the other master transmits 0, loses arbitration and turns off its data output stage. The master that lost arbitration can continue to generate clocks until the end of the byte transfer. If both masters are addressing the same slave device, the arbitration could go into the data phase.

Upon detecting that it has lost arbitration to another master, the DW_apb_i2c stops generating SCL (ic_clk_oe).

Figure 2-22 illustrates the timing of when two masters are arbitrating on the bus.

Figure 2-22 Multiple Master Arbitration



For high-speed mode, the arbitration cannot go into the data phase because each master is programmed with a unique high-speed master code. This 8-bitcode is defined by the system designer and is set by writing to the high speed Master Mode Code Address Register, IC_HS_MADDR. Because the codes are unique, only one master can win arbitration, which occurs by the end of the transmission of the high-speed master code.

Control of the bus is determined by address or master code and data sent by competing masters, so there is no central master nor any order of priority on the bus.

Arbitration is not allowed between the following conditions:

- A RESTART condition and a data bit
- A STOP condition and a data bit
- A RESTART condition and a STOP condition

Slaves are not involved in the arbitration process.

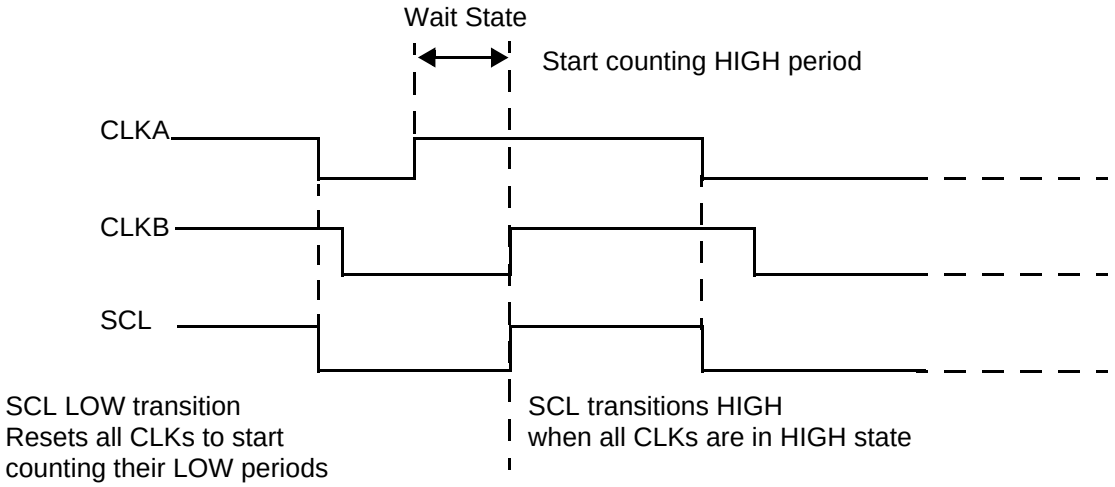
Note Multi-master arbitration is not applicable in Ultra-Fast Mode (IC_ULTRA_FAST_MODE=1) as single Master is present.

2.7 Clock Synchronization

When two or more masters try to transfer information on the bus at the same time, they must arbitrate and synchronize the SCL clock. All masters generate their own clock to transfer messages. Data is valid only during the high period of SCL clock. Clock synchronization is performed using the wired-AND connection to the SCL signal. When the master transitions the SCL clock to 0, the master starts counting the low time of the SCL clock and transitions the SCL clock signal to 1 at the beginning of the next clock period. However, if another master is holding the SCL line to 0, then the master goes into a HIGH wait state until the SCL clock line transitions to 1.

All masters then count off their high time, and the master with the shortest high time transitions the SCL line to 0. The masters then counts out their low time and the one with the longest low time forces the other master into a HIGH wait state. Therefore, a synchronized SCL clock is generated, which is illustrated in Figure 2-23. Optionally, slaves may hold the SCL line low to slow down the timing on the I^2C bus.

Figure 2-23 Multi-Master Clock Synchronization



Note Clock Synchronization is not supported in Ultra-Fast Mode (IC_ULTRA_FAST_MODE=1) as single master is present in the Ultra-Fast Mode system.

2.8 Operation Modes

This section provides information on operation modes.

Note It is important to note that the DW_apb_i2c should only be set to operate as an I^2C Master, or I^2C Slave, but not both simultaneously. This is achieved by ensuring that bit 6 (IC_SLAVE_DISABLE) and 0 (IC_MASTER_MODE) of the IC_CON register are never set to 0 and 1, respectively.

2.8.1 Slave Mode Operation

This section discusses slave mode procedures.

2.8.1.1 Initial Configuration

To use the DW_apb_i2c as a slave, perform the following steps:

1. Disable the DW_apb_i2c by writing a ‘0’ to bit 0 of the IC_ENABLE register.
2. Write to the IC_SAR register (bits 9:0) to set the slave address. This is the address to which the DW_apb_i2c responds.
3. Write to the IC_CON register to specify which type of addressing is supported (7- or 10-bit by setting bit 3). Enable the DW_apb_i2c in slave-only mode by writing a ‘0’ into bit 6 (IC_SLAVE_DISABLE) and a ‘0’ to bit 0 (MASTER_MODE).

Note Slaves and masters do not have to be programmed with the same type of addressing 7- or 10-bit address. For instance, a slave can be programmed with 7-bit addressing and a master with 10-bit addressing, and vice versa.

4. Enable the DW_apb_i2c by writing a ‘1’ in bit 0 of the IC_ENABLE register.

Note Depending on the reset values chosen, steps 2 and 3 may not be necessary because the reset values can be configured. For instance, if the device is only going to be a master, there would be no need to set the slave address because you can configure DW_apb_i2c to have the slave disabled after reset and to enable the master after reset. The values stored are static and do not need to be reprogrammed if the DW_apb_i2c is disabled.

Attention It is recommended that the DW_apb_i2c Slave be brought out of reset only when the I2C bus is IDLE. De-asserting the reset when a transfer is ongoing on the bus causes internal synchronization flip-flops used to synchronize SDA and SCL to toggle from a reset value of 1 to the actual value on the bus. This can result in SDA toggling from 1 to 0 while SCL is 1, thereby causing a false START condition to be detected by the DW_apb_i2c Slave. This scenario can also be avoided by configuring the DW_apb_i2c with IC_SLAVE_DISABLE = 1 and IC_MASTER_MODE = 1 so that the Slave interface is disabled after reset. It can then be enabled by programming IC_CON[0] = 0 and IC_CON[6] = 0 after the internal SDA and SCL have synchronized to the value on the bus; this takes approximately 6 ic_clk cycles after reset de-assertion.

2.8.1.2 Slave-Transmitter Operation for a Single Byte

When another I^2C master device on the bus addresses the DW_apb_i2c and requests data, the DW_apb_i2c acts as a slave-transmitter and the following steps occur:

1. The other I^2C master device initiates an I^2C transfer with an address that matches the slave address in the IC_SAR register of the DW_apb_i2c.
2. The DW_apb_i2c acknowledges the sent address and recognizes the direction of the transfer to indicate that it is acting as a slave-transmitter.

3. The DW_apb_i2c asserts the RD_REQ interrupt (bit 5 of the IC_RAW_INTR_STAT register) and holds the SCL line low. It is in a wait state until software responds.

   If the RD_REQ interrupt has been masked, due to IC_INTR_MASK[5] register (M_RD_REQ bit field) being set to 0, then it is recommended that a hardware and/or software timing routine be used to instruct the CPU to perform periodic reads of the IC_RAW_INTR_STAT register.

   a. Reads that indicate IC_RAW_INTR_STAT[5] (R_RD_REQ bit field) being set to 1 must be treated as the equivalent of the RD_REQ interrupt being asserted.
   b. Software must then act to satisfy the I2C transfer.
   c. The timing interval used should be in the order of 10 times the fastest SCL clock period the DW_apb_i2c can handle. For example, for 400 kb/s, the timing interval is 25us.

Note The value of 10 is recommended here because this is approximately the amount of time required for a single byte of data transferred on the I^2C bus.

4. If there is any data remaining in the Tx FIFO before receiving the read request, then the DW_apb_i2c asserts a TX_ABRT interrupt (bit 6 of the IC_RAW_INTR_STAT register) to flush the old data from the TX FIFO.

Note Because the DW_apb_i2c's Tx FIFO is forced into a flushed/reset state whenever a TX_ABRT event occurs, it is necessary for software to release the DW_apb_i2c from this state by reading the IC_CLR_TX_ABRT register before attempting to write into the Tx FIFO. See register IC_RAW_INTR_STAT for more details.

   If the TX_ABRT interrupt has been masked, due to of IC_INTR_MASK[6] register (M_TX_ABRT bit field) being set to 0, then it is recommended that re-using the timing routine (described in the previous step), or a similar one, be used to read the IC_RAW_INTR_STAT register.

   a. Reads that indicate bit 6 (R_TX_ABRT) being set to 1 must be treated as the equivalent of the TX_ABRT interrupt being asserted.
   b. There is no further action required from software.
   c. The timing interval used should be similar to that described in the previous step for the IC_RAW_INTR_STAT[5] register.

5. Software writes to the IC_DATA_CMD register with the data to be written (by writing a '0' in bit 8).
6. Software must clear the RD_REQ and TX_ABRT interrupts (bits 5 and 6, respectively) of the IC_RAW_INTR_STAT register before proceeding.

   If the RD_REQ and/or TX_ABRT interrupts have been masked, then clearing of the IC_RAW_INTR_STAT register is already been performed when either the R_RD_REQ or R_TX_ABRT bit has been read as 1.

7. The DW_apb_i2c releases the SCL and transmits the byte.
8. The master may hold the I^2C bus by issuing a RESTART condition or release the bus by issuing a STOP condition.

Note Slave-Transmitter Operation for a Single Byte is not applicable in Ultra-Fast Mode as Read transfers are not supported.

2.8.1.3 Slave-Receiver Operation for a Single Byte

When another I^2C master device on the bus addresses the DW_apb_i2c and is sending data, the DW_apb_i2c acts as a slave-receiver and the following steps occur:

1. The other I^2C master device initiates an I^2C transfer with an address that matches the DW_apb_i2c's slave address in the IC_SAR register.
2. The DW_apb_i2c acknowledges the sent address and recognizes the direction of the transfer to indicate that the DW_apb_i2c is acting as a slave-receiver.
3. DW_apb_i2c receives the transmitted byte and places it in the receive buffer.

Note If the Rx FIFO is completely filled with data when a byte is pushed, and IC_RX_FULL_HLD_BUS_EN = 0, then an overflow occurs and the DW_apb_i2c continues with subsequent I^2C transfers. Because a NACK is not generated, software must recognize the overflow when indicated by the DW_apb_i2c (by the R_RX_OVER bit in the IC_INTR_STAT register) and take appropriate actions to recover from lost data. Hence, there is a real time constraint on software to service the Rx FIFO before the latter overflows, as there is no way to re-apply pressure to the remote transmitting master. You must select a deep enough Rx FIFO depth to satisfy the interrupt service interval of the system.

If the Rx FIFO is completely filled with data when a byte is pushed, and IC_RX_FULL_HLD_BUS_EN = 1, then the DW_apb_i2c slave holds the I2C SCL line low until the Rx FIFO has some space, and then continues with the next read request.

4. DW_apb_i2c asserts the RX_FULL interrupt (IC_RAW_INTR_STAT[2] register).

   If the RX_FULL interrupt has been masked, due to setting IC_INTR_MASK[2] register to 0 or setting IC_TX_TL to a value larger than 0, then it is recommended that a timing routine (described in "Slave-Transmitter Operation for a Single Byte" on page 44) be implemented for periodic reads of the IC_STATUS register. Reads of the IC_STATUS register, with bit 3 (RFNE) set at 1, must then be treated by software as the equivalent of the RX_FULL interrupt being asserted.
5. Software may read the byte from the IC_DATA_CMD register (bits 7:0).
6. The other master device may hold the I^2C bus by issuing a RESTART condition, or release the bus by issuing a STOP condition.

2.8.1.4 Slave-Transfer Operation For Bulk Transfers

In the standard I^2C protocol, all transactions are single byte transactions and the programmer responds to a remote master read request by writing one byte into the slave's TX FIFO. When a slave (slave-transmitter) is issued with a read request (RD_REQ) from the remote master (master-receiver), at a minimum there should be at least one entry placed into the slave-transmitter's TX FIFO. DW_apb_i2c is designed to handle more data in the TX FIFO so that subsequent read requests can take that data without raising an interrupt to get more data. Ultimately, this eliminates the possibility of significant latencies being incurred between raising the interrupt for data each time had there been a restriction of having only one entry placed in the TX FIFO.

This mode only occurs when DW_apb_i2c is acting as a slave-transmitter. If the remote master acknowledges the data sent by the slave-transmitter and there is no data in the slave's TX FIFO, the DW_apb_i2c holds the I^2C SCL line low while it raises the read request interrupt (RD_REQ) and waits for data to be written into the TX FIFO before it can be sent to the remote master.

If the RD_REQ interrupt is masked, due to bit 5 (M_RD_REQ) of the IC_INTR_STAT register being set to 0, then it is recommended that a timing routine be used to activate periodic reads of the IC_RAW_INTR_STAT register. Reads of IC_RAW_INTR_STAT that return bit 5 (R_RD_REQ) set to 1 must be treated as the equivalent of the RD_REQ interrupt referred to in this section. This timing routine is similar to that described in "Slave-Transmitter Operation for a Single Byte" on page 44.

The RD_REQ interrupt is raised upon a read request, and like interrupts, must be cleared when exiting the interrupt service handling routine (ISR). The ISR allows you to either write 1 byte or more than 1 byte into the Tx FIFO. During the transmission of these bytes to the master, if the master acknowledges the last byte. then the slave must raise the RD_REQ again because the master is requesting for more data.

If the programmer knows in advance that the remote master is requesting a packet of *n* bytes, then when another master addresses DW_apb_i2c and requests data, the Tx FIFO could be written with *n* number bytes and the remote master receives it as a continuous stream of data. For example, the DW_apb_i2c slave continues to send data to the remote master as long as the remote master is acknowledging the data sent and there is data available in the Tx FIFO. There is no need to hold the SCL line low or to issue RD_REQ again.

If the remote master is to receive *n* bytes from the DW_apb_i2c but the programmer wrote a number of bytes larger than *n* to the Tx FIFO, then when the slave finishes sending the requested *n* bytes, it clears the Tx FIFO and ignores any excess bytes.

The DW_apb_i2c generates a transmit abort (TX_ABRT) event to indicate the clearing of the Tx FIFO in this example. At the time an ACK / NACK is expected, if a NACK is received, then the remote master has all the data it wants. At this time, a flag is raised within the slave's state machine to clear the leftover data in the Tx FIFO. This flag is transferred to the processor bus clock domain where the FIFO exists and the contents of the Tx FIFO is cleared at that time.

Slave Transmitter Operation for Bulk Transfers is not applicable in Ultra-Fast Mode (IC_ULTRA_FAST_MODE=1) as Master Read Transfers are not supported.

2.8.2 Master Mode Operation

This section discusses master mode procedures.

2.8.2.1 Initial Configuration

The initial configuration procedure for Master Mode Operation depends on the configuration parameter I2C_DYNAMIC_TAR_UPDATE. When set to "Yes" (1), the target address and address format can be changed dynamically without having to disable DW_apb_i2c. This parameter only applies to when DW_apb_i2c is acting as a master because the slave requires the component to be disabled before any changes can be made to the address. For more information about this parameter, see "Parameter Descriptions" on page 95. For more information about how this parameter affects the IC_TAR register, see "Register Descriptions" on page 141.

The procedures are very similar and are only different with regard to where the IC_10BITADDR_MASTER bit is set (either bit 4 of IC_CON register or bit 12 of IC_TAR register).

2.8.2.1.1 I2C_DYNAMIC_TAR_UPDATE = 0

To use the DW_apb_i2c as a master when the I2C_DYNAMIC_TAR_UPDATE configuration parameter is set to "No" (0), perform the following steps:

1. Disable the DW_apb_i2c by writing 0 to bit 0 of the IC_ENABLE register.
2. Write to the IC_CON register to set the maximum speed mode supported (bits 2:1) and the desired speed of the DW_apb_i2c master-initiated transfers, either 7-bit or 10-bit addressing (bit 4). Ensure that bit 6 (IC_SLAVE_DISABLE) is written with a '1' and bit 0 (MASTER_MODE) is written with a '1'.

Slaves and masters do not have to be programmed with the same type of addressing 7- or 10-bit address. For instance, a slave can be programmed with 7-bit addressing and a master with 10-bit addressing, and vice versa.

3. Write to the IC_TAR register the address of the I^2C device to be addressed (bits 9:0). This register also indicates whether a General Call or a START BYTE command is going to be performed by I^2C.
4. *Only applicable for high-speed mode transfers*. Write to the IC_HS_MADDR register the desired master code for the DW_apb_i2c. The master code is programmer-defined.
5. Enable the DW_apb_i2c by writing a 1 to bit 0 of the IC_ENABLE register.
6. Now write transfer direction and data to be sent to the IC_DATA_CMD register. If the IC_DATA_CMD register is written before the DW_apb_i2c is enabled, the data and commands are lost as the buffers are kept cleared when DW_apb_i2c is disabled.

   This step generates the START condition and the address byte on the DW_apb_i2c. Once DW_apb_i2c is enabled and there is data in the TX FIFO, DW_apb_i2c starts reading the data.

Depending on the reset values chosen, steps 2, 3, 4, and 5 may not be necessary because the reset values can be configured. The values stored are static and do not need to be reprogrammed if the DW_apb_i2c is disabled, with the exception of the transfer direction and data.

2.8.2.1.2 I2C_DYNAMIC_TAR_UPDATE = 1

To use the DW_apb_i2c as a master when the I2C_DYNAMIC_TAR_UPDATE configuration parameter is set to "Yes" (1), perform the following steps:

1. Disable the DW_apb_i2c by writing 0 to bit 0 of the IC_ENABLE register.
2. Write to the IC_CON register to set the maximum speed mode supported for slave operation (bits 2:1) and to specify whether the DW_apb_i2c starts its transfers in 7 / 10 bit addressing mode when the device is a slave (bit 3).
3. Write to the IC_TAR register the address of the I^2C device to be addressed. It also indicates whether a General Call or a START BYTE command is going to be performed by I^2C. The desired speed of the

DW_apb_i2c master-initiated transfers, either 7-bit or 10-bit addressing, is controlled by the IC_10BITADDR_MASTER bit field (bit 12).

4. *Only applicable for high-speed mode transfers*. Write to the IC_HS_MADDR register the desired master code for the DW_apb_i2c. The master code is programmer-defined.
5. Enable the DW_apb_i2c by writing a 1 to bit 0 of the IC_ENABLE register.
6. Now write the transfer direction and data to be sent to the IC_DATA_CMD register. If the IC_DATA_CMD register is written before the DW_apb_i2c is enabled, the data and commands are lost as the buffers are kept cleared when DW_apb_i2c is not enabled.

Note When a DW_apb_i2c Master is configured with IC_EMPTYFIFO_HOLD_MASTER_EN = 0, then for multiple I^2C transfers, perform additional writes to the Tx FIFO such that the Tx FIFO does not become empty during the I^2C transaction. If the Tx FIFO is completely emptied at any stage, then further writes to the Tx FIFO results in an independent I^2C transaction.

2.8.2.2 Dynamic IC_TAR or IC_10BITADDR_MASTER Update

The DW_apb_i2c supports dynamic updating of the IC_TAR (bits 9:0) and IC_10BITADDR_MASTER (bit 12) bit fields of the IC_TAR register. In order to perform a dynamic update of the IC_TAR register, the I2C_DYNAMIC_TAR_UPDATE configuration parameter must be set to Yes (1). You can dynamically write to the IC_TAR register provided the software ensures that there are no other commands in the Tx FIFO that use the existing TAR address. If the software does not ensure this, then IC_TAR should be re-programmed only if the following conditions are met:

- DW_apb_i2c is not enabled (IC_ENABLE[0]=0);

  OR

  DW_apb_i2c is enabled (IC_ENABLE[0]=1); AND
  DW_apb_i2c is NOT engaged in any Master (tx, rx) operation (IC_STATUS[5]=0); AND
  DW_apb_i2c is enabled to operate in Master mode (IC_CON[0]=1); AND
  there are NO entries in the Tx FIFO (IC_STATUS[2]=1);[1]

You can change the TAR address dynamically without losing the bus, only if the following conditions are met.

- DW_apb_i2c is enabled (IC_ENABLE[0]=1); AND
  IC_EMPTYFIFO_HOLD_MASTER_EN configuration parameter is set to 1; AND
  DW_apb_i2c is enabled to operate in Master mode (IC_CON[0]=1); AND
  there are NO entries in the Tx FIFO and the master is in HOLD state (IC_INTR_STAT[13]=1);[1]

Note DW_apb_i2c uses the TAR address if either of the following conditions is true:

- The command has either RESTART or STOP bit set.
- The direction is changed in commands with a read command following a write command or vice versa

The updated TAR address comes into effect only when the next START or RESTART occurs on the bus.

1. If the software or application is aware the DW_apb_i2c is not using the TAR address for the pending commands in the Tx FIFO, then it is possible to update the TAR address even while the Tx FIFO has entries (IC_STATUS[2]= 0).

2.8.2.3 Master Transmit and Master Receive

The DW_apb_i2c supports switching back and forth between reading and writing dynamically. To transmit data, write the data to be written to the lower byte of the I^2C Rx/Tx Data Buffer and Command Register (IC_DATA_CMD). The *CMD* bit [8] should be written to 0 for I^2C write operations. Subsequently, a read command may be issued by writing "don't cares" to the lower byte of the IC_DATA_CMD register, and a 1 should be written to the *CMD* bit. The DW_apb_i2c master continues to initiate transfers as long as there are commands present in the transmit FIFO. If the transmit FIFO becomes empty—depending on the value of IC_EMPTYFIFO_HOLD_MASTER_EN, the master either inserts a STOP condition after completing the current transfers, or it checks to see if IC_DATA_CMD[9] is set to 1.

- If set to 1, it issues a STOP condition after completing the current transfer.
- If set to 0, it holds SCL low until next command is written to the transmit FIFO.

For more details, see "Tx FIFO Management and START, STOP and RESTART Generation" on page 36.

Note Master Receiver Mode is not supported in Ultra-Fast Mode (IC_ULTRA_FAST_MODE=1) as Master Read transfers are not supported.

2.8.3 Disabling DW_apb_i2c

The register IC_ENABLE_STATUS is added to allow software to unambiguously determine when the hardware has completely shutdown in response to bit 0 of the IC_ENABLE register being set from 1 to 0. Only one register is required to be monitored, as opposed to monitoring two registers (IC_STATUS and IC_RAW_INTR_STAT) which is a requirement for DW_apb_i2c versions 1.05a or earlier.

Note

When IC_EMPTYFIFO_HOLD_MASTER_EN = 1, the DW_apb_i2c Master can be disabled only if the current command being processed—when the ic_enable de-assertion occurs—has the STOP bit set to 1.

When an attempt is made to disable the DW_apb_i2c Master while processing a command without the STOP bit set, the DW_apb_i2c Master continues to remain active, holding the SCL line low until a new command is received in the Tx FIFO.

When IC_EMPTYFIFO_HOLD_MASTER_EN =1 and the DW_apb_i2c Master is processing a command without the STOP bit set, you can issue the ABORT (IC_ENABLE[1]) to relinquish the I2C bus and then disable DW_apb_i2c.

2.8.3.1 Procedure

1. Define a timer interval (t_{i2c_poll}) equal to the 10 times the signaling period for the highest I^2C transfer speed used in the system and supported by DW_apb_i2c. For example, if the highest I^2C transfer mode is 400 kb/s, then this t_{i2c_poll} is 25us.
2. Define a maximum time-out parameter, MAX_T_POLL_COUNT, such that if any repeated polling operation exceeds this maximum value, an error is reported.
3. Execute a blocking thread/process/function that prevents any further I^2C master transactions to be started by software, but allows any pending transfers to be completed.

Note This step can be ignored if DW_apb_i2c is programmed to operate as an I^2C slave only.

4. The variable POLL_COUNT is initialized to zero.
5. Set bit 0 of the IC_ENABLE register to 0.
6. Read the IC_ENABLE_STATUS register and test the IC_EN bit (bit 0). Increment POLL_COUNT by one. If POLL_COUNT >= MAX_T_POLL_COUNT, exit with the relevant error code.
7. If IC_ENABLE_STATUS[0] is 1, then sleep for t_{i2c_poll} and proceed to the previous step. Otherwise, exit with a relevant success code.

2.8.4 Aborting I2C Transfers

The ABORT control bit of the IC_ENABLE register allows the software to relinquish the I2C bus before completing the issued transfer commands from the Tx FIFO. In response to an ABORT request, the controller issues the STOP condition over the I2C bus, followed by Tx FIFO flush. Aborting the transfer is allowed only in master mode of operation.

2.8.4.1 Procedure

1. Stop filling the Tx FIFO (IC_DATA_CMD) with new commands.
2. When operating in DMA mode, disable the transmit DMA by setting TDMAE to 0.
3. Set bit 1 of the IC_ENABLE register (ABORT) to 1.
4. Wait for the M_TX_ABRT interrupt.
5. Read the IC_TX_ABRT_SOURCE register to identify the source as ABRT_USER_ABRT.

2.9 Spike Suppression

The DW_apb_i2c contains programmable spike suppression logic that match requirements imposed by the *I^2C Bus Specification* for SS / FS (tSP, Table 9), high speed (tSP, Table 11), and UFm (tSP, Table 13) modes.

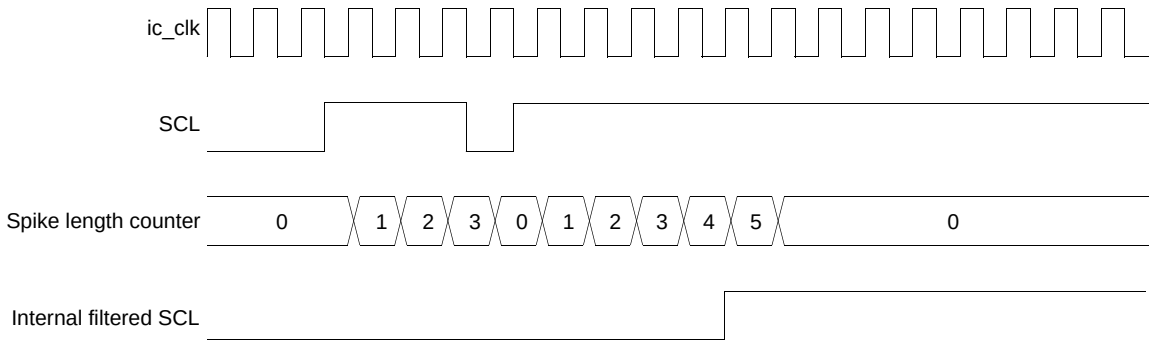
This logic is based on counters that monitor the input signals (SCL and SDA), checking if they remain stable for a predetermined amount of ic_clk cycles before they are sampled internally. There is one separate counter for each signal (SCL and SDA). The number of ic_clk cycles can be programmed and should be calculated taking into account the frequency of ic_clk and the relevant spike length specification.

Each counter is started whenever its input signal changes its value. Depending on the behavior of the input signal, one of the following scenarios occurs:

- The input signal remains unchanged until the counter reaches its count limit value. When this happens, the internal version of the signal is updated with the input value, and the counter is reset and stopped. The counter is not restarted until a new change on the input signal is detected.
- The input signal changes again before the counter reaches its count limit value. When this happens, the counter is reset and stopped, but the internal version of the signal is not updated. The counter remains stopped until a new change on the input signal is detected.

The timing diagram in Figure 2-24 illustrates the behavior described earlier.

Figure 2-24 Spike Suppression Example



The count limit value used in this example is 5 and is calculated for a 10 ns ic_clk period and for SS/FS operation (50 ns spike suppression).

Note There is a 2-stage synchronizer on the SCL input, but for the sake of simplicity this synchronization delay is not included in the timing diagram in Figure 2-24.

The *I^2C Bus Specification* calls for different maximum spike lengths according to the operating mode—50 ns for SS and FS; 10 ns for high speed, 10 ns for UFm, so three registers are required to store the values needed for each case:

- Register IC_FS_SPKLEN holds the maximum spike length for SS and FS modes
- Register IC_HS_SPKLEN holds the maximum spike value for high speed mode.
- Register IC_UFM_SPKLEN holds the maximum spike value for UFm.

Note

- IC_HS_SPKLEN is implemented only if the component is configured for high speed operation; that is, (IC_MAX_SPEED = High).
- IC_UFM_SPKLEN is implemented only if the component is configured for Ultra-Fast mode; that is, (IC_ULTRA_FAST_MODE=1).
- IC_FS_SPKLEN and IC_HS_SPKLEN are not implemented when configured for Ultra-Fast mode; that is, (IC_ULTRA_FAST_MODE=1).

These registers are 8 bits wide and accessible through the APB interface for read and write purposes; however, they can be written to only when the DW_apb_i2c is disabled. The minimum value that can be programmed into these registers is 1; attempting to program a value smaller than 1 results in the value 1 being written.

The default value for these registers is automatically calculated in coreConsultant based on the value of ic_clk period, but this value can be overridden when configuring the component.

Note

- Because the minimum value that can be programmed into the IC_FS_SPKLEN, IC_HS_SPKLEN, and IC_UFM_SPKLEN registers is 1, the spike length specification can be exceeded for low frequencies of ic_clk. Consider the simple example of a 10 MHz (100 ns period) ic_clk; in this case, the minimum spike length that can be programmed is 100 ns, which means that spikes up to this length are suppressed.
- Standard synchronization logic (two flip-flops in series) is implemented upstream of the spike suppression logic and is not affected in any way by the contents of the spike length registers or the operation of the spike suppression logic; the two operations (synchronization and spike suppression) are completely independent.

  Because the SCL and SDA inputs are asynchronous to ic_clk, there is one ic_clk cycle uncertainty in the sampling of these signals; that is, depending on when they occur relative to the rising edge of ic_clk, spikes of the same original length might show a difference of one ic_clk cycle after being sampled.
- Spike suppression is symmetrical; that is, the behavior is exactly the same for transitions from 0 to 1 and from 1 to 0.

2.10 Fast Mode Plus Operation

In fast mode plus, the DW_apb_i2c allows the fast mode operation to be extended to support speeds up to 1000 Kb / s. To enable the DW_apb_i2c for fast mode plus operation, perform the following steps before initiating any data transfer:

1. Configure the Maximum Speed mode of DW_apb_i2c Master or Slave to Fast Mode or High Speed mode (IC_MAX_SPEED_MODE> = 2).
2. Set ic_clk frequency greater than or equal to 32 MHz (see “Standard Mode (SM), Fast Mode (FM), and Fast Mode Plus (FM+) with IC_CLK_FREQ_OPTIMIZATION = 0” on page 69).
3. Program the IC_CON register [2:1] = 2’b10 for fast mode or fast mode plus.
4. Program IC_FS_SCL_LCNT and IC_FS_SCL_HCNT registers to meet the fast mode plus SCL (see “IC_CLK Frequency Configuration” on page 66).
5. Program the IC_FS_SPKLEN register to suppress the maximum spike of 50ns.
6. Program the IC_SDA_SETUP register to meet the minimum data setup time (tSU; DAT).

2.11 Bus Clear Feature

DWC_apb_i2c supports the bus clear feature that provides graceful recovery of data (SDA) and clock (SCL) lines during unlikely events in which either the clock or data line is stuck at LOW.

The following sections describes the SDA and SCL lines stuck at LOW recovery mechanisms:

- “SDA Line Stuck at LOW Recovery” on page 53
- “SCL Line is Stuck at LOW” on page 54

2.11.1 SDA Line Stuck at LOW Recovery

In case of SDA line stuck at LOW, the master performs the following actions to recover as shown in Figure 2-25 and Figure 2-26:

1. Master sends a maximum of 9 clock pulses to recover the bus LOW within those 9 clocks.
   - The number of clock pulses varies with the number of bits that remain to be sent by the slave. As the maximum number of bits is 9, master sends up to 9 clock pluses and allows the slave to recover it.
   - The master attempts to assert a Logic 1 on the SDA line and check whether SDA is recovered. If the SDA is not recovered, it continues to send a maximum of 9 SCL clocks.
2. If SDA line is recovered within 9 clock pulses then the master sends the STOP to release the bus.
3. If SDA line is not recovered even after the 9^{th} clock pulse then system needs a hardware reset.

The detailed flow to recover the SDA stuck at LOW is explained in the section "Programming Flow for SCL and SDA Bus Recovery" on page 304.

Figure 2-25 SDA Recovery with 9 SCL Clocks

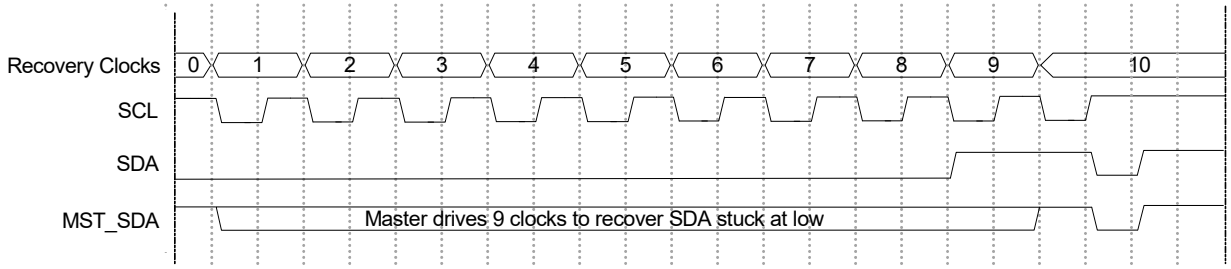
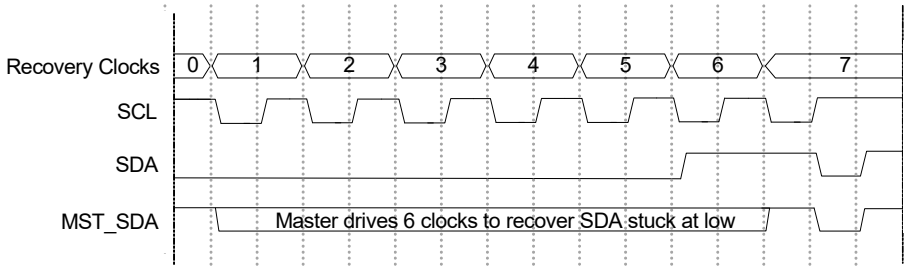


Figure 2-26 SDA Recovery with 6 SCL Clocks



2.11.2 SCL Line is Stuck at LOW

In the unlikely event (due to an electric failure of a circuit) where the clock (SCL) is stuck to LOW, there is no effective method to overcome this problem but to reset the bus using the hardware reset signal. The detailed flow to recover the SCL stuck at LOW is explained in "Programming Flow for SCL and SDA Bus Recovery" on page 304.

2.12 Device ID

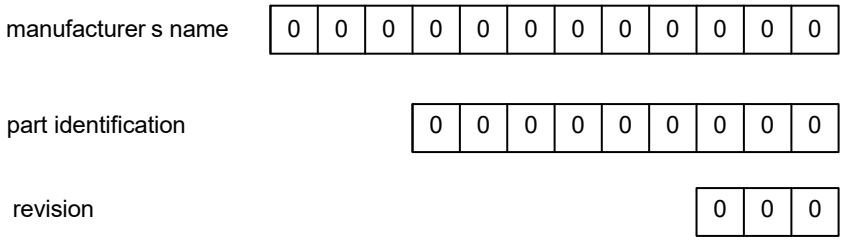
A Device ID field is an optional 3-byte read-only (24 bits) word, which provides the following information:

- Twelve bits with the manufacturer's name, which is unique for every manufacturer.
- Nine bits with the part identification, which is assigned by the manufacturer.

- Three bits with the die revision, which is assigned by the manufacturer.

Figure 2-27 shows the Device ID field structure.

Figure 2-27 Device ID Field Structure



For reading the Device ID of a particular slave, the master can follow the procedure in "Programming Flow for Reading the Device ID" on page 305. The Device ID that is read is available in RX FIFO, which can be read using IC_DATA_CMD register.

In case of a slave, you have to configure the Device ID using the IC_DEVICE_ID_VALUE coreConsultant parameter and you can read the Device ID of the slave using IC_DEVICE_ID register.

Note Device ID is not supported for 10-bit addressing and High Speed transfers (high speed mode).

2.13 Ultra-Fast Speed Mode

The Ultra-Fast Speed mode is a variant of I2C Bus Speed mode that operates from DC (0) to 5 MHz transmitting data in one direction. It is useful for speeds greater than 1 MHz to drive LED controllers and other gaming systems that do not need feedback.

Ultra-Fast speed mode is based on the standard I2C Protocol, which consists of START, slave address, command bit, ninth clock (ACK cycle) and a STOP bit. The command bit should be always 'write' (0) only since it is a unidirectional bus (except for the START byte). The data bit on the ninth (ACK) cycle is driven high by the master, ignoring the ACK cycle due to unidirectional nature of bus. The driver used for Ultra-Fast Mode is push-pull driver.

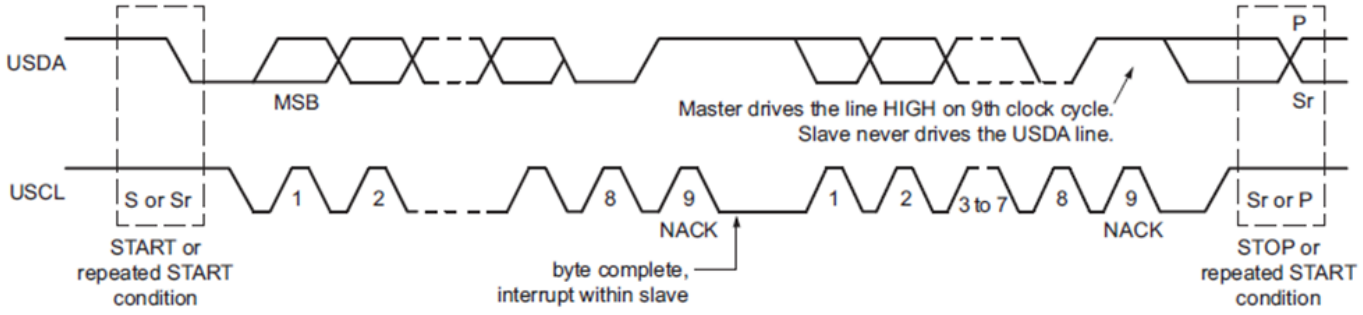
The Master consists of serial clock (ic_clk_oe, USCL) and a serial data (ic_data_oe, USDA) output signals. The Output signals are Active-Low in nature.

The Slave consists of serial clock (ic_clk_in_a, USCL) and serial data (ic_data_in_a, USDA) input signals. The input signals are Active-High in nature.

The UFm I2C-bus does not have the multi-master capability and hence, it does not consist of wired-AND open-drain driver. In the UFm I2C bus, the master is the only device that initiates a data transfer (write transfer) on the bus and provides the clock signals to support that transfer. All other devices are considered as slaves. Because of single master support, the arbitration, synchronization, clock stretching mechanisms are not applicable.

The Byte format, START and STOP generation are same as in other modes of the I2C Protocol except for the ignorance of ACK cycle. The Slave never drives anything on the bus hence, the master always drives NACK during the ninth cycle of the transfer as shown in Figure 2-28.

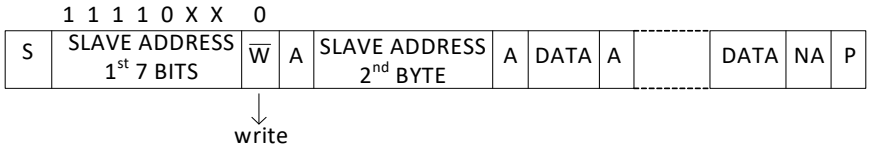
Figure 2-28 UFm-I2C Byte Transfer



In UFm-I2C mode, the slave is not allowed to hold the clock LOW if it cannot receive another complete byte of data or while it is performing some other function, for example, servicing an internal interrupt. The ninth clock cycle that represents ACK / NACK of the byte is not applicable because slave does not respond and it is preserved in UFm to be compatible with the I2C Protocol. The 8th bit of the address that represents Read or write transfer should be always set to write (0), since Read is not supported in UFm (except for the START Byte).

The Combined format of I2C Protocol is not supported in UFm-I2C mode. The 10-bit addressing that expands the number of possible devices is supported in UFm-I2C mode and it behaves similar to other modes as shown in Figure 2-29 (Only write transfer is supported).

Figure 2-29 10-bit addressing write transfer



The UFm-I2C mode supports START byte and general call features similar to other I2C modes. If the slave is not responsive (determined through external feedback and not through UFm I2C-bus), then the slave can reset through software reset or external hardware reset.

2.14 SMBus/PMBus

The SMBus is designed to provide a predictable communication line between a system and its devices. It describes the Device timeout definitions and their conditions.

2.14.1 $t_{Timeout,MIN}$ Parameter

This Parameter allows a master or slave to conclude that a defective device is holding the clock low indefinitely or a master is intentionally trying to drive devices off the bus. It is highly recommended that a slave device release the bus (stop driving the bus and let SMBCLK and SMBDAT float high) when it detects any single clock held low longer than tTIMEOUT,MIN. Devices that have detected this condition must reset their communication interface and be able to receive a new START condition in no later than tTIMEOUT,MAX.

The DW_apb_i2c enables the Bus clear feature in SMBus mode and the you can use the IC_SCL_STUCK_AT_LOW_TIMEOUT Register to program the tTIMEOUT,MIN Value to detect the SMBCLK low timeout.

The DW_apb_i2c slave device resets its communication interface and release both SCL and SDA lines after detecting the SCL_STUCK_TIMEOUT interrupt.

The DW_apb_i2c master has a provision to generate the Abort which completes the current transfer and generate STOP condition on the bus through programming the IC_ENABLE[1] register bit.

2.14.2 Master Device Clock Extension

The interval tLOW: MEXT is defined as the cumulative time a master device is allowed to extend its clock cycles within one byte in a message as measured from:

- START to ACK
- ACK to ACK
- ACK to STOP.

The DW_apb_i2c Master uses the IC_SMBUS_CLOCK_LOW_MEXT register to detect the Master device clock extension timeout and generates SMBUS_CLK_LOW_MEXT interrupt.

2.14.3 Slave Device Clock Extension

The interval tLOW:SEXT is the cumulative time a given slave device is allowed to extend the clock cycles in one message from the initial START to the STOP.

The DW_apb_i2c Master uses the IC_SMBUS_CLOCK_LOW_SEXT register to detect the Slave device clock extension timeout and generates SMBUS_CLK_LOW_SEXT interrupt.

A Master is allowed to abort the transaction in progress to any slave that violates the tLOW:SEXT or tTIMEOUT,MIN specifications through the enabling the user abort (IC_ENABLE[1]).

2.14.4 SMBDAT Low Timeout

A malfunctioning device holds the SMBDAT line low indefinitely. This would prevent the master from issuing a STOP condition and ending a transaction. If SMBDAT is still low tTIMEOUT,MAX after SMBCLK has gone high at the end of a transaction the master should hold SMBCLK low for at least tTIMEOUT,MAX in an attempt to reset the SMBus interface of all of the devices on the bus.

The DW_apb_i2c enables the Bus clear feature in SMBus mode and you can use the IC_SDA_STUCK_AT_LOW_TIMEOUT Register to program the SMBDAT timeout value to detect the SMBDAT low timeout. If SMBDAT line is stuck at low, the SDA_STUCK_TIMEOUT abort is generated and software can enable the SMBUS_CLK_RESET register bit of IC_ENABLE register to hold the SCL low for IC_SCL_STUCK_AT_LOW_TIMEOUT which in turn resets the SMBus interface of all devices on the bus.

2.14.5 Bus Protocols

A typical SMBus device has a set of commands by which data can be read and written. All commands are one byte long while their arguments and return values can vary in length. In accordance with the SMBus specification, the most significant bit (MSB) is transferred first. There are eleven possible command protocols for any given device. These commands are Quick Command, Send Byte, Receive Byte, Write Byte,

Write Word, Read Byte, Read Word, Process Call, Block Read, Block Write, and Block Write-Block Read Process Call.

SMBus protocols for message transactions are generally different from I2C data transfer commands. It is still possible to program an SMBus master to deliver I2C data transfer commands. The following table describes the derivation of SMBus Bus Protocols through Tx-FIFO commands in DW_apb_i2c.

In the SMBus Master mode, all the receive data bytes are available in Rx-FIFO. In the SMBus Slave mode, all the bus protocol command codes and data bytes are received in the Rx-FIF0 and read request data bytes must be sent using Tx-FIFO, similar to the I2C mode.

Table 2-2 SMBus Bus Protocols Usage in DW_apb_i2c

Protocol	Required TxFIFO Commands	Command/Data (IC_DATA_CMD[7:0])	CMD bit (IC_DATA_CMD[8])	STOP bit (IC_DATA_CMD[9])	Remarks
Quick Command	1	Not Applicable	Set the command [R/W]	Set to 1	Set IC_TAR[11] and IC_TAR[16] to 1
Send Byte	1	Data Byte	Set to 0	Set to 1	
Receive Byte	1	Not Applicable	Set to 1	Set to 1	
Write Byte	2	Command Code	Set to 0	Set to 0	
		Data Byte	Set to 0	Set to 1	
Write Word	3	Command Code	Set to 0	Set to 0	
		Data Byte Low	Set to 0	Set to 0	
		Data Byte High	Set to 0	Set to 1	
Read Byte	2	Command Code	Set to 0	Set to 0	
		Not Applicable	Set to 1	Set to 1	
Read Word	3	Command Code	Set to 0	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 1	
Process Call	5	Command Code	Set to 0	Set to 0	
		Data Byte Low	Set to 0	Set to 0	
		Data Byte High	Set to 0	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 1	

Protocol	Required TxFIFO Commands	Command/Data (IC_DATA_CMD[7:0])	CMD bit (IC_DATA_CMD[8])	STOP bit (IC_DATA_CMD[9])	Remarks
Block Write	N+2	Command Code	Set to 0	Set to 0	
		Data Byte	Set to 0	Set to 0	
		N+1) Data Byte N	Set to 0	Set to 1	
Block Read	N+2	Command Code	Set to 0	Set to 0	
		Not Applicable	Set to 0	Set to 0	
		N+1) Not Applicable	Set to 0	Set to 1	
Block Write-Block Read Process Call	M+N+2	Command Code	Set to 0	Set to 0	
		Data Byte 1	Set to 0	Set to 0	
		M+1) Data Byte M	Set to 0	Set to 0	
		M+2) Not Applicable	Set to 1	Set to 0	
		M+3) Not Applicable	Set to 1	Set to 0	
		M+N+1) Not Applicable	Set to 1	Set to 1	
SMBUS Host Notify Protocol	3	Device-Address	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Host Address (0001 000)
		Data Byte Low	Set to 0	Set to 0	
		Data Byte High	Set to 0	Set to 1	

DW_apb_i2c Slave can be enabled to receive only Quick command through enabling the SLAVE_QUICK_CMD_EN bit in the IC_CON Register. Whenever this bit is selected the slave only receives quick commands and are not accept other Bus Protocols. The DW_apb_i2c slave issues the SMBUS_QUICK_DET interrupt upon receiving the QUICK command.

SMBus introduces a Packet Error checking Mechanism through appending PEC Byte at the end of the Bus Protocol. This can be achieved through adding an extra command (PEC byte) while transferring and decoding it while receiving by the software.

2.14.6 SMBUS Address Resolution Protocol

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device by the Host. This feature allows the devices to be 'hot-plugged' in to the system.

SMBus introduces a 128-bit Unique Device ID (UDID) for each device in the system to isolate each device for the purpose of address assignment. DW_apb_i2c uses the IC_SMBUS_UDID_MSB parameter for upper constant 96 bits and 'IC_SMBUS_ARP_UDID_LSB' register for lower variable 32 bits of the UDID.

DW_apb_i2c uses the PERSISTANT_SLV_ADDR_EN register bit in IC_CON register to indicate whether the DW_apb_i2c supports persistent slave address.

DW_apb_i2c master can issue general and directed Address Resolution Protocol (ARP) commands to assign the dynamic address for the slaves in the SMBus system.

Table 2-3 describes the derivation of SMBus ARP commands through Tx-FIFO commands in DW_apb_i2c.

Table 2-3 Derivation of SMBus ARP Command Through TxFIFO Commands in DW_apb_i2c

ARP Command	Required Tx_FIFO Commands	Command/Data (IC_DATA_CMD[7:0])	CMD Bit (IC_DATA_CMD[8])	STOP bit (IC_DATA_CMD[9])	Remarks
Prepare for ARP	2	Command = '0000 0001'	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Default Address (1100 001)
		PEC Byte	Set to 0	Set to 1	
Reset Device (General)	2	Command = '0000 0010'	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Default Address (1100 001)
		PEC Byte	Set to 0	Set to 1	
Get UDID (General)	20	Command = '0000 0011'	Set to 0	Set to 0	1. Set IC_TAR[6:0] to SMB Default Address (1100 001). 2. 16 Reads to be performed for the 128 UDID bytes. 3. Last read command for the slave address.
		Not Applicable	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		PEC Byte	Set to 1	Set to 1	
Assign Address	20	Command = '0000 0011'	Set to 0	Set to 0	1. Set IC_TAR[6:0] to SMB Default Address (1100 001). 2. 16 Writes to be performed for the 128 UDID byte. 3. Last Write command for the Assigned slave address.
		Byte Count = 17	Set to 0	Set to 0	
		UDID Byte 15	Set to 0	Set to 0	
		UDID Byte 14	Set to 0	Set to 0	
		Assigned Address	Set to 0	Set to 0	
		PEC Byte	Set to 01	Set to 1	

ARP Command	Required Tx_FIFO Commands	Command/Data (IC_DATA_CMD[7:0])	CMD Bit (IC_DATA_CMD[8])	STOP bit (IC_DATA_CMD[9])	Remarks
Get UDID (Directed)	19	Command = '0000 0011'	Set to 0	Set to 0	1. Set IC_TAR[6:0] to SMB Default Address (1100 001). 2. 16 Reads to be performed for the 128 UDID byte. 3. Last Read command for the slave address.
		Slave address[6:0],1}	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		PEC Byte	Set to 1	Set to 1	
Reset Device (Directed)	2	command = {slave address[6:0],0}	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Default Address (1100 001)
		PEC byte	Set to 0	Set to 1	
Notify ARP Master	3	Device Address = '1100 0010'	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Host Address (0001 000)
		Data Byte Low = '0000 0000'	Set to 0	Set to 0	
		Data Byte High = '0000 0000'	Set to 0	Set to 1	

Note

- DW_apb_i2c slave hardware:
  - Handles the generation, detection, and NACKing of the wrong PEC (CRC8 C(X)=X8+X2+X1+1) for the ARP Commands.
  - Does not handle the PEC for Non-ARP commands.
- DW_apb_i2c master hardware does not handle PEC for both APR and non-ARP commands.

2.14.6.1 Procedure to Perform ARP in Master Mode

To use the DW_apb_i2c as a SMBus Master / Host for assigning the unique address to each slave device to resolve the slave address conflicts, perform the following steps:

1. After a reset or a cold power up, the SMBus host or master issues a "Prepare to ARP" command to indicate that the master is carrying an ARP to assign dynamic addresses to all devices. Slave must flush any pending host notify commands.
2. An acknowledgement received for the "Prepare to ARP" command indicates that ARP-capable devices exist in the system and hte "Get UDID" command must be issued. A NACK indicates that ARP-capable devices do not exist or currently all slaves have their addresses resoved. In this case, the master must complete steps outlined from Step 8 onwards. The DW_apb_i2c master indicates NACK reception through 'ABRT_7B_ADDR_NOACK' and ‘ABRT_TXDATA_NOACK’ bits of IC_TX_ABRT_SOURCE register.

3. DW_apb_i2c Master issues ‘Get UDID’ to receive the UDID information of the slave for assigning the dynamic address.
4. If the first three bytes of the "Get UDID" command are ACK'ed and the receive byte count is 0x11, then the master issues the "Assign Address" command. Else, the master must complete steps outlined in step 8 onwards to indicate that the ARP is complete. DW_apb_i2c Master indicates NACK reception through ABRT_7B_ADDR_NOACK and ABRT_TXDATA_NOACK bits of the IC_TX_ABRT_SOURCE register.
5. The Master issues the "Assign Address" command to assign the Dynamic address to the slave whose UDID is received through "Get UDID command".
6. If the assigned address packet is ACK'ed, then Master removes the assigned address from the address pool and moves to Step 3 to get UDID of another slave. If the packet is not ACK'ed, then master does not remove the address from the address pool and moves to Step 3 to get UDID of same slave or another slave.
7. If the Assign Address is ACK'ed, then Master stores the assigned address in the used address pool with the UDID characteristics of the device.
8. The Master moves to Step 3 to issue a 'Get UDID' command again to receive the UDID of another slave. If it receives NACK for 'Get UDID', the Master moves to Step 9.
9. The DW_apb_i2c can be switched to Slave mode to detect device requests for Host Notify Protocol.
10. If the DW_apb_i2c switched to slave mode and DW_apb_i2c detects the Host Notify Protocol, then this indicates that a slave is requesting for the dynamic address and the Master has to undergo the ARP as outlined in Step 11.
11. If the DW_apb_i2c is in Master mode, then move to Step 3 for performing ARP procedure, otherwise move to Step 12.
12. The DW_apb_i2c is switched to Master Mode and moves to Step 3 to perform ARP procedure.

The detailed flow diagram is explained in Figure 6-10.

2.14.6.2 Procedure to Perform ARP in Slave Mode

The DW_apb_i2c as a SMBus Slave performs the following tasks:

- Decodes the ARP commands and responds based on internal state flags SMBUS_SLAVE_ADDR_VALID and 'SMBUS_SLAVE_ADDR_RESOLVED' of the IC_STATUS register.
- Generates and Validates the PEC byte of ARP commands
- Generates ACK for the PEC byte only if it matches the CRC value calculated on data it received. If not, NACK the PEC byte.

When another SMBus Master / Host device on the bus generates the ARP commands and requests to participate in the ARP, the DW_apb_i2c acts as a SMBus slave and performs the following steps:

1. After a reset or a cold power up, the DW_apb_i2c slave device checks whether it supports a persistent slave address.

2. If DW_apb_i2c has a persistent slave address (PSA), which is indicated by the Address Valid flag being set, then PSA is set in the Slave Address Register (IC_SAR) register. If the flag is not set, then proceed to Step 4.
3. DW_apb_i2c persistent slave stores the persistent address in IC_SAR and sets Address Valid flag to 1 and Address Resolved Flag to 0.
4. DW_apb_i2c Non Persistent slave (non-PSA) clears both Address Valid and Address Resolved Flags.
5. DW_apb_i2c Checks whether any Packet received has ARP Default address in the slave address field of the packet to decide on ARP command or normal command. If there is a match then DW_apb_i2c slave proceeds to Step 6, otherwise to Step 25.
6. If DW_apb_i2c detects a packet addressed to the SMBus Device Default Address, it checks the command field to determine if this is the "Prepare to ARP" command. If so, then it proceeds to Step 7, otherwise it proceeds to Step 8.
7. Upon receipt of the "Prepare to ARP" command, the DW_apb_i2c acknowledges the packet and clears the Address Resolved flag in order to participate in the ARP Process. DW_apb_i2c proceeds to Step 5 and waits for another SMBus Packet.
8. The DW_apb_i2c checks the command field to verify if the "Reset Device" command is issued. If yes, the DW_apb_i2c proceeds to Step 9, otherwise it proceeds to Step 10.
9. Upon receipt of the "Reset Device" command, the DW_apb_i2c acknowledges the packet and clears the Address Resolved and Address Valid (If non-PSA and ic_con[19]=0) flags. DW_apb_i2c procceds to Step 5 and waits for another SMBus Packet.
10. The device checks the command to verify if the "Assign Address" command is issued. If yes, then it proceeds to Step 11, otherwise proceeds to Step 13.
11. Upon receipt of the "Assign Address" command, the DW_apb_i2c compares its UDID with one its received bytes. If any byte does not match, then DW_apb_i2c does not acknowledge that byte and subsequent bytes also. If all bytes in the UDID matches, then the DEVICE proceeds to Step 12, otherwise it proceeds to Step 5 and waits for another SMBus packet.
12. After the UDID is matched in Step 11, the DW_apb_i2c receives the slave address and sets the IC_SAR register with this slave address. The DW-apb_i2c sets its Address Valid and Address Resolved flags, which means it has received the dynamic address and is no longer respond to the "Get UDID" command unless it receives the "Prepare to ARP" or "Reset Device" commands. DW_apb_i2c now proceeds to Step 5 and waits for another SMBus packet.
13. The DW_apb_i2c checks the command field to verify if the "Get UDID" command is issued. If yes, then it proceeds to Step 14, otherwise to Step 19.
14. Upon receipt of the "Get UDID" command, the DW_apb_i2c checks its Address Resolved flag to determine whether it must participate in an ARP process. If set, then its address has already been resolved by the ARP Master, so the device proceeds to Step 5 and waits for another SMBus packet. If the ARP Flag is cleared, then it proceeds to Step 15.
15. The DW_apb_i2c returns its UDID and monitors the SMBus data line for collisions. If a collision is detected at any time, DW_apb_i2c generates the SLV_ARB_LOST bit and stops transmitting. Further, it proceeds to Step 5 and waits for another SMBus packet. If collisions are not detected, then DW_apb_i2c proceeds to Step 16.

16. The DW_apb_i2c check its Address Valid (AV) flag to determine the value to return for the Device Slave Address field. If the AV flag is set, then it proceeds to Step 17, otherwise it proceeds to Step 18.
17. When the AV flag is set, the current IC_SAR is valid, therefore the device returns this for the Device Slave Address field (with bit 0 set) and monitors the SMBus data line for collisions. DW_apb_i2c proceeds to Step 5 and waits for another SMBus Packet.
18. When the AV flag is not set, the current slave address (IC_SAR) is invalid. Therefore, the DW_apb_i2c returns a value of FFh and monitors the SMBus data line for collisions. The device requires an address assignment if the ARP master receives the FFH value. DW_apb_i2c proceeds to Step 5 and waits for another SMBus packet.
19. The DW_apb_i2c may be receiving a directed command. If the Address Valid flag is set and address is the same as in IC_SAR, then proceed to Step 20 otherwise, proceed to Step 5 to wait for another SMBus packet.
20. If the Address Valid flag is set, check if the command is a directed "Reset Device" command. If yes, then proceed to Step 21, otherwise proceed to Step 22.
21. Upon receipt of the "Reset Device" command, the DW_apb_i2c acknowledges the packet and clears the Address Resolved and Address Valid (If non-PSA and ic_con[19]=0) flags. DW_apb_i2c procceds to Step 5 and waits for another SMBus Packet.
22. DW_apb_i2c checks whether the received command is a "Directed Get UDID" command. If yes, then proceed to Step 23 and return the UDID information. If not, then proceed to Step 24.
23. If the received command is a "Directed Get UDID" command, then return the UDID information and current slave address, proceed to Step 5 and wait for another SMBus Packet.
24. If the received command is a "Directed Get UDID" command, the DW_apb_i2c has not received a valid ARP command and hence DW_apb_i2c NACKs the command and proceeds to Step 5 and wait for another SMBus Packet.
25. If the Address Valid bit is set then it proceeds to Step 26, otherwise it proceeds to Step 5 and waits for another SMBus Packet. The received address is not the SMBus Device Default Address and the packet may be addresses to the DW_apb_i2c's core function. The device checks its Address Valid bit to determine whether to respond.
26. When the address valid bit is set, DW_apb_i2c has a valid slave address. It compares the received slave address to its slave address, and if there is s a match, DW_apb_i2c proceeds to Step 27, otherwise it proceeds to Step 5 and waits for another SMBus Packet.
27. The DW_apb_i2c receives a packet addresses to its core function and hence it acknowledges the packet and processes it accordingly. DW_apb_i2c proceeds to step 5 and waits for another SMBus Packet.

The detailed flow diagram is explained in Figure 6-11.

2.14.7 SMBUS Additional Slave Address

DW_apb_i2c supports second optional slave address decode capability. It can be configured to contain an extra slave address register IC_OPTIONAL_SAR. If configured with this additional register, you can write any valid slave address to this register which is matched against an incoming slave address on SMBus. A match of incoming address with either IC_SAR register or IC_OPTIONAL_SAR register causes DW_apb_i2c to acknowledge the transaction and respond to it accordingly. Use of this additional slave

address register is controlled by OPTIONAL_SAR_CTRL (IC_CON[17]) bit. If OPTIONAL_SAR_CTRL bit is programmed to be 1, then IC_OPTIONAL_SAR register is used to match the incoming address. All restrictions of IC_SAR register applies to IC_OPTIONAL_SAR register as well.

The default value that IC_OPTIONAL_SAR register obtains after reset can be configured by the IC_OPTIONAL_SAR_DEFAULT parameter.

2.14.8 SMBUS Optional Signals

The SMBus standard supports these optional signals:

- SMBus Suspend Signal
- SMBus Alert Signal

As these signals are optional, DW_apb_i2c can be configured to include these signals through IC_SMBUS_SUSPEND_ALERT parameter.

2.14.8.1 SMBus Suspend Signal

The SMBus suspend signal (SMBSUS#) is an optional signal which is asserted by the system controller (mostly the Host) to indicate that the system should enter in low power suspend mode. It is output from the system controller and input to all other devices. This signal is an active low signal. DW_apb_i2c implements this functionality using following signals:

- ic_smbsus_in_n
- ic_smbsus_out_n

Output signal ic_smbsus_out_n is controlled directly by the SMBUS_SUS_CTRL bit (IC_ENABLE[17]). If this bit is programmed to 1, ic_smbsus_out_n signal goes to 0 as soon as master finishes any ongoing transfer. For coming out of the suspend mode, you need to clear this bit, which de-asserts the ic_smbus_out_n signal.

Input signal ic_smbsus_in_n generates interrupt ic_smbsus_det_intr (or ic_smbsus_det_intr_n) on the falling edge. This interrupt can be used by the software to enter the Low Power Mode. Current status of this ic_smbsus_in_n can be read from SMBUS_SUSPEND_STATUS bit of IC_STATUS (19) register.

2.14.8.2 SMBus Alert Signal

The SMBus alter signal (SMBALERT#) is other optional signal specified by the SMBus standard. It can be used by simple devices to request the attention of the host. Devices can use the SMBALERT# signal to request the attention of the host with master functionality. This signal is input to host device and output from all other devices. Since multiple devices may implement SMBALERT#, it is required to be a wired-AND signal. Upon detecting a SMBALERT# signal, a host must send an alert response address which is acknowledge by alerting the device and it sends the address to the host and de-asserts the alert signal. If host still detects an asserted alert signal, it repeats sending alert response address. DW_apb_i2c implements this functionality using following signals:

- ic_smbalert_in_n
- ic_smbalert_oe

Output signal ic_smbalert_oe is open drain / open collector pull down driver and should be used similar to ic_clk_oe and ic_data_oe on a system implementation. Assertion of ic_smbalert_oe is controlled by SMBUS_ALERT_CTRL bit (IC_ENABLE[18]). Once asserted, DW_apb_i2c waits for alert response address

to be sent by master. Upon receiving it, contents of IC_SAR[7:0] register are sent to the master. When successful, DW_apb_i2c clears the SMBUS_ALERT_CTRL bit and de-asserts the ic_smbalert_oe signal.

Input signal ic_smbalert_in_n generates interrupt ic_smbalert_det_intr (or ic_smbalert_det_intr_n) on falling edge. If working as host, you need to service this interrupt by sending read byte command with Alert Response Address. Current status of ic_smbalert_in_n can be read from SMBUS_ALERT_STATUS bit (IC_STATUS[20])

2.15 IC_CLK Frequency Configuration

When the DW_apb_i2c is configured as a Standard (SS), Fast (FS) / Fast-Mode Plus (FM+), or High Speed (HS) master, the *CNT registers must be set before any I^2C bus transaction can take place in order to ensure proper I / O timing. The *CNT registers are:

- IC_SS_SCL_HCNT
- IC_SS_SCL_LCNT
- IC_FS_SCL_HCNT
- IC_FS_SCL_LCNT
- IC_HS_SCL_HCNT
- IC_HS_SCL_LCNT

When the DW_apb_i2c is configured as a Ultra-Fast Mode master, the *CNT registers must be set before any I^2C bus transaction can take place in order to ensure proper I / O timing. The *CNT registers for this mode are:

- IC_UFM_SCL_HCNT
- IC_UFM_SCL_LCNT

Note The tBUF timing and setup/hold time of START, STOP and RESTART registers uses *HCNT/ *LCNT register settings for the corresponding speed mode.

Note It is not necessary to program any of the *CNT registers if the DW_apb_i2c is enabled to operate only as an I^2C slave, since these registers are used only to determine the SCL timing requirements for operation as an I^2C master.

Table 2-4 lists the derivation of I^2C timing parameters from the *CNT programming registers.

Table 2-4 Derivation of I^2C Timing Parameters from *CNT Registers

Timing Parameter	Symbol	Standard Speed	Fast Speed / Fast Speed Plus	High Speed (100 pf)	High Speed (400 pf)
LOW period of the SCL clock	tLOW	IC_SS_SCL_LCNT	IC_FS_SCL_LCNT	IC_HS_SCL_LCNT	IC_HS_SCL_LCNT

Timing Parameter	Symbol	Standard Speed	Fast Speed / Fast Speed Plus	High Speed (100 pf)	High Speed (400 pf)
HIGH period of the SCL clock	tHIGH	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT	IC_HS_SCL_HCNT	IC_HS_SCL_HCNT
Setup time for a repeated START condition	tSU;STA	IC_SS_SCL_LCNT	IC_FS_SCL_HCNT	IC_HS_SCL_LCNT	(IC_HS_SCL_LCNT)/2
Hold time (repeated) START condition*	tHD;STA	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT	IC_HS_SCL_LCNT	(IC_HS_SCL_LCNT)/2
Setup time for STOP condition	tSU;STO	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT	IC_HS_SCL_LCNT	(IC_HS_SCL_LCNT)/2
Bus free time between a STOP and a START condition	tBUF	IC_SS_SCL_LCNT	IC_FS_SCL_LCNT	NA	NA
Spike length	tSP	IC_FS_SPKLEN	IC_FS_SPKLEN	IC_HS_SPKLEN	IC_HS_SPKLEN
Data hold time	tHD;DAT	IC_SDA_HOLD	IC_SDA_HOLD	IC_SDA_HOLD	IC_SDA_HOLD
Data setup time	tSU;DAT	IC_SDA_SETUP	IC_SDA_SETUP	IC_SDA_SETUP	IC_SDA_SETUP

2.15.1 Minimum High and Low Counts in SS, FS, FM+ and high speed Modes With IC_CLK_FREQ_OPTIMIZATION = 0.

When the DW_apb_i2c operates as an I2C master, in both transmit and receive transfers:

- IC_SS_SCL_LCNT and IC_FS_SCL_LCNT register values must be larger than *IC_FS_SPKLEN* + 7.
- IC_SS_SCL_HCNT and IC_FS_SCL_HCNT register values must be larger than *IC_FS_SPKLEN* + 5.
- If the component is programmed to support high speed, IC_HS_SCL_LCNT register value must be larger than *IC_HS_SPKLEN* + 7.
- If the component is programmed to support high speed, IC_HS_SCL_HCNT register value must be larger than *IC_HS_SPKLEN* + 5.

Details regarding the DW_apb_i2c high and low counts are as follows:

- The minimum value of *IC_*_SPKLEN* + 7 for the *_LCNT registers is due to the time required for the DW_apb_i2c to drive SDA after a negative edge of SCL.
- The minimum value of *IC_*_SPKLEN* + 5 for the *_HCNT registers is due to the time required for the DW_apb_i2c to sample SDA during the high period of SCL.
- The DW_apb_i2c adds one cycle to the programmed *_LCNT value in order to generate the low period of the SCL clock; this is due to the counting logic for SCL low counting to (**_LCNT* + 1).

- The DW_apb_i2c adds *IC_*_SPKLEN* + 7 cycles to the programmed *_HCNT value in order to generate the high period of the SCL clock; this is due to the following factors:
  - The counting logic for SCL high counts to (**_HCNT*+1).
  - The digital filtering applied to the SCL line incurs a delay of *SPKLEN* + 2 ic_clk cycles, where SPKLEN is:
    - *IC_FS_SPKLEN* if the component is operating in SS or FS
    - *IC_HS_SPKLEN* if the component is operating in high speed.

    This filtering includes metastability removal and the programmable spike suppression on SDA and SCL edges.
  - Whenever SCL is driven 1 to 0 by the DW_apb_i2c—that is, completing the SCL high time—an internal logic latency of three ic_clk cycles is incurred. Consequently, the minimum SCL low time of which the DW_apb_i2c is capable is nine (9) ic_clk periods (7 + 1 + 1), while the minimum SCL high time is thirteen (13) ic_clk periods (6 + 1 + 3 + 3).

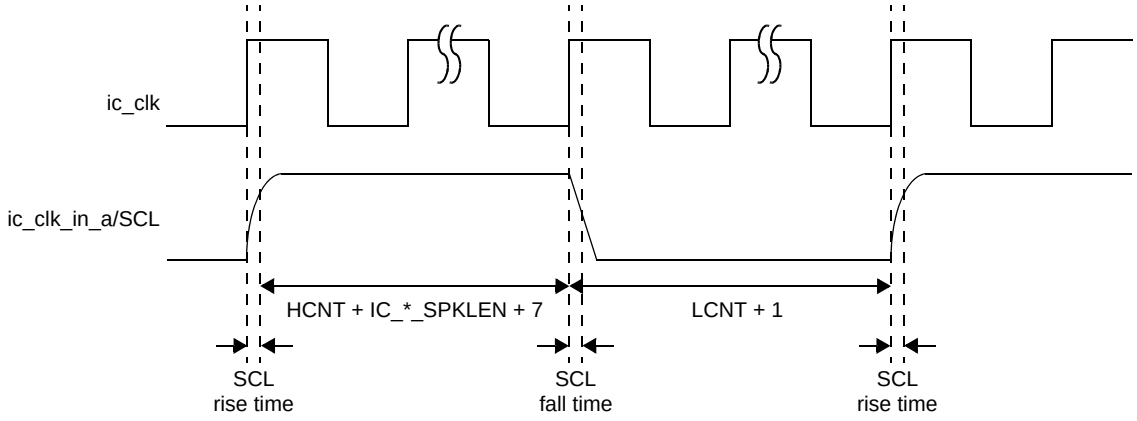
Note

The total high time and low time of SCL generated by the DW_apb_i2c master is also influenced by the rise time and fall time of the SCL line, as shown in the illustration and equations in Figure 2-30. It should be noted that the SCL rise and fall time parameters vary, depending on external factors such as:

- Characteristics of IO driver
- Pull-up resister value
- Total capacitance on SCL line, and so on

These characteristics are beyond the control of the DW_apb_i2c.

Figure 2-30 Impact of SCL Rise Time and Fall Time on Generated SCL



SCL_High_time = [(HCNT + IC_*_SPKLEN + 7) * ic_clk] + SCL_Fall_time
SCL_Low_time = [(LCNT + 1) * ic_clk] - SCL_Fall_time + SCL_Rise_time

2.15.2 Minimum High and Low Counts in SS, FS, FM+ and high speed Modes With IC_CLK_FREQ_OPTIMIZATION = 1

The minimum high and low counts in SS, FS, FM+ and high speed Modes with the IC_CLK_FREQ_OPTIMIZATION parameter set to one is such that:

- The total SCL LOW period is driven by DW_apb_i2c is IC_*_LCNT register value. The hardware does not support a value less than 6 to be written to the IC_*_LCNT register. Additionally, the minimum SCL low time of which the DW_apb_i2c is capable is 6 ic_clk periods.
- The total SCL HIGH period driven by DW_apb_i2c is IC_*_HCNT register value + SPKLEN + 3. Additionally, the minimum SCL high time of which the DW_apb_i2c is capable is 5 ic_clk periods [1+1+3].

The total high time and low time of SCL generated by the DW_apb_i2c master is also influenced by the rise time and fall time of the SCL line. The SCL rise and fall time parameters vary depending on external factors such as:

- Characteristics of IO driver
- Pull-up resister value
- Total capacitance on SCL line, and so on

These characteristics are beyond the control of the DW_apb_i2c.

2.15.3 Minimum High and Low counts in Ultra-Fast mode (IC_ULTRA_FAST_MODE = 1)

When the DW_apb_i2c operates as an I2C master:

- The IC_UFM_SCL_HCNT register value must be equal or larger than 3.
- The IC_UFM_SCL_LCNT register Value must be equal or larger than 5.

2.15.4 Minimum IC_CLK Frequency

This section describes the minimum ic_clk frequencies that the DW_apb_i2c supports for each speed mode, and the associated high and low count values. In Slave mode, IC_SDA_HOLD (Thd;dat) and IC_SDA_SETUP (Tsu:dat) need to be programmed to satisfy the I2C protocol timing requirements.

The following examples are for the case where IC_FS_SPKLEN and IC_HS_SPKLEN are programmed to 2.

2.15.4.1 Standard Mode (SM), Fast Mode (FM), and Fast Mode Plus (FM+) with IC_CLK_FREQ_OPTIMIZATION = 0

This section details how to derive a minimum ic_clk value for standard and fast modes of the DW_apb_i2c. Although the following method shows how to do fast mode calculations, you can also use the same method in order to do calculations for standard mode and fast mode plus.

Note The following computations do not consider the SCL_Rise_time and SCL_Fall_time.

Given conditions and calculations for the minimum DW_apb_i2c ic_clk value in fast mode:

- Fast mode has data rate of 400kb/s; implies SCL period of 1/400khz = 2.5us
- Minimum hcnt value of 14 as a seed value; IC_HCNT_FS = 14
- Protocol minimum SCL high and low times:
  - MIN_SCL_LOWtime_FS = 1300ns
  - MIN_SCL_HIGHtime_FS = 600ns

Derived equations:

$$\frac{SCL\_PERIOD\_FS}{IC\_HCNT\_FS + IC\_LCNT\_FS} = IC\_CLK\_PERIOD$$

$$IC\_LCNT\_FS \times IC\_CLK\_PERIOD = MIN\_SCL\_LOWtime\_FS$$

Combined, the previous equations produce the following:

$$IC\_LCNT\_FS \times \frac{SCL\_PERIOD\_FS}{IC\_LCNT\_FS + IC\_HCNT\_FS} = MIN\_SCL\_LOWtime\_FS$$

Solving for IC_LCNT_FS:

$$IC\_LCNT\_FS \times \frac{2.5\mu s}{IC\_LCNT\_FS + 14} = 1.3\mu s$$

The previous equation gives:

$$IC\_LCNT\_FS = roundup(15.166) = 16$$

These calculations produce IC_LCNT_FS = 16 and IC_HCNT_FS = 14, giving an ic_clk value of:

$$\frac{2.5\ \mu s}{16 + 14} = 83.3ns = 12Mhz$$

Testing these results shows that protocol requirements are satisfied.

2.15.4.2 High-Speed (HS) Mode With IC_CLK_FREQ_OPTIMIZATION = 0

The method used for standard and fast modes can also be used to derive ic_clk values for high speed modes. For example, given a high speed mode with a 100pf bus loading, using the standard and fast modes method produces the following:

- IC_LCNT_HS = 17
- IC_HCNT_HS = 14
- ic_clk = 105.4 Mhz

Table 2-5 lists the minimum ic_clk values for all modes with high and low count values.

Table 2-5 ic_clk in Relation to High and Low Counts When IC_CLK_FREQ_OPTIMIZATION = 0

Speed Mode	ic_clk_{freq} (MHz)	Minimum Value of IC_*_SPKLEN	SCL Low Time in ic_clks	SCL Low Program Value	SCL Low Time	SCL High Time in ic_clks	SCL High Program Value	SCL High Time
SS	2.7	1	13	12	4.7 µs	14	6	5.2 µs
FS	12.0	1	16	15	1.33 µs	14	6	1.16 µs
FM+	32	2	16	15	500 ns	16	7	500 ns
HS (400pf)	51	1	17	16	333 ns	14	6	274 ns
HS (100pf)	105.4	1	17	16	161 ns	14	6	132 ns

Note

- The IC_*_SCL_LCNT and IC_*_SCL_HCNT registers are programmed using the SCL low and high program values in Table 2-5, which are calculated using SCL low count minus 1, and SCL high counts minus 8, respectively.

  The values in Table 2-5 are based on IC_SDA_RX_HOLD = 0. The maximum IC_SDA_RX_HOLD value depends on the IC_*CNT registers in Master mode, as described in "SDA Hold Timings in Receiver" on page 78.
- In order to compute the HCNT and LCNT considering RC timings, use the following equations:

  IC_HCNT_* = [(HCNT + IC_*_SPKLEN + 7) * ic_clk] + SCL_Fall_time

  IC_LCNT_* = [(LCNT + 1) * ic_clk] - SCL_Fall_time + SCL_Rise_time

2.15.4.3 SM, FM, FM+ and high speed Modes With IC_CLK_FREQ_OPTIMIZATION = 1

2.15.4.3.1 Master Mode

This section describes the minimum ic_clk frequencies that the DW_apb_i2c supports for each speed mode and the associated high and low count values. The following examples are for the case where IC_FS_SPKLEN = 1, IC_HS_SPKLEN = 1 and IC_CLK_FREQ_OPTIMIZATION = 1.

Following calculations show how to derive a minimum ic_clk value for fast mode of the DW_apb_i2c. Although the following method shows how to do fast mode calculations, you can also use the same method in order to do calculations for any speed mode.

Note The computation in this section does not consider SCL_Rise_time and SCL_Fall_time.

Following are the conditions and calculations for the minimum DW_apb_i2c ic_clk value in fast mode:

- Fast mode has data rate of 400kb/s; implies SCL period of 1/400KHz = 2.5 us
- Minimum hcnt value of 5 as a seed value; IC_HCNT_FS = 5
- Protocol minimum SCL high and low times:

- MIN_SCL_LOWtime_FS = 1300 ns
- MIN_SCL_HIGHtime_FS = 600 ns

Following are the derived equations:

SCL_PERIOD_FS / (IC_HCNT_FS + IC_LCNT_FS) = IC_CLK_PERIOD

IC_LCNT_FS × IC_CLK_PERIOD = MIN_SCL_LOWtime_FS

Following is the result of combining previous equations:

IC_LCNT_FS × SCL_PERIOD_FS / (IC_LCNT_FS + IC_HCNT_FS) = MIN_SCL_LOWtime_FS

By solving for IC_LCNT_FS:

IC_LCNT_FS × 2.5 µs / (IC_LCNT_FS + 5) = 1.3 µs

The previous equation provides:

IC_LCNT_FS = roundup(5.417) = 6

Note Minimum IC_*_LCNT value should be equal 6. If derived value is less than 6, consider IC_LCNT_FS as 6 only.

These calculations produce IC_LCNT_FS = 6 and IC_HCNT_FS = 5, providing an ic_clk value of:

2.5 µs / (6 + 5) = 227.27ns = 4.4 MHz

Testing these results shows that the protocol requirements are satisfied.

Table 2-6 lists the minimum ic_clk values for all modes with high and low count values.

Table 2-6 ic_clk in Relation to High and Low Counts When IC_CLK_FREQ_OPTIMIZATION = 1

Speed Mode	ic_clk Frequency (MHz)	Minimum Value of IC_*_SPKLEN	SCL Low Time in ic_clks	SCL Low Program Value	SCL Low Time in ns	SCL High Time in ic_clks	SCL High Program Value	SCL High Time in ns
SS	1.1	1	6	6	5454.545	5	1	4545.455
FS	4.4	1	6	6	1363.636	5	1	1136.364
FM+	11	1	6	6	545.4545	5	1	454.5455
HS (400pf)	18.7	1	6	6	320.8527	5	1	267.3773
HS (100pf)	37.4	1	6	6	160.4236	5	1	133.6864

Note

- The IC_*_SCL_LCNT and IC_*_SCL_HCNT registers are programmed using the SCL low and high program values in Table 2-6, which are calculated as SCL low count, and SCL high count minus 4, respectively. The values in Table 2-6 are based on IC_SDA_RX_HOLD = 0. The maximum IC_SDA_RX_HOLD value depends on the IC_*CNT registers in master mode, as described in "SDA Hold Timings in Receiver" on page 78.
- To compute the HCNT and LCNT considering RC timings, use the following equations:

  IC_HCNT_* = [(HCNT + IC_*_SPKLEN + 3) * ic_clk] + SCL_Fall_time

  IC_LCNT_* = [LCNT * ic_clk] - SCL_Fall_time + SCL_Rise_time

2.15.4.3.2 Slave Mode

DW_apb_i2c in slave mode requires minimum 5 ic_clk cycles [SPKLEN + 3 (Metastability removal, worst case) + 1] to drive SDA after a falling edge of SCL. Therefore, the ic_clk frequency must be selected such that the maximum data hold time (thd;dat) / data valid time (tVD;DAT) is not violated.

For example, in high speed mode with a 100pf bus loading (SCLH clock frequency upto 3.4 MHz), the maximum data hold time is 70 ns. Therefore, the minimum frequency in which DW_apb_i2c can operate in slave mode without violating thd;dat is 70ns / 5 = 14ns = 71.42 MHz.

Table 2-7 lists the minimum IC_CLK frequency in slave mode when IC_CLK_FREQ_OPTIMIZATION is set to 1.

Table 2-7 Minimum IC_CLK Frequency in Slave Mode When IC_CLK_FREQ_OPTIMIZATION=1

Speed Mode	ic_clk Frequency (MHz)	Minimum Value of IC_*_SPKLEN	Minimum data hold time in ic_clks	Maximum data hold time
SS	1.45	1	5	3.45 µs
FS	5.56	1	5	0.9 µs
FM+	11.11	1	5	0.45 µs
HS (400pf)	35.71	1	5	140 ns
HS (100pf)	71.42	1	5	70 ns

2.15.4.4 ULTRA-FAST Mode

2.15.4.4.1 Master Mode

This section describes the minimum ic_clk frequency that the DW_apb_i2c supports for Ultra-Fast speed mode and the associated high and low count values.

The following calculations show how to derive a minimum ic_clk value.

Note The following computations do not consider the SCL_Rise_time and SCL_Fall_time.

Given conditions and calculations for the minimum DW_apb_i2c ic_clk value in Ultra-Fast mode:

- Fast mode has data rate of 5000kb/s; implies SCL period of 1/5000khz = 200ns
- Minimum hcnt value of 3 as a seed value; IC_UFM_SCL_HCNT = 3
- Protocol minimum SCL high and low times:
  - MIN_SCL_LOWtime_UFm = 50 ns
  - MIN_SCL_HIGHtime_UFm = 50ns

Derived equations:

- SCL_PERIOD_UFm/(IC_HCNT_UFm + IC_LCNT_UFm) = IC_CLK_PERIOD
- IC_LCNT_UFm × IC_CLK_PERIOD = MIN_SCL_LOWtime_UFm

Combined, the previous equations produce the following:

IC_LCNT_UFm × SCL_PERIOD_UFm /(IC_LCNT_UFm + IC_HCNT_UFm) = MIN_SCL_LOWtime_UFm

Solving for IC_LCNT_UFm:

IC_LCNT_UFm × 200ns /(IC_LCNT_UFm + 3) = 50ns

The previous equation gives:

IC_LCNT_UFm = 1

Note Minimum IC_SCL_UFM_LCNT value should be equal 5. If derived value is less than 5, consider IC_LCNT_UFm as 5 only.

These calculations produce IC_LCNT_UFm = 5 and IC_HCNT_UFm = 3, giving an ic_clk value of:

200 ns/(5 + 3) = 25ns = 40Mhz

Testing these results shows that protocol requirements are satisfied.

Table 2-8 describes the relation between the High and Low counts with ic_clk frequency

Table 2-8 ic_clk in relation to High and Low Counts when IC_ULTRA_FAST_MODE=1

Speed	ic_clk (freq) (Mhz)	SCL Low Program Value	SCL Low Time in ic_clks	SCL Low Time	SCL High Program Value	SCL HighTime in ic_clks	SCL HighTime
UltraFast Mode	40	5	5	125 ns	3	3	75 ns

Note

- The IC_UFM_SCL_LCNT and IC_UFM_SCL_HCNT registers are programmed using the SCL low and high program values in Table 2-8, which are calculated as SCL low count, and SCL high count, respectively. The values in Table 2-8 are based on IC_SDA_RX_HOLD = 0. The maximum IC_SDA_RX_HOLD value depends on the IC_UFM_SCL_LCNT registers in Master mode, as described in "SDA Hold Timings in Receiver" on page 78.
- In order to compute the HCNT and LCNT considering RC timings, use the following equations:

  IC_UFM_SCL_HCNT = [HCNT * ic_clk] + SCL_Fall_time

  IC_UFM_SCL_LCNT = [LCNT * ic_clk] - SCL_Fall_time + SCL_Rise_time

2.15.4.4.2 Slave mode

DW_apb_i2c in slave mode requires minimum of 2 ic_clk cycles for SCL High period and SCL Low Period. Therefore, the minimum ic_clk frequency for the slave mode is 40 MHz.

2.15.4.5 Calculating High and Low Counts with IC_CLK_FREQ_OPTIMIZATION = 0

The following calculations show how to calculate SCL high and low counts for each speed mode in the DW_apb_i2c. For the calculations to work, the ic_clk frequencies used must not be less than the minimum ic_clk frequencies specified in Table 2-5.

The DW_apb_i2c coreConsultant GUI can automatically calculate SCL high and low count values. By specifying an integer ic_clk period value in nanoseconds for the IC_CLK_PERIOD parameter, SCL high and low count values are automatically calculated for each speed mode. The ic_clk period must not specify a clock of a lower frequency than required for all supported speed modes. It is possible that the automatically calculated values may result in a baud rate higher than the maximum rate specified by the protocol. If this happens, either the low or high count values can be scaled up to reduce the baud rate.

The equation to calculate the proper number of ic_clk signals required for setting the proper SCL clocks high and low times is as follows:

```
IC_xCNT = (ROUNDUP(MIN_SCL_xxxtime*OSCFREQ,0))
  ROUNDUP is an explicit Excel function call that is used to convert a real number to its
equivalent integer number.
  MIN_SCL_HIGHtime = Minimum High Period
    MIN_SCL_HIGHtime = 4000 ns for 100 kbps
                        600 ns for 400 kbps
                        260 ns for 1000 kbps
                        60 ns for 3.4 Mbs, bus loading = 100pF
                        120 ns for 3.4 Mbs, bus loading = 400pF

  MIN_SCL_LOWtime = Minimum Low Period
    MIN_SCL_LOWtime = 4700 ns for 100 kbps
                      1300 ns for 400 kbps
                       500 ns for 1000 kbps
                      160 ns for 3.4Mbs, bus loading = 100pF
                      320 ns for 3.4Mbs, bus loading = 400pF

OSCFREQ = ic_clk Clock Frequency (Hz).
```

For example:

```
                OSCFREQ = 100 MHz
                I2Cmode = fast, 400 kbit/s
                MIN_SCL_HIGHtime = 600 ns.
                MIN_SCL_LOWtime = 1300 ns.

IC_xCNT = (ROUNDUP(MIN_SCL_HIGH_LOWtime*OSCFREQ,0))

IC_HCNT = (ROUNDUP(600 ns * 100 MHz,0))
IC_HCNTSCL PERIOD = 60
IC_LCNT = (ROUNDUP(1300 ns * 100 MHz,0))
IC_LCNTSCL PERIOD = 130
Actual MIN_SCL_HIGHtime = 60*(1/100 MHz) = 600 ns
Actual MIN_SCL_LOWtime = 130*(1/100 MHz) = 1300 ns
```

Note Once the default values for SCL HighCount and LowCount are computed by the coreConsultant GUI, check that the values are consistent with the required baud rate. In case the computed values do not match with the required values, you can manually scale the values, as described in the section “High-Speed (HS) Mode With IC_CLK_FREQ_OPTIMIZATION = 0” on page 70.

2.15.4.6 Calculating High and Low counts with IC_CLK_FREQ_OPTIMIZATION = 1

The following calculations show how to calculate SCL high and low counts for each speed mode in the DW_apb_i2c. For the calculations to work, the ic_clk frequencies used must not be less than the minimum ic_clk frequencies specified in Table 2-6.

The DW_apb_i2c coreConsultant GUI can automatically calculate SCL high and low count values. By specifying an integer ic_clk period value in nanoseconds for the IC_CLK_PERIOD parameter, SCL high and low count values are automatically calculated for each speed mode. The ic_clk period must not specify a clock of a lower frequency than required for all supported speed modes. It is possible that the automatically calculated values may result in a baud rate higher than the maximum rate specified by the protocol. If this happens, either the low or high count values can be scaled up to reduce the baud rate. For more information, see “Master Mode” on page 71.

The equation to calculate the proper number of ic_clk signals required for setting the proper SCL clocks high and low times is as follows:

```
IC_xCNT = (ROUNDUP(MIN_SCL_xxxtime*OSCFREQ,0))
  ROUNDUP is an explicit Excel function call that is used to convert a real number to its
equivalent integer number.
  MIN_SCL_HIGHtime = Minimum High Period
    MIN_SCL_HIGHtime = 4000 ns for 100 kbps
                        600 ns for 400 kbps
                        260 ns for 1000 kbps
                         60 ns for 3.4 Mbps, bus loading = 100pF
                        160 ns for 3.4 Mbps, bus loading = 400pF
  MIN_SCL_LOWtime = Minimum Low Period
    MIN_SCL_LOWtime = 4700 ns for 100 kbps
                      1300 ns for 400 kbps
                       500 ns for 1000 kbps
                       120 ns for 3.4Mbps, bus loading = 100pF
                       320 ns for 3.4Mbps, bus loading = 400pF
OSCFREQ = ic_clk Clock Frequency (Hz).
```

For example:

```
                OSCFREQ = 100 MHz
                I2Cmode = fast, 400 kbit/s
                MIN_SCL_HIGHtime = 600 ns.
                MIN_SCL_LOWtime = 1300 ns.

IC_xCNT = (ROUNDUP(MIN_SCL_HIGH_LOWtime*OSCFREQ,0))

IC_HCNT = (ROUNDUP(600 ns * 100 MHz,0))
IC_HCNTSCL PERIOD = 60
IC_LCNT = (ROUNDUP(1300 ns * 100 MHz,0))
IC_LCNTSCL PERIOD = 130
Actual MIN_SCL_HIGHtime = 60*(1/100 MHz) = 600 ns
Actual MIN_SCL_LOWtime = 130*(1/100 MHz) = 1300 ns
```

Note When the default values for SCL HighCount and LowCount are computed by the coreConsultant GUI, check that the values are consistent with the required baud rate. In case the computed values do not match with the required values, you can manually scale the values, as described in "Master Mode" on page 73.

2.16 SDA Hold Time

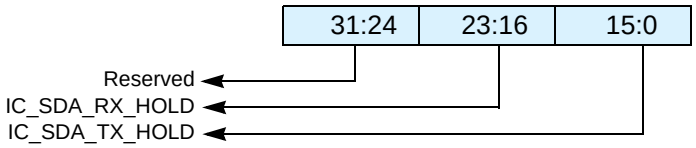
The I^2C protocol specification requires 300ns of hold time on the SDA signal (tHD;DAT) in standard mode and fast mode, and a hold time long enough to bridge the undefined part between logic 1 and logic 0 of the falling edge of SCL in high speed mode and fast mode plus.

Board delays on the SCL and SDA signals can mean that the hold-time requirement is met at the I^2C master, but not at the I^2C slave (or vice-versa). As each application encounters differing board delays, the DW_apb_i2c contains a software programmable register (IC_SDA_HOLD) to enable dynamic adjustment of the SDA hold-time.

The bits [15:0] are used to control the hold time of SDA during transmit in both slave and master mode (after SCL goes from HIGH to LOW).

The bits [23:16] are used to extend the SDA transition (if any) whenever SCL is HIGH in the receiver (in either master or slave mode).

Figure 2-31 IC_SDA_HOLD Register



If different SDA hold times are required for different speed modes, the IC_SDA_HOLD register must be reprogrammed when the speed mode is being changed. The IC_SDA_HOLD register can be programmed only when the DW_apb_i2c is disabled (IC_ENABLE[0] = 0).

The reset value of the IC_SDA_HOLD register can be set through the coreConsultant parameter IC_DEFAULT_SDA_HOLD

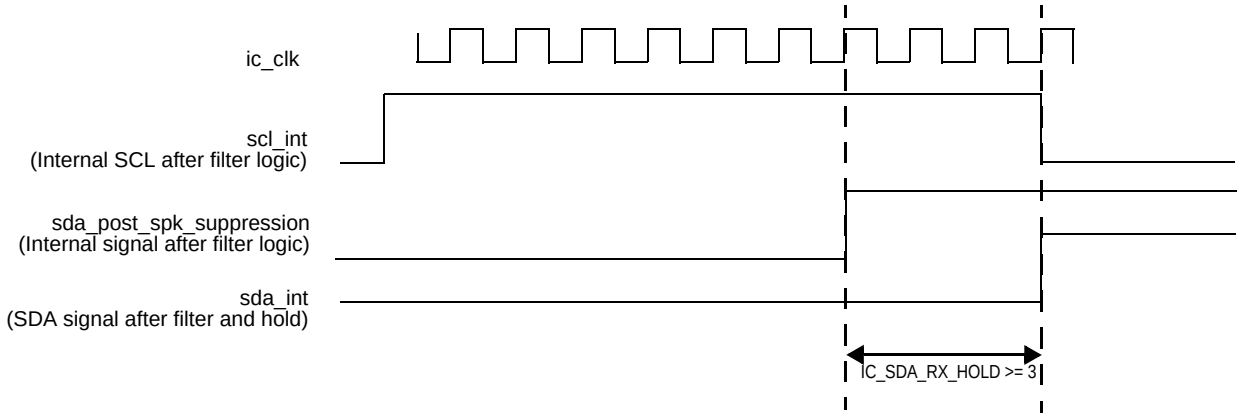
2.16.1 SDA Hold Timings in Receiver

When DW_apb_i2c acts as a receiver, according to the I^2C protocol, the device should internally hold the SDA line to bridge undefined gap between logic 1 and logic 0 of SCL.

IC_SDA_RX_HOLD can be used to alter the internal hold time which DW_apb_i2c applies to the incoming SDA line. Each value in the IC_SDA_RX_HOLD register represents a unit of one ic_clk period. The minimum value of IC_SDA_RX_HOLD is 0. This hold time is applicable only when SCL is HIGH. The receiver does not extend the SDA after SCL goes LOW internally.

Figure 2-32 shows the DW_apb_i2c as receiver with IC_SDA_RX_HOLD programmed to greater than or equal to 3.

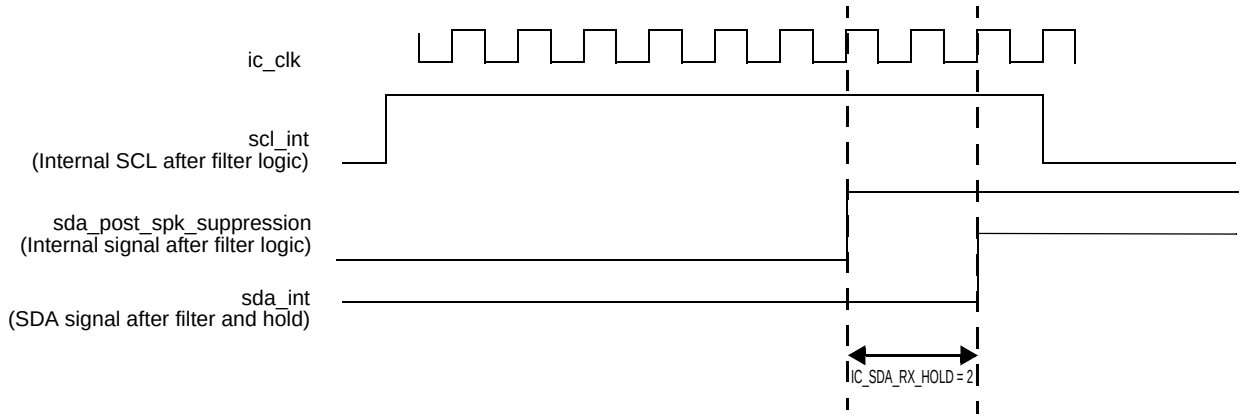
Figure 2-32 DW_apb_i2c as Receiver With IC_SDA_RX_HOLD Programmed to Greater Than or Equal to 3



If IC_SDA_RX_HOLD is greater than 3, DW_apb_i2c does not hold SDA beyond 3 ic_clk cycles, because SCL goes LOW internally.

Figure 2-33 shows the DW_apb_i2c as receiver with IC_SDA_RX_HOLD programmed to 2.

Figure 2-33 DW_apb_i2c as Receiver With IC_SDA_RX_HOLD Programmed to 2



The maximum values of IC_SDA_RX_HOLD that can be programmed in the register for the respective speed modes are derived from the equations show in Table 2-9.

Table 2-9 Maximum Values for IC_SDA_RX_HOLD

Speed Mode	Maximum IC_SDA_RX_HOLD Value
Standard Mode	IC_SS_SCL_HCNT – IC_FS_SPKLEN – 3
Fast Mode or Fast Mode Plus	IC_FS_SCL_HCNT – IC_FS_SPKLEN – 3
High Speed (IC_CAP_LOADING =100)	Min {IC_FS_SCL_HCNT – IC_FS_SPKLEN – 3, IC_HS_SCL_LCNT – IC_HS_SPKLEN – 3}
High Speed (IC_CAP_LOADING =400)	Min {IC_FS_SCL_HCNT – IC_FS_SPKLEN – 3, (IC_HS_SCL_LCNT/2) – IC_HS_SPKLEN – 3}

Note The maximum values in Table 2-9 is applicable in Master mode. In Slave mode, make sure the IC_SDA_RX_HOLD does not exceed the maximum SCL fall time (tf in SS and FS mode or tfcl in HS Mode).

2.16.2 SDA Hold Timings in Transmitter

The IC_SDA_TX_HOLD register can be used to alter the timing of the generated SDA (ic_data_oe) signal by the DW_apb_i2c. Each value in the IC_SDA_TX_HOLD register represents a unit of one ic_clk period.

When the DW_apb_i2c is operating in Master Mode, the minimum tHD:DAT timing is one ic_clk period. Therefore even when IC_SDA_TX_HOLD has a value of zero, the DW_apb_i2c drives SDA (ic_data_oe) one ic_clk cycle after driving SCL (ic_clk_oe) to logic 0. For all other values of IC_SDA_TX_HOLD, the following is true:

- Drive on SDA (ic_data_oe) occurs *IC_SDA_TX_HOLD* ic_clk cycles after driving SCL (ic_clk_oe) to logic 0

When the DW_apb_i2c is operating in Slave Mode, the minimum tHD:DAT timing is *SPKLEN* + 7 ic_clk periods, where SPKLEN is:

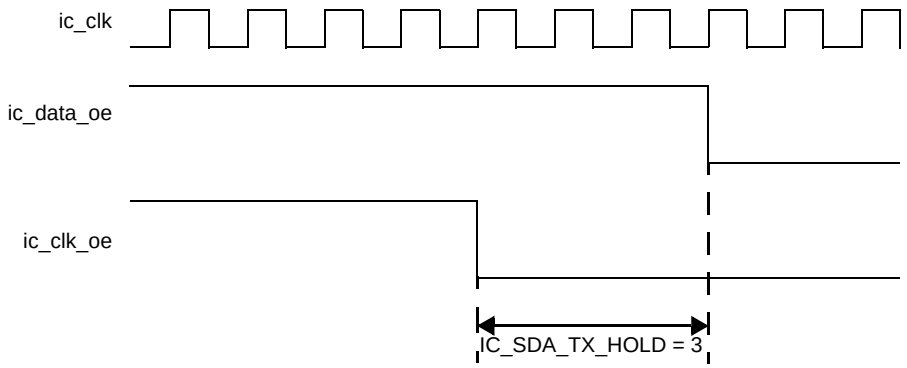
- IC_FS_SPKLEN if the component is operating in standard mode, fast mode, or fast mode plus
- IC_HS_SPKLEN if the component is operating in high speed mode

This delay allows for synchronization and spike suppression on the SCL (ic_clk_in_a) sample. Therefore, even when IC_SDA_TX_HOLD has a value less than *SPKLEN* + 7, the DW_apb_i2c drives SDA (ic_data_oe) *SPKLEN* + 7 ic_clk cycles after SCL (ic_clk_in) has transitioned to logic 0. For all other values of IC_SDA_TX_HOLD, the following is true:

- Drive on SDA (ic_data_oe) occurs *IC_SDA_TX_HOLD* ic_clk cycles after SCL (ic_clk_in_a) has transitioned to logic 0.

Figure 2-34 shows the tHD:DAT timing generated by the DW_apb_i2c operating in Master Mode when IC_SDA_TX_HOLD = 3.

Figure 2-34 DW_apb_i2c Master Implementing tHD;DAT with IC_SDA_HOLD = 3



Note The programmed SDA hold time cannot exceed at any time the duration of the low part of scl. Therefore the programmed value cannot be larger than N_SCL_LOW-2, where N_SCL_LOW is the duration of the low part of the scl period measured in ic_clk cycles.

2.17 DMA Controller Interface

The DW_apb_i2c has an optional built-in DMA capability that can be selected at configuration time; it has a handshaking interface to a DMA Controller to request and control transfers. The APB bus is used to perform the data transfer to or from the DMA. While the DW_apb_i2c DMA operation is designed in a generic way to fit any DMA controller as easily as possible, it is designed to work seamlessly, and best used, with the DesignWare DMA Controller, the DW_ahb_dmac. The settings of the DW_ahb_dmac that are relevant to the operation of the DW_apb_i2c are discussed here, mainly bit fields in the DW_ahb_dmac channel control register, CTL*x*, where *x* is the channel number.

Note	When the DW_apb_i2c 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 relevant DMA settings are discussed in the following sections.

Note	The DMA output dma_finish is a status signal to indicate that the DMA block transfer is complete. DW_apb_i2c does not use this status signal, and therefore does not appear in the I/O port list.

2.17.1 Enabling the DMA Controller Interface

To enable the DMA Controller interface on the DW_apb_i2c, you must write the DMA Control Register (IC_DMA_CR). Writing a 1 into the TDMAE bit field of IC_DMA_CR register enables the DW_apb_i2c transmit handshaking interface. Writing a 1 into the RDMAE bit field of the IC_DMA_CR register enables the DW_apb_i2c receive handshaking interface.

2.17.2 Overview of Operation

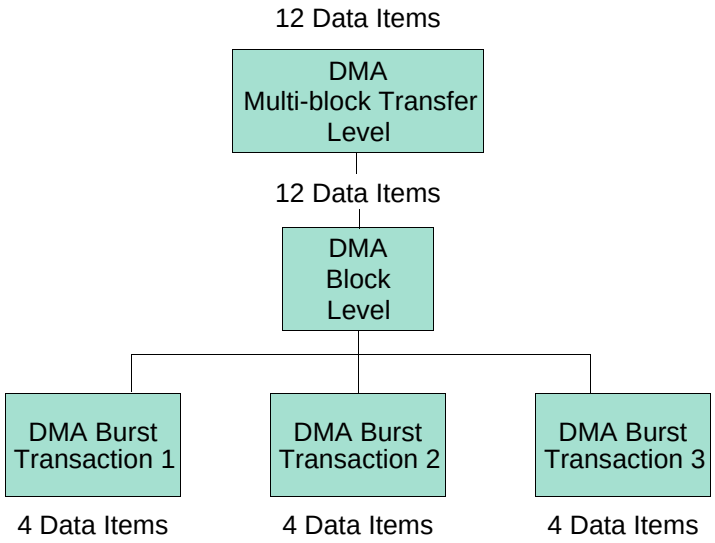
As a block flow control device, the DMA Controller is programmed by the processor with the number of data items (block size) that are to be transmitted or received by DW_apb_i2c; this is programmed into the BLOCK_TS field of the DW_ahb_dmac CTL*x* register.

The block is broken into a number of transactions, each initiated by a request from the DW_apb_i2c. The DMA Controller must also be programmed with the number of data items (in this case, DW_apb_i2c FIFO entries) to be transferred for each DMA request. This is also known as the burst transaction length and is programmed into the SRC_MSIZE / DEST_MSIZE fields of the DW_ahb_dmac CTL*x* register for source and destination, respectively.

Figure 2-35 shows a single block transfer, where the block size programmed into the DMA Controller is 12 and the burst transaction length is set to 4. In this case, the block size is a multiple of the burst transaction length. Therefore, the DMA block transfer consists of a series of burst transactions. If the DW_apb_i2c makes a transmit request to this channel, four data items are written to the DW_apb_i2c TX FIFO. Similarly, if the DW_apb_i2c makes a receive request to this channel, four data items are read from the DW_apb_i2c

RX FIFO. Three separate requests must be made to this DMA channel before all 12 data items are written or read.

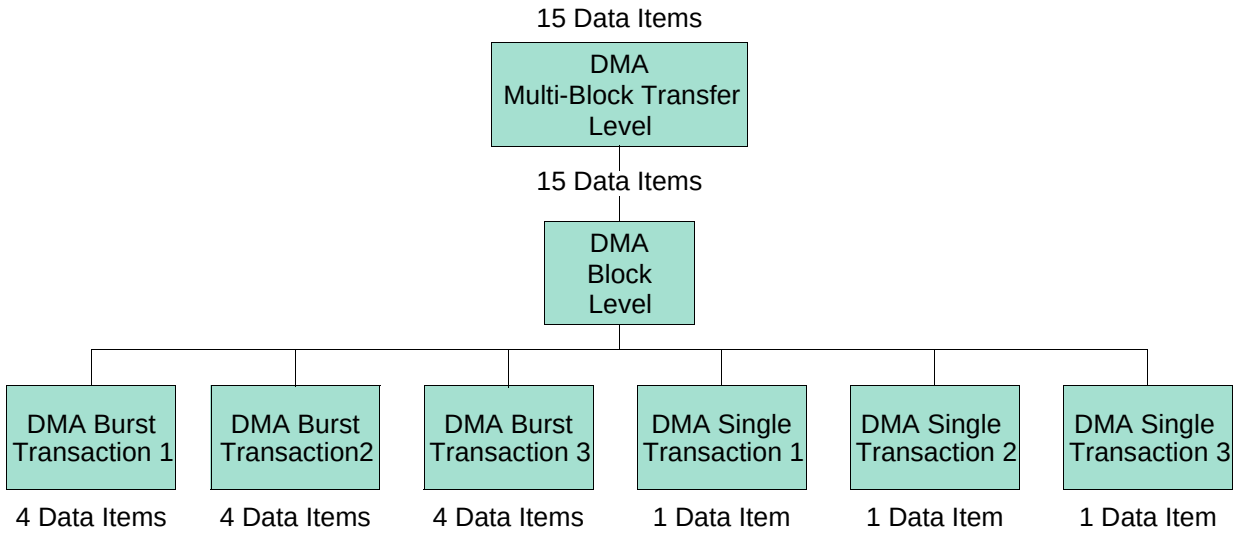
Figure 2-35 Breakdown of DMA Transfer into Burst Transactions



Block Size: DMA.CTLx.BLOCK_TS=12
Number of data items per source burst transaction: DMA.CTLx.SRC_MSIZE = 4
I^2C receive FIFO watermark level: I2C.DMARDLR + 1 = DMA.CTLx.SRC_MSIZE = 4
(for more information, see discussion on page 86)

When the block size programmed into the DMA Controller is not a multiple of the burst transaction length, as shown in Figure 2-36, a series of burst transactions followed by single transactions are needed to complete the block transfer.

Figure 2-36 Breakdown of DMA Transfer into Single and Burst Transactions



Block Size: DMA.CTLx.BLOCK_TS=15
Number of data items per burst transaction: DMA.CTLx.DEST_MSIZE = 4
I^2C transmit FIFO watermark level: I2C.IC_DMA_TDLR = DMA.CTLx.DEST_MSIZE = 4
(for more information, see discussion on page 85)

2.17.3 Transmit Watermark Level and Transmit FIFO Underflow

During DW_apb_i2c 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 (IC_DMA_TDLR) value; this is known as the watermark level. The DW_ahb_dmac responds by writing a burst of data to the transmit FIFO buffer, of length CTL*x*.DEST_MSIZE.

If IC_EMPTYFIFO_HOLD_MASTER_EN parameter is set to 0, data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously; that is, when the FIFO begins to empty another DMA request should be triggered. Otherwise, the FIFO runs out of data causing a STOP to be inserted on the I^2C bus. To prevent this condition, you must set the watermark level correctly.

2.17.4 Choosing the Transmit Watermark Level

Consider the example where the assumption is made:

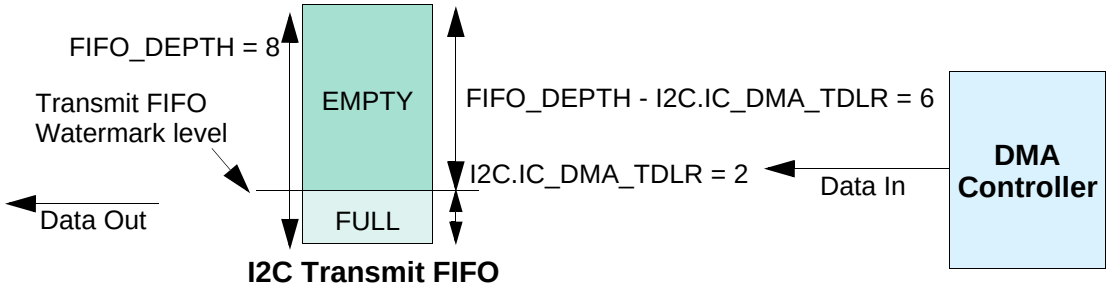
DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - I2C.IC_DMA_TDLR

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

2.17.4.1 Case 1: IC_DMA_TDLR = 2

- Transmit FIFO watermark level = I2C.IC_DMA_TDLR = 2
- DMA.CTL*x*.DEST_MSIZE = FIFO_DEPTH - I2C.IC_DMA_TDLR = 6
- I2C transmit FIFO_DEPTH = 8
- DMA.CTL*x*.BLOCK_TS = 30

Figure 2-37 Case 1 Watermark Levels



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

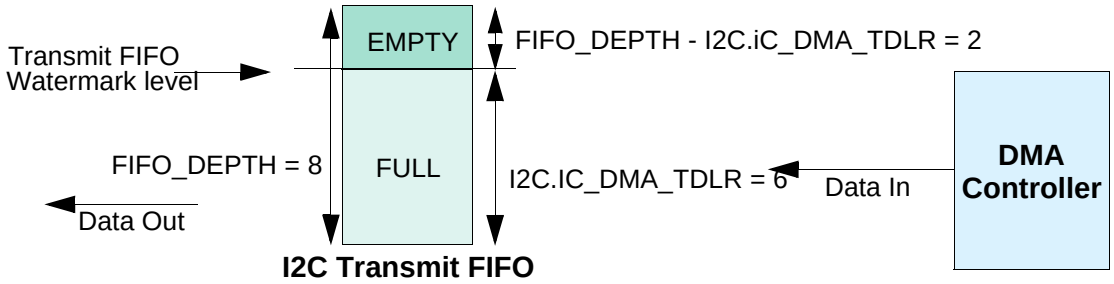
DMA.CTL*x*.BLOCK_TS / DMA.CTL*x*.DEST_MSIZE = 30 / 6 = 5

The number of burst transactions in the DMA block transfer is 5. But the watermark level, I2C.IC_DMA_TDLR, is quite low. Therefore, the probability of an I^2C underflow is high where the I^2C serial transmit line needs to transmit data, but where there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the transmit FIFO becomes empty.

2.17.4.2 Case 2: IC_DMA_TDLR = 6

- Transmit FIFO watermark level = I2C.IC_DMA_TDLR = 6
- DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - I2C.IC_DMA_TDLR = 2
- I2C transmit FIFO_DEPTH = 8
- DMA.CTLx.BLOCK_TS = 30

Figure 2-38 Case 2 Watermark Levels



Number of burst transactions in Block:

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

In this block transfer, there are 15 destination burst transactions in a DMA block transfer. But the watermark level, I2C.IC_DMA_TDLR, is high. Therefore, the probability of an I^2C underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the I^2C transmit FIFO becomes empty.

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

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

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

2.17.5 Selecting DEST_MSIZE and Transmit FIFO Overflow

As can be seen from Figure 2-38, programming DMA.CTL*x*.DEST_MSIZE to a value greater than the watermark level that triggers the DMA request may cause overflow when there is not enough space in the I^2C transmit FIFO to service the destination burst request. Therefore, the following equation must be adhered to in order to avoid overflow:

DMA.CTL*x*.DEST_MSIZE <= I2C.FIFO_DEPTH - I2C.IC_DMA_TDLR **(1)**

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

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

DMA.CTLx.DEST_MSIZE = I2C.FIFO_DEPTH - I2C.IC_DMA_TDLR **(2)**

This is the setting used in Figure 2-36.

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

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

2.17.6 Receive Watermark Level and Receive FIFO Overflow

During DW_apb_i2c 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, IC_DMA_RDLR+1. This is known as the watermark level. The DW_ahb_dmac responds by fetching a burst of data from the receive FIFO buffer of length CTLx.SRC_MSIZE.

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

2.17.7 Choosing the Receive Watermark level

Similar to choosing the transmit watermark level described earlier, the receive watermark level, IC_DMA_RDLR+1, should be set to minimize the probability of overflow, as shown in Figure 2-39. It is a trade-off between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

2.17.8 Selecting SRC_MSIZE and Receive FIFO Underflow

As can be seen in Figure 2-39, programming a source burst transaction length greater than the watermark level may cause underflow when there is not enough data to service the source burst request. Therefore, equation 3 following must be adhered to avoid underflow.

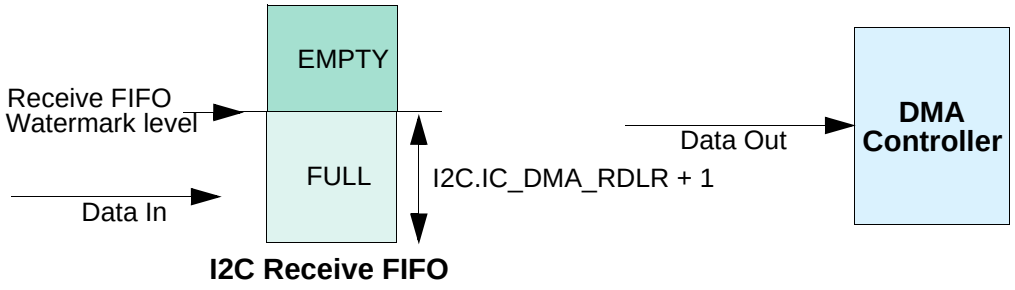
If the number of data items in the receive FIFO is equal to the source burst length at the time the burst request is made – DMA.CTL*x*.SRC_MSIZE – the receive FIFO may be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA.CTL*x*.SRC_MSIZE should be set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

DMA.CTL*x*.SRC_MSIZE = I2C.IC_DMA_RDLR + 1 **(3)**

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

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

Figure 2-39 I^2C Receive FIFO



2.17.9 Handshaking Interface Operation

The following sections discuss the handshaking interface.

2.17.9.1 dma_tx_req, dma_rx_req

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

The DW_ahb_dmac uses rising-edge detection of the dma_tx_req signal / dma_rx_req to identify a request on the channel. Upon reception of the dma_tx_ack / dma_rx_ack signal from the DW_ahb_dmac to indicate the burst transaction is complete, the DW_apb_i2c de-asserts the burst request signals, dma_tx_req / dma_rx_req, until dma_tx_ack / dma_rx_ack is de-asserted by the DW_ahb_dmac.

When the DW_apb_i2c 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 2-40 shows a timing diagram of a burst transaction where pclk = hclk.

Figure 2-40 Burst Transaction – pclk = hclk

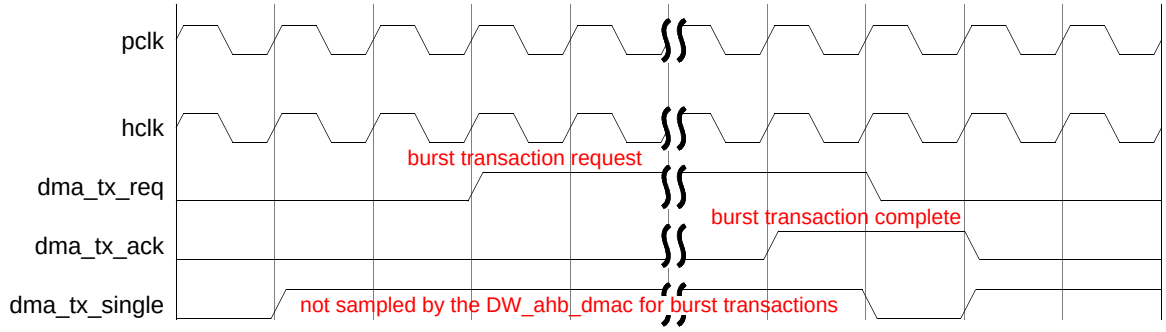
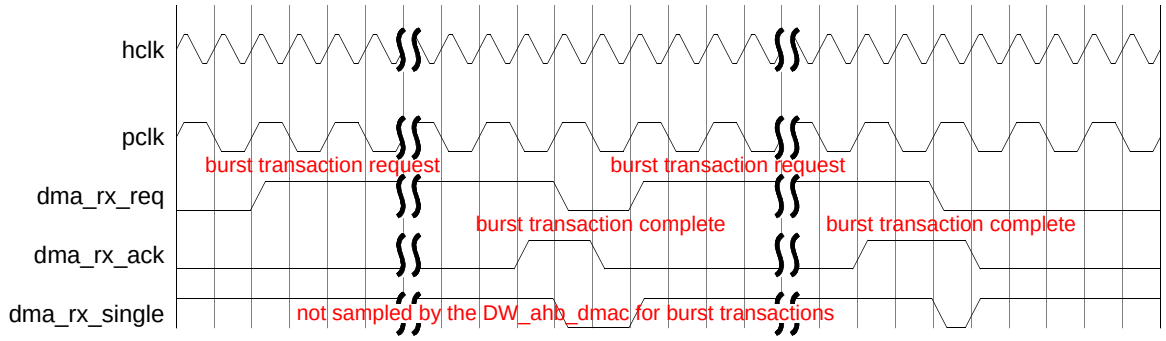


Figure 2-41 shows two back-to-back burst transactions where the hclk frequency is twice the pclk frequency.

Figure 2-41 Back-to-Back Burst Transactions – hclk = 2*pclk



The handshaking loop is as follows:

dma_tx_req / dma_rx_req asserted by DW_apb_i2c

-> dma_tx_ack / dma_rx_ack asserted by DW_ahb_dmac

-> dma_tx_req / dma_rx_req de-asserted by DW_apb_i2c

-> dma_tx_ack / dma_rx_ack de-asserted by DW_ahb_dmac

-> dma_tx_req / dma_rx_req reasserted by DW_apb_i2c, if back-to-back transaction is required

Note The burst transaction request signals, dma_tx_req and dma_rx_req, are generated in the DW_apb_i2c off pclk and sampled in the DW_ahb_dmac by hclk. The acknowledge signals, dma_tx_ack and dma_rx_ack, are generated in the DW_ahb_dmac off hclk and sampled in the DW_apb_i2c of pclk. The handshaking mechanism between the DW_ahb_dmac and the DW_apb_i2c supports quasi-synchronous clocks; that is, hclk and pclk must be phase-aligned, and the hclk frequency must be a multiple of the pclk frequency.

Two things to note here:

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

2.17.9.2 dma_tx_single, dma_rx_single

The dma_tx_single signal is a status signal. It is asserted when there is at least one free entry in the transmit FIFO and cleared when the transmit FIFO is full. The dma_rx_single signal is a status signal. It is asserted when there is at least one valid data entry in the receive FIFO and cleared when the receive FIFO is empty.

These signals are needed by only the DW_ahb_dmac for the case where the block size, CTL*x*.BLOCK_TS, that is programmed into the DW_ahb_dmac is not a multiple of the burst transaction length, CTL*x*.SRC_MSIZE, CTL*x*.DEST_MSIZE, as shown in Figure 2-36. In this case, the DMA single outputs inform the DW_ahb_dmac that it is still possible to perform single data item transfers, so it can access all data items in the transmit / receive FIFO and complete the DMA block transfer. The DMA single outputs from the DW_apb_i2c are not sampled by the DW_ahb_dmac otherwise. This is illustrated in the following example.

Consider first an example where the receive FIFO channel of the DW_apb_i2c is as follows:

DMA.CTLx.SRC_MSIZE = I2C.iC_DMA_RDLR + 1 = 4

DMA.CTLx.BLOCK_TS = 12

For the example in Figure 2-35, with the block size set to 12, the dma_rx_req signal is asserted when four data items are present in the receive FIFO. The dma_rx_req signal is asserted three times during the DW_apb_i2c serial transfer, ensuring that all 12 data items are read by the DW_ahb_dmac. All DMA requests read a block of data items and no single DMA transactions are required. This block transfer is made up of three burst transactions.

Now, for the following block transfer:

DMA.CTLx.SRC_MSIZE = I2C.IC_DMA_RDLR + 1 = 4

DMA.CTLx.BLOCK_TS = 15

The first 12 data items are transferred as already described using three burst transactions. But when the last three data frames enter the receive FIFO, the dma_rx_req 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 2-42 shows a single transaction. The handshaking loop is as follows:

dma_tx_single / dma_rx_single asserted by DW_apb_i2c

-> dma_tx_ack / dma_rx_ack asserted by DW_ahb_dmac

-> dma_tx_single / dma_rx_single de-asserted by DW_apb_i2c

-> dma_tx_ack / dma_rx_ack de-asserted by DW_ahb_dmac.

Figure 2-42 Single Transaction

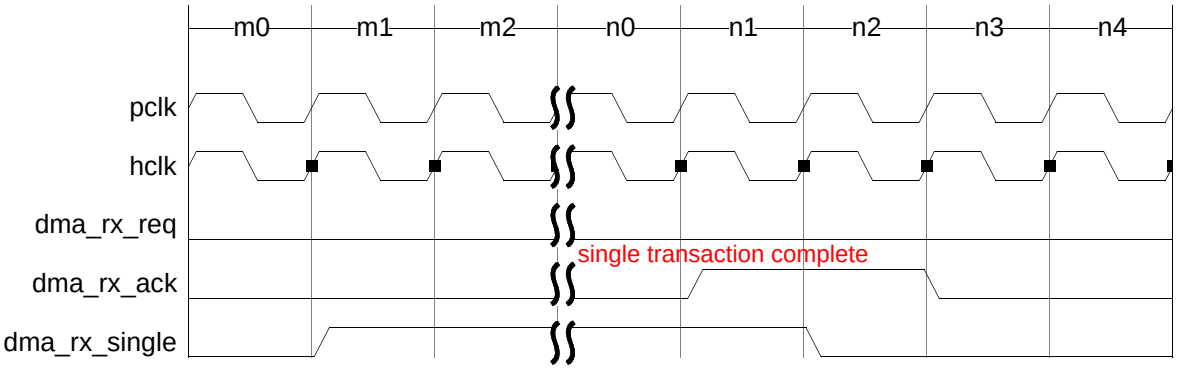
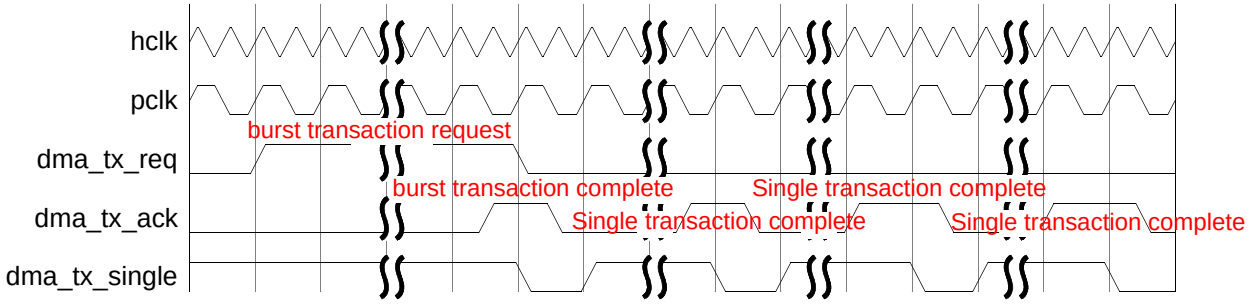


Figure 2-43 shows a burst transaction, followed by three back-to-back single transactions, where the hclk frequency is twice the pclk frequency.

Figure 2-43 Burst Transaction + 3 Back-to-Back Singles – hclk = 2*pclk



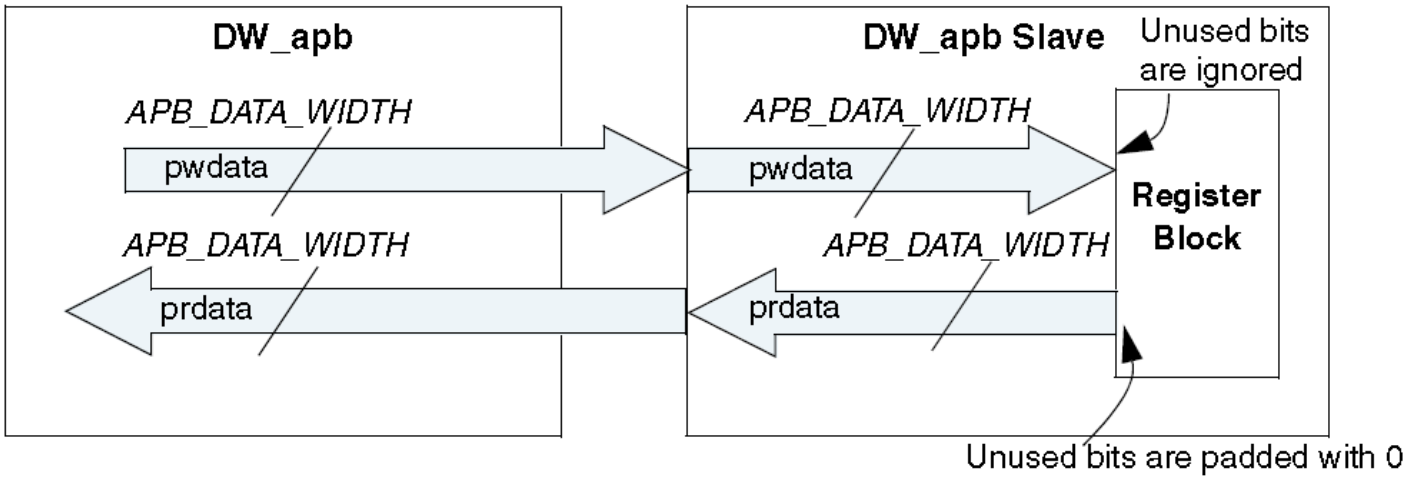
Note The single transaction request signals, dma_tx_single and dma_rx_single, are generated in the DW_apb_i2c on the pclk edge and sampled in DW_ahb_dmac on hclk. The acknowledge signals, dma_tx_ack and dma_rx_ack, are generated in the DW_ahb_dmac on the hclk edge hclk and sampled in the DW_apb_i2c on pclk. The handshaking mechanism between the DW_ahb_dmac and the DW_apb_i2c supports quasi-synchronous clocks; that is, hclk and pclk must be phase aligned and the hclk frequency must be a multiple of pclk frequency.

2.18 APB Interface

The host processor accesses data, control, and status information on the DW_apb_i2c peripheral through the AMBA APB 2.0 interface. This peripheral supports APB data bus widths of 8, 16, or 32 bits, which is set with the APB_DATA_WIDTH parameter.

Figure 2-44 shows the read / write buses between the DW_apb and the APB slave.

Figure 2-44 Read/Write Buses Between the DW_apb and an APB Slave



The register interface of DW_apb_i2c is compliant to APB 2.0, APB 3.0, and APB 4.0 specifications. The SLAVE_INTERFACE_TYPE parameter is used to select the APB interface type of the register interface.

2.18.1 APB 3.0 Support

The following signals are included as part of APB 3.0 interface.

The PREADY signal should always be kept to its default value that is high for all APB accesses except for the IC_DATA_CMD register access. During the IC_DATA_CMD access, the PREADY signal is de-asserted to stall the APB transaction under the following conditions:

- **Tx FIFO is empty:** The APB transaction completes and PREADY is asserted if the data is received in the Rx FIFO before the register read / write timeout happens.
- **Tx FIFO is full:** The APB transaction completes and PREADY is asserted if the data is read out of the Tx FIFO before the register read / write timeout happens.

If timeout happens, then the APB transaction is unsuccessful that is the RX FIFO is not read or the TX FIFO is not written. The APB transaction must be re-initiated by application for successful completion.

The PSLVERR signal functionality is enabled when the SLVERR_RESP_EN parameter is set to 1, so that DW_apb_i2c register interface can provide slave error response (if required). The DW_apb_i2c generates the error response under the following conditions:

- Enabling both master mode and slave mode in IC_CON register, which is an invalid programming.
- Writing high into CMD bit of the IC_DATA_CMD register, when IC_SLV_DISABLE bit of IC_CON register is low.

- The DW_apb_i2c stalls the APB transaction by pulling PREADY signal low, because the Rx FIFO is empty or the Tx FIFO is full. To avoid locking of the bus for the large number of clock cycles, a timeout option is provided through configuration parameter REG_TIMEOUT_VALUE. The timeout is triggered under the following conditions:
  - Rx FIFO remains empty or
  - TX FIFO remains full

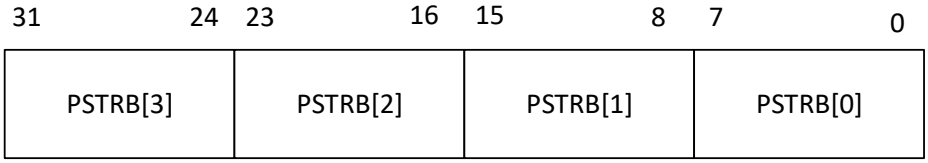
  If the duration is equal to the timeout period that is REG_TIMEOUT_VALUE, then APB interface asserts PSLVERR signal to indicate the register read / write timeout.

2.18.2 APB 4.0 Support

The DW_apb_i2c register interface is compliant to the APB 4.0 specification and to adhere to this compliance, the PSTRB and PPROT signals are added to the APB interface.

In the write transaction to this interface, the PSTRB signal indicates validity of PWDATA bytes. DW_apb_i2c component selectively writes to the bytes of the addressed register whose corresponding bit in the PSTRB signal is high. Bytes strobed low by the corresponding PSTRB bits are not modified. The incoming strobe bits for a read transaction is always zero as per the protocol.

Following figure shows the byte lane mapping of the PSTRB signal.



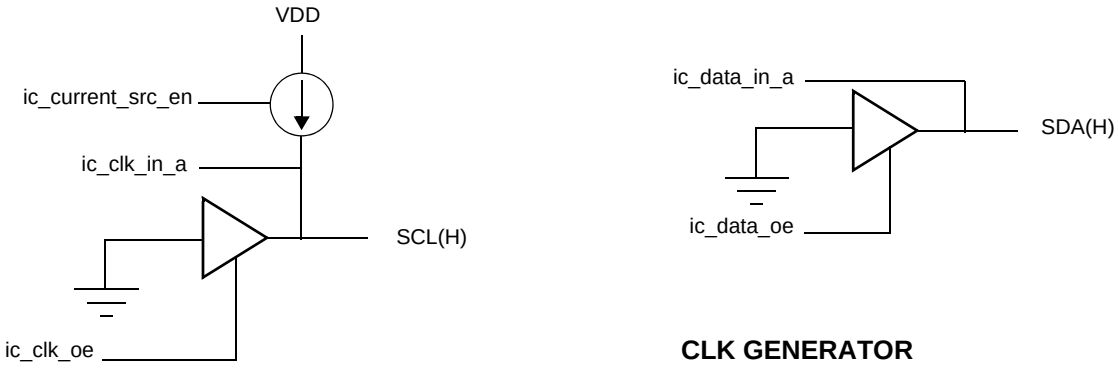
Note

- The PSTRB is not supported for write transaction on the IC_DATA_CMD register.
- The PPROT signal is added for interface consistency and PPROT signal is not used internally.

2.19 I/O Connections

As illustrated in Figure 2-45, the I2C interface consists of two wires, a clock (SCL) and data (SDA). For high speed systems, the names are SCLH and SDAH. For high speed mode, a current source pull-up may be used on the SCLH line. It is enabled during some active master transactions. The SDA and SDAH connections are the same at any speed. There are no special connections required for the DesignWare APB slave interface side of the DW_apb_i2c.

Figure 2-45 I/O Connection to I2C Interface



2.20 DW_apb_i2c Registers

The “Register Descriptions” on page 141 list all the registers available in DW_apb_i2c. Note that there are references to both hardware parameters and software registers throughout the chapter “Register Descriptions” on page 141. Parameters and many of the register bits are prefixed with an IC_*. However, the software register bits are distinguished in “Register Descriptions” on page 141 by italics. For instance, IC_MAX_SPEED_MODE is a hardware parameter and configured once using Synopsys coreConsultant, whereas the IC_SLAVE_DISABLE bit in the IC_CON register controls whether I2C has its slave disabled.

An address definition (memory map) C header file is shipped with the DW_apb_i2c component. You can use this header file when the DW_apb_i2c is programmed in a C environment.

Note A read operation to an address location that contains unused bits results in a 0 value being returned on each of the unused bits.

2.20.1 Registers and Field Descriptions

Registers in DW_apb_i2c are on the pclk domain, but status bits reflect actions that occur in the ic_clk domain. Therefore, there is delay when the pclk register reflects the activity that occurred on the ic_clk side.

Some registers may be written only when the DW_apb_i2c is disabled, programmed by the IC_ENABLE register. Software should not disable the DW_apb_i2c while it is active. If the DW_apb_i2c is in the process of transmitting when it is disabled, it stops as well as deletes the contents of the transmit buffer after the current transfer is complete. The slave continues receiving until the remote master aborts the transfer, in which case the DW_apb_i2c could be disabled. Registers that cannot be written to when the DW_apb_i2c is enabled are indicated in their descriptions.

Unless the clocks pclk and ic_clk are identical (IC_CLK_TYPE = 0), there is a two-register delay for synchronous and asynchronous modes.

2.20.2 Operation of Interrupt Registers

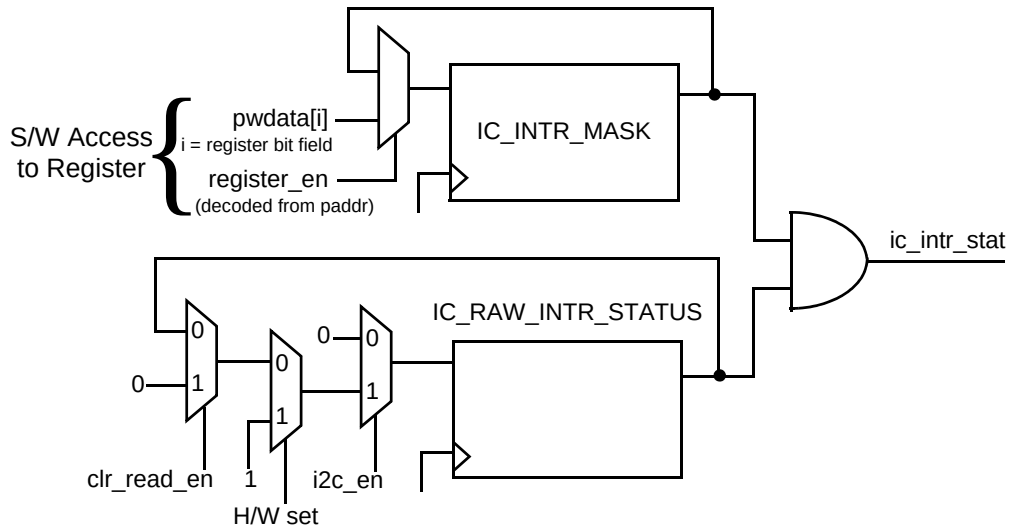
Table 2-10 lists the operation of the DW_apb_i2c interrupt registers and how they are set and cleared. Some bits are set by hardware and cleared by software, whereas other bits are set and cleared by hardware.

Table 2-10 Clearing and Setting of Interrupt Registers

Interrupt Bit Fields	Set by Hardware/ Cleared by Software	Set and Cleared by Hardware
MST_ON_HOLD	✘	✓
RESTART_DET	✓	✘
GEN_CALL	✓	✘
START_DET	✓	✘
STOP_DET	✓	✘
ACTIVITY	✓	✘
RX_DONE	✓	✘
TX_ABRT	✓	✘
RD_REQ	✓	✘
TX_EMPTY	✘	✓
TX_OVER	✓	✘
RX_FULL	✘	✓
RX_OVER	✓	✘
RX_UNDER	✓	✘

Figure 2-46 shows the operation of the interrupt registers where the bits are set by hardware and cleared by software.

Figure 2-46 Interrupt Scheme



2.21 UDID Feature

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device through the SMBus Host Controller. The SMBus Host controller makes use of SMBus Address Resolution Protocol to assign unique dynamic address for the slaves in the SMBus system. This SMBus Address Resolution protocol uses the unique 128-bit Unique Device ID (UDID), which helps to isolate each device in the SMBus system for address assignment. The DW_apb_i2c supports the 128-bit SMBus UDID through the configuration parameter IC_SMBUS_UDID_HC.

When IC_SMBUS_UDID_HC=1, the SMBus UDID is partially configurable and programmable that is Upper 96-bits of the UDID is configurable through the parameter IC_SMBUS_UDID_MSB and lower 32-bits are programmable through the register IC_SMBUS_UDID_LSB. The register default value is configurable through the parameter IC_SMBUS_UDID_LSB_DEFAULT.

When IC_SMBUS_UDID_HC=0, the SMBus UDID is completely software programmable through the registers IC_SMBUS_UDID_WORD0/1/2/3. This feature is useful when you want to update the SMBus UDID through the software. The register IC_SMBUS_UDID_WORD0 default value is configurable through the parameter IC_SMBUS_UDID_LSB_DEFAULT. The IC_SMBUS_UDID_WORD1/2/3 register, default value is derived based on the configurable parameter IC_SMBUS_UDID_MSB.

Software programmable UDID feature helps you to update the UDID quickly using the application software.

3 Parameter Descriptions

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

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

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

- Top Level Parameters on page 96
- I2C Version 3.0 Features on page 112
- SMBus Features on page 114
- I2C Version 6.0 Features on page 117

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
System Configuration	
Register Interface Type	Select Register Interface type as APB2, APB3 or APB4. By default, DW_apb_i2c supports APB2 interface. **Values:** ■ APB2 (0) ■ APB3 (1) ■ APB4 (2) **Default Value:** APB2 **Enabled:** DWC-APB-Advanced-Source source license exists. **Parameter Name:** SLAVE_INTERFACE_TYPE
Slave Error Response Enable	Enable Slave Error response signaling:The component will refrain From signaling an error response if this parameter is disabled. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** SLAVE_INTERFACE_TYPE>0 **Parameter Name:** SLVERR_RESP_EN
APB data bus width	Width of the APB data bus. **Values:** 8, 16, 32 **Default Value:** 8 **Enabled:** Always **Parameter Name:** APB_DATA_WIDTH
Width of Register timeout counter	Defines the width of Register timeout counter. If set to zero, the timeout counter register is disabled, and timeout is triggered as soon as the transaction tries to read an empty RX_FIFO or write to a full TX_FIFO. As these are the only cases where PREADY signal goes low , it ensures that PREADY is tied high throughout. Setting values from 4 through 32 for this parameter configures the timeout period from 2^4 to 2^8 pclk cycles. **Values:** 0, 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** SLAVE_INTERFACE_TYPE>0 && SLVERR_RESP_EN==1 **Parameter Name:** REG_TIMEOUT_WIDTH

Table 3-1 Top Level Parameters (Continued)

Label	Description
Hardcode Register timeout counter value	Checking this parameter makes Register timeout counter a read-only register. The register can be programmed by user if the hardcode option is turned off. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** SLAVE_INTERFACE_TYPE>0 && SLVERR_RESP_EN==1 && REG_TIMEOUT_WIDTH>0 **Parameter Name:** HC_REG_TIMEOUT_VALUE
Register Timeout counter default value	Defines the reset value of Register timeout counter register. This value can be over - ridden by programming the timeout counter register before enabling the component , if the HC_REG_TIMEOUT_VALUE is left un-checked **Values:** 1, ..., POW_2_REG_TIMEOUT_WIDTH **Default Value:** 8 **Enabled:** SLAVE_INTERFACE_TYPE>0 && SLVERR_RESP_EN==1 && REG_TIMEOUT_WIDTH>0 **Parameter Name:** REG_TIMEOUT_VALUE
Device Configuration	
Highest speed I2C mode supported	Maximum I2C mode supported. Controls the reset value of the SPEED bit field [2:1] of the I2C Control Register (IC_CON). Count registers are used to generate the outgoing clock SCL on the I2C interface. For speed modes faster than the configured maximum speed mode, the corresponding registers are not present in the top-level RTL. For unsupported speed modes those registers are not present as described below. ■ If this parameter is set to "Standard Mode" then the IC_FS_SCL_*, IC_HS_MADDR, and IC_HS_SCL_* registers are not present. ■ If this parameter is set to "Fast Mode" then the IC_HS_MADDR, and IC_HS_SCL_* registers are not present. **Values:** ■ Standard Mode (0x1) ■ Fast Mode or Fast Mode Plus (0x2) ■ High Speed Mode (0x3) **Default Value:** (IC_ULTRA_FAST_MODE ==1)? 1 : (IC_SMBUS == 1 ? 2 : 3) **Enabled:** IC_ULTRA_FAST_MODE == 0 **Parameter Name:** IC_MAX_SPEED_MODE

Table 3-1 Top Level Parameters (Continued)

Label	Description
Has I2C default slave address of?	Reset Value of DW_apb_i2c Slave Address. Controls the reset value of Register (IC_SAR). The default values cannot be any of the reserved address locations: 0x00 to 0x07 or 0x78 to 0x7f. **Values:** 0x000, ..., 0x3ff **Default Value:** 0x055 **Enabled:** Always **Parameter Name:** IC_DEFAULT_SLAVE_ADDR
Has I2C default target slave address of?	Reset value of DW_apb_i2c target slave address. Controls the reset value of the IC_TAR bit field (9:0) of the I2C Target Address Register (IC_TAR). The default values cannot be any of the reserved address locations: 0x00 to 0x07 or 0x78 to 0x7f. **Values:** 0x000, ..., 0x3ff **Default Value:** 0x055 **Enabled:** Always **Parameter Name:** IC_DEFAULT_TAR_SLAVE_ADDR
Has High Speed mode master code of?	High Speed mode master code of the DW_apb_i2c block. Controls the reset value of I2C HS Master Mode Code Address Register (IC_HS_MADDR). This is a unique code that alerts other masters on the I2C bus that a high-speed mode transfer is going to begin. For more information about this code, refer to "Multiple Master Arbitration" section in data book. **Values:** 0x0, ..., 0x7 **Default Value:** 0x1 **Enabled:** (IC_MAX_SPEED_MODE == 3) && (IC_ULTRA_FAST_MODE ==0) **Parameter Name:** IC_HS_MASTER_CODE
Is an I2C Master?	Controls whether DW_apb_i2c has its master enabled to be a master after reset. This parameter controls the reset value of bit 0 of the I2C Control Register (IC_CON). To enable the component to be a master, you must write a 1 in bit 0 of the IC_CON register. **Note:** If this parameter is checked (1), then you must ensure that the parameter IC_SLAVE_DISABLE is checked (1) as well. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** IC_MASTER_MODE

Table 3-1 Top Level Parameters (Continued)

Label	Description
Disable Slave after reset?	Controls whether DW_apb_i2c has its slave enabled or disabled after reset. If checked, the DW_apb_i2c slave interface is disabled after reset. The slave also can be disabled by programming a 1 into IC_CON[6]. By default the slave is enabled. **Note:** If this parameter is unchecked (0), then you must ensure that the parameter IC_MASTER_MODE is unchecked (0) as well. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** IC_SLAVE_DISABLE
Supports 10-bit addressing in slave mode?	Controls whether DW_apb_i2c slave supports 7 or 10 bit addressing on the I2C interface after reset when acting as a slave. Controls reset value of part of Register IC_CON. The DW_apb_i2c module will respond to this number of address bits when acting as a slave; it can be reprogrammed by software. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** IC_SMBUS == 1 ? 0 : 1 **Enabled:** Always **Parameter Name:** IC_10BITADDR_SLAVE
Supports 10-bit addressing in master mode?	Controls whether DW_apb_i2c supports 7 or 10 bit addressing on the I2C interface after reset when acting as a master. Controls reset value of part of Register IC_CON. Master generated transfers will use this number of address bits. Additionally, it can be reprogrammed by software by writing to the IC_CON register. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** IC_SMBUS == 1 ? 0 : 1 **Enabled:** Always **Parameter Name:** IC_10BITADDR_MASTER
Depth of transmit buffer is?	Depth of transmit buffer. The buffer is 9 bits wide; 8 bits for the data, and 1 bit for the read or write command. **Values:** 2, ..., 256 **Default Value:** 8 **Enabled:** Always **Parameter Name:** IC_TX_BUFFER_DEPTH

Table 3-1 Top Level Parameters (Continued)

Label	Description
Depth of receive buffer is?	Depth of receive buffer, the buffer is 8 bits wide. **Values:** 2, ..., 256 **Default Value:** 8 **Enabled:** Always **Parameter Name:** IC_RX_BUFFER_DEPTH
Transmit buffer threshold value is?	Reset value for threshold level of transmit buffer. This parameter controls the reset value of the I2C Transmit FIFO Threshold Level Register (IC_TX_TL). **Values:** 0x0, ..., IC_TX_BUFFER_DEPTH-1 **Default Value:** 0x0 **Enabled:** Always **Parameter Name:** IC_TX_TL
Receive buffer threshold value is?	Reset value for threshold level of receive buffer. This parameter controls the reset value of the I2C Receive FIFO Threshold Level Register (IC_RX_TL). **Values:** 0x0, ..., IC_RX_BUFFER_DEPTH-1 **Default Value:** 0x0 **Enabled:** Always **Parameter Name:** IC_RX_TL
Allow re-start conditions to be sent when acting as a master?	Controls the reset value of bit 5 (IC_RESTART_EN) in the IC_CON register. By default, this parameter is checked, which allows RESTART conditions to be sent when DW_apb_i2c is acting as a master. Some older slaves do not support handling RESTART conditions; however, RESTART conditions are used in several I2C operations. When the RESTART is disabled, the DW_apb_i2c master is incapable of performing the following functions: ■ Sending a START BYTE ■ Performing any high-speed mode operation ■ Performing direction changes in combined format mode ■ Performing a read operation with a 10-bit address **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** IC_RESTART_EN

Table 3-1 Top Level Parameters (Continued)

Label	Description
Hardware reset value for IC_SDA_SETUP register	Determines the reset value for the register IC_SDA_SETUP, which in turn controls the time delay - in terms of number of ic_clk clock periods - introduced in the rising edge of SCL, relative to SDA changing when a read-request is serviced. The relevant I2C requirement is t[su:DAT] as detailed in the I2C Bus Specifications. **Values:** 0x02, ..., 0xff **Default Value:** 0x64 **Enabled:** IC_ULTRA_FAST_MODE ==0 **Parameter Name:** IC_DEFAULT_SDA_SETUP
Hardware reset value for IC_SDA_HOLD register	Determines the reset value for the register IC_SDA_HOLD, which in turn controls the SDA hold time implemented by DW_apb_i2c (when transmitting or receiving, as either master or slave) as a master/slave transmitter or Master/Slave Reciever). The relevant I2C requirement is t[HD:DAT] as detailed in the I2C Bus Specifications. The programmed SDA hold time as transmitter cannot exceed at any time the duration of the low part of scl. Therefore it is recommended that the configured default value should not be larger than N_SCL_LOW-2, where N_SCL_LOW is the duration of the low part of the scl period measured in ic_clk cycles, for the maximum speed mode the component is configured for. **Values:** 0x000001, ..., 0xffffff **Default Value:** [<functionof> IC_USE_COUNTS IC_CLOCK_PERIOD IC_ULTRA_FAST_MODE] **Enabled:** Always **Parameter Name:** IC_DEFAULT_SDA_HOLD
IC_ACK_GENERAL_CALL set to acknowledge I2C general calls on reset	This parameter determines the reset value for the register IC_ACK_GENERAL_CALL, which in turn controls whether I2C general call addresses are to responded or not. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** IC_ULTRA_FAST_MODE == 0 **Parameter Name:** IC_DEFAULT_ACK_GENERAL_CALL
External Configuration	
Include DMA handshaking interface signals?	Configures the inclusion of DMA handshaking interface signals. When checked, includes the DMA handshaking interface signals at the top-level I/O. For more information about these signals, see "Signal Descriptions" in data book. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** IC_HAS_DMA

Table 3-1 Top Level Parameters (Continued)

Label	Description
Single Interrupt output port present?	If unchecked, each interrupt source has its own output. If checked, all interrupt sources are combined into a single output. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** IC_INTR_IO
Polarity of Interrupts is active high?	Configures the active level of the output interrupt lines. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** IC_INTR_POL
Internal Configuration	
Add Encoded Parameters	Adding the encoded parameters gives firmware an easy and quick way of identifying the DesignWare component within an I/O memory map. Some critical design-time options determine how a driver should interact with the peripheral. There is a minimal area overhead by including these parameters. Allows a single driver to be developed for each component which will be self-configurable. When bit 7 of the IC_COMP_PARAM_1 is read and contains a '1' the encoded parameters can be read via software. If this bit is a '0' then the entire register is '0' regardless of the setting of any of the other parameters that are encoded in the register's bits. For details about this register, see the IC_COMP_PARAM_1 register. **Note:** Unique drivers must be developed for each configuration of the DW_apb_i2c. Based on the configuration, the registers in the IP can differ; thus the same driver cannot be used with different configurations of the IP. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** IC_ADD_ENCODED_PARAMS

Table 3-1 Top Level Parameters (Continued)

Label	Description
Specify clock counts directly instead of supplying clock frequency?	Determines whether *CNT values are provided directly or by specifying the ic_clk clock frequency and letting coreConsultant (or coreAssembler) calculate the count values. When this parameter is checked, the reset values of the *CNT registers are specified by the corresponding *COUNT configuration parameters which may be user-defined or derived (see standard, fast, fast mode plus, and high speed mode parameters later in this table). When unchecked (default setting), the reset values of the *CNT registers are calculated from the configuration parameter IC_CLOCK_PERIOD. **Note:** For fast mode plus, reprogram the IC_FS_SCL_*CNT register to achieve the required data rate when unchecked. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** IC_USE_COUNTS
Hard code the count values for each mode?	By checking this parameter, the *CNT registers are set to read only. Unchecking this parameter (default setting) allows the *CNT registers to be writable. Regardless of the setting, the *CNT registers are always readable and have reset values from the corresponding *COUNT configuration parameters, which may be user defined or derived (see standard, fast, fast mode plus, or high speed mode parameters later in this table). **Note:** Since the DW_apb_i2c uses the same high and low count registers for fast mode and fast mode plus operation, if this parameter is checked (1) the IC_FS_SCL_*CNT registers are hard coded to either one of the fast mode and fast mode plus. Consequently, DW_apb_i2c can operate in either fast mode or fast mode plus, but not in both modes simultaneously. For fast mode plus, it is recommended that this parameter be Unchecked (0). **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** IC_HC_COUNT_VALUES

Table 3-1 Top Level Parameters (Continued)

Label	Description
ic_clk has a period of? (ns integers only)	Specifies the period of incoming ic_clk, used to generate outgoing I2C interface SCL clock. (ns integers only) When the count values are used to generate the IC_CLOCK_PERIOD then the IC_MAX_SPEED_MODE setting determines the actual period IC_MAX_SPEED_MODE = Standard => 500ns IC_MAX_SPEED_MODE = Fast => 100ns IC_MAX_SPEED_MODE = High => 10ns IC_ULTRA_FAST_MODE = 1 => 25ns **Note:** For fast mode plus, user has to reprogram the IC_FS_SCL_*CNT register to achieve required data rate. **Values:** 2, ..., 2147483647 **Default Value:** [<functionof> IC_MAX_SPEED_MODE IC_ULTRA_FAST_MODE] **Enabled:** IC_USE_COUNTS == 0 **Parameter Name:** IC_CLOCK_PERIOD
Relationship between pclk and ic_clk is?	Specifies the relationship between pclk and ic_clk Identical (0): clocks are identical; no meta-stability flops used for data passing between clock domains. Asynchronous (1): clocks may be completely asynchronous to each other, meta-stability flops are required for data passing between clock domains. **Values:** ■ Identical (0x0) ■ Asynchronous (0x1) **Default Value:** 0x1 **Enabled:** Always **Parameter Name:** IC_CLK_TYPE
Enable Async FIFO Mode?	This parameter controls whether DW_apb_i2c consist of Asynchronous or Synchronous FIFO's for the Transmit and Receive Data Buffers. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_CLK_TYPE==ASYNC **Parameter Name:** IC_HAS_ASYNC_FIFO

Table 3-1 Top Level Parameters (Continued)

Label	Description
Standard Speed Mode Configuration	
Std speed SCL high count is?	Reset value of Standard Speed I2C Clock SCL High Count register (IC_SS_SCL_HCNT). The value must be calculated based on the I2C data rate desired and I2C clock frequency. When parameter IC_USE_COUNTS = 0, this parameter is automatically calculated using the IC_CLOCK_PERIOD parameter. For more information, see the IC_SS_SCL_HCNT register. **Values:** IC_HCNT_LO_LIMIT, ..., 0xffff **Default Value:** [<functionof> IC_USE_COUNTS IC_HCNT_LO_LIMIT IC_CLOCK_PERIOD] **Enabled:** (IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE ==0) **Parameter Name:** IC_SS_SCL_HIGH_COUNT
Std speed SCL low count is?	Reset value of Standard Speed I2C Clock SCL High Count register (IC_SS_SCL_HCNT). Value must be calculated based on I2C data rate desired and I2C clock frequency. When parameter IC_USE_COUNTS = 0, this parameter is automatically calculated using the IC_CLOCK_PERIOD parameter. For more information, see IC_SS_SCL_LCNT register. **Values:** IC_LCNT_LO_LIMIT, ..., 0xffff **Default Value:** [<functionof> IC_USE_COUNTS IC_LCNT_LO_LIMIT IC_CLOCK_PERIOD] **Enabled:** (IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE ==0) **Parameter Name:** IC_SS_SCL_LOW_COUNT
Fast Mode or Fast Mode Plus Configuration	
Fast speed SCL high count is?	Reset value of Fast Mode or Fast Mode Plus I2C Clock SCL High Count register (IC_FS_SCL_HCNT). The value must be calculated based on I2C data rate desired and I2C clock frequency. When parameter IC_USE_COUNTS = 0, this parameter is automatically calculated using the IC_CLOCK_PERIOD parameter. For more information, see IC_FS_SCL_HCNT register. **Values:** IC_HCNT_LO_LIMIT, ..., 0xffff **Default Value:** [<functionof> IC_MAX_SPEED_MODE IC_USE_COUNTS IC_HCNT_LO_LIMIT IC_CLOCK_PERIOD] **Enabled:** (IC_MAX_SPEED_MODE>=2 && IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE==0) **Parameter Name:** IC_FS_SCL_HIGH_COUNT

Table 3-1 Top Level Parameters (Continued)

Label	Description
Fast speed SCL low count is?	Reset value of Fast Mode or Fast Mode Plus I2C Clock SCL Low Count register (IC_FS_SCL_LCNT). The value must be calculated based on I2C data rate desired and I2C clock frequency. When parameter IC_USE_COUNTS = 0, this parameter is automatically calculated using the IC_CLOCK_PERIOD parameter. For more information, see the IC_FS_SCL_LCNT register **Values:** IC_LCNT_LO_LIMIT, ..., 0xffff **Default Value:** [<functionof> IC_MAX_SPEED_MODE IC_USE_COUNTS IC_LCNT_LO_LIMIT IC_CLOCK_PERIOD] **Enabled:** (IC_MAX_SPEED_MODE>=2 && IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE==0) **Parameter Name:** IC_FS_SCL_LOW_COUNT
High Speed Mode Configuration	
For high speed mode systems the I2C bus loading is? (pF)	For high speed mode, the bus loading affects the high and low pulse width of SCL. **Values:** ■ 100 (100) ■ 400 (400) **Default Value:** 100 **Enabled:** (IC_MAX_SPEED_MODE==3) && (IC_ULTRA_FAST_MODE ==0) **Parameter Name:** IC_CAP_LOADING
High speed SCL high count is?	Reset value of High Speed I2C Clock SCL High Count register (IC_HS_SCL_HCNT). The value must be calculated based on I2C data rate desired and high speed I2C clock frequency. When parameter IC_USE_COUNTS = 0, this parameter is automatically calculated using the IC_CLOCK_PERIOD parameter. For more information, see IC_HS_SCL_HCNT register. **Values:** IC_HCNT_LO_LIMIT, ..., 0xffff **Default Value:** [<functionof> IC_MAX_SPEED_MODE IC_USE_COUNTS IC_HCNT_LO_LIMIT IC_CLOCK_PERIOD IC_CAP_LOADING] **Enabled:** (IC_MAX_SPEED_MODE==3 && IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE==0) **Parameter Name:** IC_HS_SCL_HIGH_COUNT
High speed SCL low count is?	Reset value of High Speed I2C Clock SCL Low Count register (IC_HS_SCL_LCNT). The value must be calculated based on I2C data rate and I2C clock frequency. When parameter IC_USE_COUNTS = 0, this parameter is automatically calculated using the IC_CLOCK_PERIOD parameter. For more information, see IC_HS_SCL_LCNT register. **Values:** IC_LCNT_LO_LIMIT, ..., 0xffff **Default Value:** [<functionof> IC_MAX_SPEED_MODE IC_USE_COUNTS IC_LCNT_LO_LIMIT IC_CLOCK_PERIOD IC_CAP_LOADING] **Enabled:** (IC_MAX_SPEED_MODE==3 && IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE==0) **Parameter Name:** IC_HS_SCL_LOW_COUNT

Table 3-1 Top Level Parameters (Continued)

Label	Description
Spike Suppression Configuration	
Maximum length (in ic_clk cycles) of suppressed spikes in Standard Mode, Fast Mode, and Fast Mode Plus	Reset value of maximum suppressed spike length register in Standard Mode, Fast Mode, and Fast Mode Plus modes (IC_FS_SPKLEN Register). Spike length is expressed in ic_clk cycles and this value is calculated based on the value of IC_CLOCK_PERIOD. **Values:** 0x1, ..., 0xff **Default Value:** [<functionof> IC_CLOCK_PERIOD IC_FS_MAX_SPKLEN] **Enabled:** IC_ULTRA_FAST_MODE==0 **Parameter Name:** IC_DEFAULT_FS_SPKLEN
Maximum length (in ic_clk cycles) of suppressed spikes in HS mode	Reset value of maximum suppressed spike length register in HS modes (Register IC_HS_SPKLEN). Spike length is expressed in ic_clk cycles and this value is calculated based on the value of IC_CLOCK_PERIOD. **Values:** 0x1, ..., 0xff **Default Value:** [<functionof> IC_CLOCK_PERIOD IC_HS_MAX_SPKLEN] **Enabled:** (IC_MAX_SPEED_MODE==3) && (IC_ULTRA_FAST_MODE ==0) **Parameter Name:** IC_DEFAULT_HS_SPKLEN
Additional Features	
Allow dynamic updating of the TAR address?	When checked, allows the IC_TAR register to be updated dynamically. Setting this parameter affects the operation of DW_apb_i2c when it is in master mode. For more details, see "Master Mode Operation". **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** I2C_DYNAMIC_TAR_UPDATE

Table 3-1 Top Level Parameters (Continued)

Label	Description
Enable register to generate NACKs for data received by Slave?	Enables an additional register which controls whether the DW_apb_i2c generates a NACK after a data byte has been transferred to it. This NACK generation only occurs when the DW_apb_i2c is a Slave-Receiver. If this register is set to a value of 1, it can only generate a NACK after a data byte is received; hence, the data transfer is aborted. Also, the data received is not pushed to the receive buffer. When the register is set to a value of 0, it generates NACK/ACK depending on normal criteria. If this option is selected, the default value of the register IC_SLV_DATA_NACK_ONLY is always 0. The register must be explicitly programmed to a value of 1 if NACKs are to be generated. The register can only be written to successfully if DW_apb_i2c is disabled (IC_ENABLE[0] = 0) or the slave part is inactive (IC_STATUS[6] = 0). **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_ULTRA_FAST_MODE ==0 **Parameter Name:** IC_SLV_DATA_NACK_ONLY
When Receive Fifo is Physically full, Generate NACK for data received by slave?	This parameter enables DW_apb_i2c in Slave mode to generate NACK for a data byte recieved when Receive FIFO is physically full. The new data byte will not be pushed to the Receive FIFO, hence no overflow happens and rx_over interrupt will not be set. This works only when DW_apb_i2c is in Slave/Receiver mode (data being written to the slave) and is not applicable in Master mode. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** (IC_ULTRA_FAST_MODE ==0) && (IC_SLV_DATA_NACK_ONLY ==0) **Parameter Name:** IC_RX_FULL_GEN_NACK
Hold transfer when Tx FIFO is empty	If this parameter is set, the master will only complete a transfer - that is issues a STOP - when it finds a Tx FIFO entry tagged with a Stop bit. If the Tx FIFO becomes empty and the last byte does not have the Stop bit set, the master stalls the transfer by holding the SCL line low. If this parameter is not set, the master completes a transfer when the Tx FIFO is empty. In SMbus Mode (IC_SMBUS=1), IC_EMPTYFIFO_HOLD_MASTER_EN should be always enabled. **Values:** ■ false (0) ■ true (1) **Default Value:** IC_SMBUS == 1 ? 1 : 0 **Enabled:** Always **Parameter Name:** IC_EMPTYFIFO_HOLD_MASTER_EN

Table 3-1 Top Level Parameters (Continued)

Label	Description
When Receive Fifo is physically full, Hold the bus till Receive fifo has space avialable?	When the Rx FIFO is physically full to its RX_BUFFER_DEPTH, this parameter provides a hardware method to hold the bus till Rx FIFO data is read out and there is a space available in the FIFO. This parameter can be used when DW_apb_i2c is either a slave-receiver (that is, data is written to the device) or a master-receiver (that is, the device reads data from a slave). **Note:** If parameter "IC_RX_FULL_GEN_NACK" is enabled, then setting this parameter has no impact in slave-receiver mode since, the controller NACK's the Data byte if Rx-FIFO has no empty space.**Note:** If this parameter is checked, then the RX_OVER interrupt is never set to 1 as the criteria to set this interrupt is never met. The RX_OVER interrupt can be found in IC_INTR_STAT and IC_RAW_INTR_STAT registers. It is also an optional output signal, ic_rx_over_intr(_n). **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_ULTRA_FAST_MODE ==0 **Parameter Name:** IC_RX_FULL_HLD_BUS_EN
Enable restart detect interrupt in slave mode?	When checked, allows the slave to detect and issue the restart interrupt when slave is addressed. Setting this parameter affects the operation of DW_apb_i2c only when it is in slave mode. This controls the "RESTART_DET" bit in the IC_RAW_INTR_STAT, IC_INTR_MASK, IC_INTR_STAT, and IC_CLR_RESTART_DET registers.This also controls the ic_restart_det_intr(_n) and ic_intr(_n) signals. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** IC_SLV_RESTART_DET_EN

Table 3-1 Top Level Parameters (Continued)

Label	Description
Generate STOP_DET interrupt only if Master is active?	Controls whether DW_apb_i2c generates STOP_DET interrupt when master is active: ■ Checked (1): Allows the master to detect and issue the stop interrupt when master is active. ■ Unchecked (0): The master always detects and issues the stop interrupt irrespective of whether it is active. This parameter affects the operation of DW_apb_i2c when it is in master mode. This controls the STOP_DET bit of the IC_RAW_INTR_STAT, IC_INTR_MASK, IC_INTR_STAT and IC_CLR_STOP_DET registers. This also controls the ic_stop_det_intr(_n) and ic_intr(_n) signals. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_ULTRA_FAST_MODE ==0 **Parameter Name:** IC_STOP_DET_IF_MASTER_ACTIVE
Include Status bits to indicate the reason for clock stretching?	If this parameter is set, the DW_apb_i2c consists of status bits indicating the reason for clock stretching in the IC_STATUS Register. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_ULTRA_FAST_MODE ==0 **Parameter Name:** IC_STAT_FOR_CLK_STRETCH
Include programmable bit for blocking Master commands?	Controls whether DW_apb_i2c transmits data on I2C bus as soon as data is available in Tx FIFO. When checked, allows the master to hold the transmission of data on I2C bus when Tx FIFO has data to transmit. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** IC_TX_CMD_BLOCK

Table 3-1 Top Level Parameters (Continued)

Label	Description
Enable blocking Master commands after reset?	Controls whether DW_apb_i2c has its transmit command block enabled or disabled after reset. If checked, the DW_apb_i2c blocks the transmission of data on I2C bus. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_TX_CMD_BLOCK==1 **Parameter Name:** IC_TX_CMD_BLOCK_DEFAULT
Include First data byte indication in IC_DATA_CMD register?	Controls whether DW_apb_i2c generates FIRST_DATA_BYTE status bit in IC_DATA_CMD register. When checked, the master/slave receiver to set the FIRST_DATA_BYTE status bit in IC_DATA_CMD register to indicate whether the data present in IC_DATA_CMD register is first data byte after the address phase of a receive transfer. **Note**: In the case when APB_DATA_WIDTH is set to 8, you must perform two APB reads to the IC_DATA_CMD register to get status on bit 11. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** IC_FIRST_DATA_BYTE_STATUS
Avoid Rx FIFO Flush on Tranmsit Abort?	This Parameter controls the Rx FIFO Flush during the Transmit Abort. If this parameter is checked(1), only the Tx FIFO is flushed (not the Rx FIFO) Flush on the Transmit Abort. If this parameter is unchecked(0), both Tx FIFO and Rx FIFO are flushed on Transmit Abort. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_ULTRA_FAST_MODE ==0 **Parameter Name:** IC_AVOID_RX_FIFO_FLUSH_ON_TX_ABRT
Enable IC_CLK Frequency Reduction?	This parameter is used to reduce the system clock frequency (ic_clk) by reducing the internal latency required to generate the high period and low period of the SCL line. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** IC_ULTRA_FAST_MODE == 1 ? 1 : 0 **Enabled:** DWC-APB-Advanced-Source License Required and IC_ULTRA_FAST_MODE=0 **Parameter Name:** IC_CLK_FREQ_OPTIMIZATION

3.2 I2C Version 3.0 Features Parameters

Table 3-2 I2C Version 3.0 Features Parameters

Label	Description
I2C 3.0 Features	
Include Bus Clear feature?	This parameter will enable the Bus clear feature for the DW_apb_i2c core. If this parameter is set: ■ If an SDA line is stuck at low for IC_SDA_STUCK_LOW_TIMEOUT period of ic_clk, DW_apb_i2c master generates a master transmit abort (IC_TX_ABRT_SOURCE[17]: ABRT_SDA_STUCK_AT_LOW) to indicate SDA stuck at low. User can enable the SDA_STUCK_RECOVERY_EN (IC_ENABLE[3]) register bit to recover the SDA by sending at most 9 SCL clocks. If SDA line is recovered, then the master generates a STOP and auto clear the 'SDA_STUCK_RECOVERY_EN' register bit and resume the normal I2C transfers. If an SDA line is not recovered, then the master auto clears the SDA_STUCK_RECOVERY_EN register bit and asserts the SDA_STUCK_NOT_RECOVERED (IC_STATUS[12]) status bit to indicate the SDA is not recovered after sending 9 SCL clocks which intimate the user for system reset. ■ If SCL line is stuck at low for IC_SCL_STUCK_LOW_TIMEOUT period of ic_clk, DW_apb_i2c Master will generate an SCL_STUCK_AT_LOW (IC_INTR_RAW_STATUS[14]) interrupt to intimate the user for system reset. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** IC_SMBUS==1 ? 1 : 0 **Enabled:** IC_ULTRA_FAST_MODE ==0 **Parameter Name:** IC_BUS_CLEAR_FEATURE
Has SCL Stuck Timeout value of ?	Default value of the IC_SCL_STUCK_LOW_TIMEOUT Register. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0xffffffff **Enabled:** IC_BUS_CLEAR_FEATURE==1 **Parameter Name:** IC_SCL_STUCK_TIMEOUT_DEFAULT
Has SDA Stuck Timeout value of ?	Default value of the IC_SDA_STUCK_LOW_TIMEOUT Register. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0xffffffff **Enabled:** IC_BUS_CLEAR_FEATURE==1 **Parameter Name:** IC_SDA_STUCK_TIMEOUT_DEFAULT

Table 3-2 I2C Version 3.0 Features Parameters (Continued)

Label	Description
Enable DEVICE-ID feature?	If this Parameter is enabled, the DW_apb_i2c slave includes a 24-bit IC_DEVICE_ID Register to store the value of Device-ID and transmits whenever master is requested. The Master mode includes a DEVICE_ID bit 13 in IC_TAR register to initiate the Device ID read for a particular slave address mentioned in IC_TAR[6:0] register. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_ULTRA_FAST_MODE ==0 **Parameter Name:** IC_DEVICE_ID
Has I2C Slave DEVICE ID value of?	Device ID Value of the I2C Slave stored in the IC_DEVICE_ID Register (24 bit, MSB is transferred first on the Device ID read from the master). **Values:** 0x0, ..., 0xffffff **Default Value:** 0x0 **Enabled:** IC_DEVICE_ID==1 **Parameter Name:** IC_DEVICE_ID_VALUE

3.3 SMBus Features Parameters

Table 3-3 SMBus Features Parameters

Label	Description
I2C System Management Bus Features	
Enable SMBus Mode?	Controls whether DW_apb_i2c Master/Slave supports SMBus mode. If checked, the DW_apb_i2c includes the SMBus mode related registers, real-time checks, timeout interrupts, and SMBus optional signals. **Note:** If this parameter is selected (1), then the user can set the parameter IC_MAX_SPEED_MODE to Standard mode(1) or Fast Mode/Fast Mode Plus (2). **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** DWC-APB-Advanced-Source License Required and IC_ULTRA_FAST_MODE=0 **Parameter Name:** IC_SMBUS
Has SMBus clock low Slave extend default Timeout value of ?	Default value of the IC_SMBUS_CLK_LOW_SEXT Register. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0xffffffff **Enabled:** IC_SMBUS==1 **Parameter Name:** IC_SMBUS_CLK_LOW_SEXT_DEFAULT
Has SMBus clock low Master extend default Timeout value of ?	Default value of the IC_SMBUS_CLK_LOW_MEXT Register. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0xffffffff **Enabled:** IC_SMBUS==1 **Parameter Name:** IC_SMBUS_CLK_LOW_MEXT_DEFAULT
Has SMBus Thigh:Max Idle count Value of ?	Default value of the IC_SMBUS_THIGH_MAX_IDLE_COUNT Register. **Values:** 0x0, ..., 0xffff **Default Value:** 0xffff **Enabled:** IC_SMBUS==1 **Parameter Name:** IC_SMBUS_RST_IDLE_CNT_DEFAULT
Enable SMBus Optional Signals?	This parameter controls whether DW_apb_i2c includes Optional SMBus Suspend and Alert signals on the interface. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_SMBUS==1 **Parameter Name:** IC_SMBUS_SUSPEND_ALERT

Table 3-3 SMBus Features Parameters (Continued)

Label	Description
Include Optional slave address register?	This parameter controls whether to include optional Slave Address Register in SMBus Mode. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** IC_SMBUS==1 **Parameter Name:** IC_OPTIONAL_SAR
Has I2C default optional slave address of?	Controls whether to include Optional Slave Address Register in SMBus Mode. A user is not allowed to assign any reserved addresses. The reserved address are as follows: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f **Values:** 0x0, ..., 0x7f **Default Value:** 0x0 **Enabled:** IC_OPTIONAL_SAR==1 **Parameter Name:** IC_OPTIONAL_SAR_DEFAULT
Enable Address Resolution Protocol in SMBus Mode?	Controls whether DW_apb_i2c includes logic to detect and respond ARP commands in Slave mode. It also includes logic to generate/validate the PEC byte at the end of the transfer in Slave mode only. **Values:** 0x0, 0x1 **Default Value:** 0x0 **Enabled:** IC_SMBUS==1 **Parameter Name:** IC_SMBUS_ARP
SMBus Unique Device Identifier (UDID) Hardcode?	Controls whether Unique Device Identifier (UDID) used for Dynamic Address Resolution process in SMBus ARP Mode is Hardcoded (Upper 96-bits) or Complete UDID is Software Programmable. **Values:** 0x0, 0x1 **Default Value:** 0x1 **Enabled:** IC_SMBUS_ARP==1 **Parameter Name:** IC_SMBUS_UDID_HC
Has SMBUS Unique device identifier (MSB 96 bits) value of?	If the parameter IC_SMBUS_UDID_HC is 1, stores the Static Unique Device Identifier used for Dynamic Address Resolution process in SMBus ARP Mode (Upper 96bits of UDID). If the parameter IC_SMBUS_UDID_HC is 0, then this field is used as the default value of the upper 96bits of the UDID Registers {IC_SMBUS_UDID_WORD3, IC_SMBUS_UDID_WORD2, IC_SMBUS_UDID_WORD1} **Values:** 0x0, ..., 0xffffffffffffffffffffffff **Default Value:** 0x0 **Enabled:** IC_SMBUS_ARP==1 **Parameter Name:** IC_SMBUS_UDID_MSB

Table 3-3 SMBus Features Parameters (Continued)

Label	Description
Has Default SMBus Unique device identifier (LSB 32 bits) value of?	If the parameter IC_SMBUS_UDID_HC is 1, specifies default value of the IC_SMBUS_UDID_LSB register used for Dynamic Address Resolution process in SMBus ARP mode (Lower 32bits of UDID). If the parameter IC_SMBUS_UDID_HC is 0, specifies default value of the IC_SMBUS_UDID_WORD0 register used for Dynamic Address Resolution process in SMBus ARP mode (Lower 32bits of UDID). **Values:** 0x0, ..., 0xffffffff **Default Value:** 0xffffffff **Enabled:** IC_SMBUS_ARP==1 **Parameter Name:** IC_SMBUS_UDID_LSB_DEFAULT
Has Default Persistent Slave Address register bit Value of ?	Default value of the Persistent Slave Address register bit in IC_CON Register. **Values:** 0x0, 0x1 **Default Value:** 0x0 **Enabled:** IC_SMBUS_ARP==1 **Parameter Name:** IC_PERSISTANT_SLV_ADDR_DEFAULT

3.4 I2C Version 6.0 Features Parameters

Table 3-4 I2C Version 6.0 Features Parameters

Label	Description
I2C 6.0 Features	
Enable Ultra-Fast Mode?	This parameter is used to control whether DW_apb_i2c supports Ultra-Fast speed mode or not. If this Parameter is enabled, the Master ■ Disables the Arbitration, clock synchronization features. ■ Support only write transfers. ■ Does not check the validity of ACK/NACK for each byte. The Slave ■ Supports only write transfers. ■ Disables the logic to generate ACK/NACK after the end of each byte. ■ Disables the logic to stretch the clock if RX-FIFO is full. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** DWC-APB-Advanced-Source License Required **Parameter Name:** IC_ULTRA_FAST_MODE
Ultra Fast speed SCL high count is?	Reset value of Ultra-Fast Speed I2C Clock SCL High Count register (IC_UFM_SCL_HCNT). The value must be calculated based on the I2C data rate desired and I2C clock frequency. When parameter IC_USE_COUNTS = 0, this parameter is automatically calculated using the IC_CLOCK_PERIOD parameter. **Values:** IC_HCNT_LO_LIMIT, ..., 0xffff **Default Value:** [<functionof> IC_USE_COUNTS IC_HCNT_LO_LIMIT IC_CLOCK_PERIOD IC_ULTRA_FAST_MODE] **Enabled:** (IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE==1) **Parameter Name:** IC_UFM_SCL_HIGH_COUNT
Ultra Fast speed SCL low count is?	Reset value of Ultra-Fast Speed I2C Clock SCL Low Count register (IC_UFM_SCL_LCNT). The value must be calculated based on the I2C data rate desired and I2C clock frequency. When parameter IC_USE_COUNTS = 0, this parameter is automatically calculated using the IC_CLOCK_PERIOD parameter. **Values:** IC_LCNT_LO_LIMIT, ..., 0xffff **Default Value:** [<functionof> IC_USE_COUNTS IC_LCNT_LO_LIMIT IC_CLOCK_PERIOD IC_ULTRA_FAST_MODE] **Enabled:** (IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE==1) **Parameter Name:** IC_UFM_SCL_LOW_COUNT

Table 3-4 I2C Version 6.0 Features Parameters (Continued)

Label	Description
Maximum length (in ic_clk cycles) of suppressed spikes in Ultra Fast mode	Reset value of maximum suppressed spike length register in Ultra-Fast Mode (IC_UFM_SPKLEN Register). Spike length is expressed in ic_clk cycles and this value is calculated based on the value of IC_CLOCK_PERIOD. **Values:** 0x1, ..., 0xff **Default Value:** [<functionof> IC_CLOCK_PERIOD IC_HS_MAX_SPKLEN] **Enabled:** IC_ULTRA_FAST_MODE ==1 **Parameter Name:** IC_DEFAULT_UFM_SPKLEN
Has Ultra Fast mode tBuf count Value of ?	Default value of the IC_UFM_TBUF_CNT Register. This parameter is active when the IC_USE_COUNTS and IC_ULTRA_FAST_MODE parameters are checked (1); otherwise, this value is automatically calculated using the IC_CLK_PERIOD parameter. **Values:** 0x0, ..., 0xffff **Default Value:** [<functionof> IC_USE_COUNTS IC_CLOCK_PERIOD] **Enabled:** (IC_USE_COUNTS==1) && (IC_ULTRA_FAST_MODE==1) **Parameter Name:** IC_UFM_TBUF_CNT_DEFAULT

4
Signal Descriptions

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

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

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

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

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

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

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

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

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

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

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

Attributes used with Synchronous To

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

The I/O signals are grouped as follows:

- Interrupts on page 121
- I2C Interface (Master/Slave) on page 129
- APB Slave Interface on page 132
- DMA Interface on page 135
- SMBus Interface on page 137
- I2C Debug on page 138

4.1 Interrupts Signals

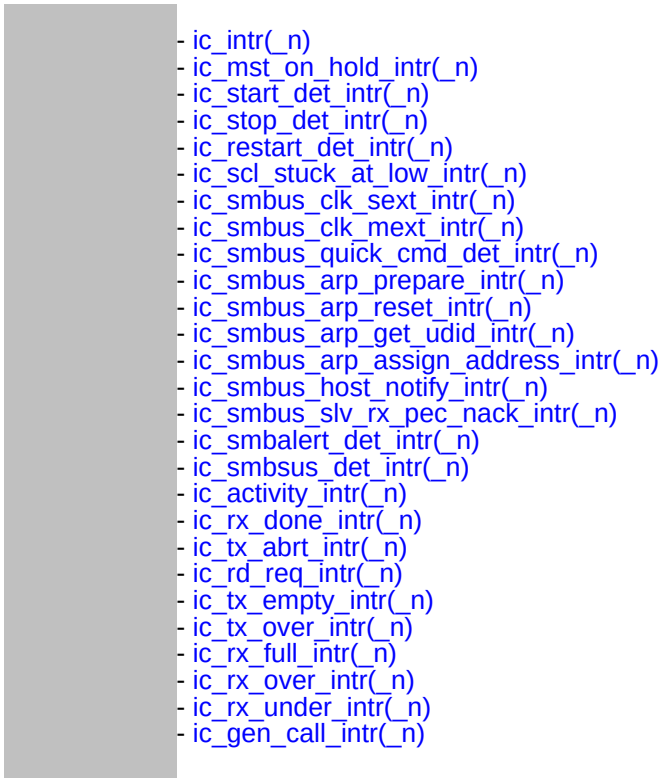


Table 4-1 Interrupts Signals

Port Name	I/O	Description
ic_intr(_n)	O	Optional. Combined interrupt. This signal is included on the interface when the configuration parameter IC_INTR_IO is unchecked (0) to indicate that only one interrupt line appears on the I/O (as opposed to individual interrupt signals). **Exists:** IC_INTR_IO == 1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low

Table 4-1 Interrupts Signals (Continued)

Port Name	I/O	Description
ic_mst_on_hold_intr(_n)	O	Optional. Optional. Master on hold I2C interrupt. This signal is included on the interface when the configuration parameters I2C_DYNAMIC_TAR_UPDATE and IC_EMPTYFIFO_HOLD_MASTER_EN are checked (1) and the configuration parameter IC_INTR_IO is unchecked (0), indicating that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & I2C_DYNAMIC_TAR_UPDATE==1 & IC_EMPTYFIFO_HOLD_MASTER_EN==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_start_det_intr(_n)	O	Optional. Start condition detect on I2C interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_stop_det_intr(_n)	O	Optional. Stop condition detect on I2C interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_restart_det_intr(_n)	O	Optional. Restart condition detect on I2C interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SLV_RESTART_DET_EN==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low

Table 4-1 Interrupts Signals (Continued)

Port Name	I/O	Description
ic_scl_stuck_at_low_intr(_n)	O	Optional. SCL Stuck condition detect on I2C interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_BUS_CLEAR_FEATURE==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbus_clk_sext_intr(_n)	O	Optional. SMBUS Slave clock extend timeout detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbus_clk_mext_intr(_n)	O	Optional. SMBUS Master clock extend timeout detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbus_quick_cmd_det_intr(_n)	O	Optional. SMBUS ARP Quick Command detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low

Table 4-1 Interrupts Signals (Continued)

Port Name	I/O	Description
ic_smbus_arp_prepare_intr(_n)	O	Optional. SMBUS ARP Prepare Command detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS_ARP==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbus_arp_reset_intr(_n)	O	Optional. SMBUS ARP Reset Command detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS_ARP==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbus_arp_get_udid_intr(_n)	O	Optional. SMBUS ARP Get UDID Command detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS_ARP==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbus_arp_assign_address_intr(_n)	O	Optional. SMBUS ARP Assign Command detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS_ARP==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low

Table 4-1 Interrupts Signals (Continued)

Port Name	I/O	Description
ic_smbus_host_notify_intr(_n)	O	Optional. SMBUS ARP Host Notify Command detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbus_slv_rx_pec_nack_intr(_n)	O	Optional. SMBUS ARP Slave Received incorrect PEC Byte and generated Nack interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS_ARP==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbalert_det_intr(_n)	O	Optional. SMBUS Alert detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS_SUSPEND_ALERT==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_smbsus_det_intr(_n)	O	Optional. SMBUS Suspend detect interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_SMBUS_SUSPEND_ALERT==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low

Table 4-1 Interrupts Signals (Continued)

Port Name	I/O	Description
ic_activity_intr(_n)	O	Optional. I2C activity interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_rx_done_intr(_n)	O	Optional. Receive done interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_ULTRA_FAST_MODE==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_tx_abrt_intr(_n)	O	Optional. Transmit abort interrupt. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_rd_req_intr(_n)	O	Optional. Slave read request interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 & IC_ULTRA_FAST_MODE==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low

Table 4-1 Interrupts Signals (Continued)

Port Name	I/O	Description
ic_tx_empty_intr(_n)	O	Optional. Transmit buffer empty interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. When bit 0 of the IC_ENABLE register is 0, the TX FIFO is flushed and held in reset, where it looks like it has no data within it. The ic_tx_empty_intr_n bit is raised when bit 0 of the IC_ENABLE register is 0, provided there is activity in the master or slave state machines. When there is no longer activity, then this interrupt bit is masked with ic_en. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_tx_over_intr(_n)	O	Optional. Transmit buffer overflow interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. When the module is disabled, this interrupt keeps its level until the master or slave state machines go into idle and bit 0 of the IC_ENABLE register is 0. When ic_en goes to 0, this interrupt is cleared. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_rx_full_intr(_n)	O	Optional. Receive buffer full interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. When bit 0 of the IC_ENABLE register is 0, the RX FIFO is flushed and held in resetthe RX FIFO is not fullso this ic_rx_full_intr_n bit is cleared once the ic_enable bit is programmed with a 0, regardless of the activity that continues. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low

Table 4-1 Interrupts Signals (Continued)

Port Name	I/O	Description
ic_rx_over_intr(_n)	O	Optional. Receive buffer overflow interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. When the module is disabled, this interrupt keeps its level until the master or slave state machines go into idle and bit 0 of the IC_ENABLE register is 0. When ic_en goes to 0, this interrupt is cleared. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_rx_under_intr(_n)	O	Optional. Receive buffer underflow interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. When the module is disabled, this interrupt keeps its level until the master or slave state machines go into idle and bit 0 of the IC_ENABLE register is 0. When ic_en goes to 0, this interrupt is cleared. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low
ic_gen_call_intr(_n)	O	Optional. General Call received interrupt. This signal is included on the interface when the configuration IC_INTR_IO parameter is unchecked (0), which indicates that individual interrupt lines appear on the I/O. **Exists:** IC_INTR_IO==0 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High when IC_INTR_POL=1 otherwise Low

4.2 I2C Interface (Master/Slave) Signals

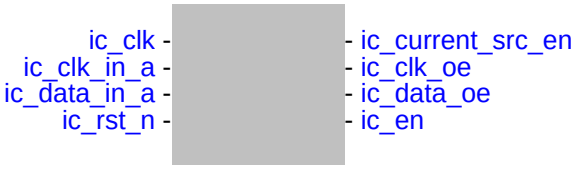


Table 4-2 I2C Interface (Master/Slave) Signals

Port Name	I/O	Description
ic_current_src_en	O	Optional. Current source pull-up. Controls the polarity of the current source pull-up on the SCLH. This pull-up is used to shorten the rise time on SCLH by activating an user-supplied external current source pull-up circuit. It is disabled after a RESTART condition and after each A/A bit when acting as the active master. This signal enables other devices to delay the serial transfer by stretching the LOW period of the SCLH signal. The active master re-enables its current source pull-up circuit again when all devices have released and the SCLH signal reaches high level, therefore, shortening the last part of the SCLH signal's rise time. **Exists:** (IC_MAX_SPEED_MODE==3) **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
ic_clk	I	Peripheral clock. DW_apb_i2c runs on this clock and is used to clock transfers in standard, fast, and high-speed mode. **Note:** ic_clk frequency must be greater than or equal to pclk frequency. The configuration parameter IC_CLK_TYPE indicates the relationship between pclk and ic_clk. It can be asynchronous (1) or identical (0). **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 I2C Interface (Master/Slave) Signals (Continued)

Port Name	I/O	Description
ic_clk_in_a	I	In (IC_ULTRA_FAST_MODE = 0) mode - Incoming I2C clock. This is the input SCL signal. Double-registered for metastability synchronization. **Note:** DW_apb_i2c provides filtering on the SDA (ic_data_in_a) and SCL (ic_clk_in_a) inputs, suppressing noise and signal spikes with durations less than one ic_clk period. In Ultra-Fast(IC_ULTRA_FAST_MODE = 1) mode - Incoming I2C clock. This is the input SCL signal. Double-registered for metastability synchronization. **Note:** DW_apb_i2c provides filtering on the SDA (ic_data_in_a) and SCL (ic_clk_in_a) inputs, suppressing noise and signal spikes with durations less than one ic_clk period. This signal is used as USCL input for slave device. This signal is asynchronous to ic_clk. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
ic_data_in_a	I	In (IC_ULTRA_FAST_MODE = 0) mode - Incoming I2C Data. It is the input SDA signal. Double-registered for metastability synchronization. **Note:** DW_apb_i2c provides filtering on the SDA (ic_data_in_a) and SCL (ic_clk_in_a) inputs, suppressing noise and signal spikes with durations less than one ic_clk period. In Ultra-Fast(IC_ULTRA_FAST_MODE = 1) mode - Incoming I2C Data. It is the input SDA signal. Double-registered for metastability synchronization. **Note:** DW_apb_i2c provides filtering on the SDA (ic_data_in_a) and SCL (ic_clk_in_a) inputs, suppressing noise and signal spikes with durations less than one ic_clk period.This signal is used as USDA input for slave device. This signal is asynchronous to ic_clk. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 I2C Interface (Master/Slave) Signals (Continued)

Port Name	I/O	Description
ic_rst_n	I	I2C reset. Used to reset flip-flops that are clocked by the ic_clk clock. **Note:** This signal does not reset DW_apb_i2c control, configuration, and status registers. The signal is asserted asynchronously, but is deasserted synchronously after the rising edge of ic_clk. The synchronization must be provided external to this component. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
ic_clk_oe	O	In (IC_ULTRA_FAST_MODE = 0) mode - Outgoing I2C clock. Open drain synchronous with ic_clk. In Ultra-Fast(IC_ULTRA_FAST_MODE = 1) mode - Outgoing I2C clock, inverted. This signal is used as USCL out from master device. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
ic_data_oe	O	In (IC_ULTRA_FAST_MODE = 0) mode - Outgoing I2C Data. Open Drain Synchronous to ic_clk. In Ultra-Fast(IC_ULTRA_FAST_MODE = 1) mode - Outgoing I2C Data, inverted. This signal is used as USDA out from master device. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
ic_en	O	I2C interface enable. Indicates whether DW_apb_i2c is enabled; this signal is set to 0 when IC_ENABLE[0] is set to 0 (disabled). Because DW_apb_i2c always finishes its current transfer before turning off ic_en, this signal may be used by a clock generator to control whether the DW_apb_i2c ic_clk is active or inactive. **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.3 APB Slave Interface Signals

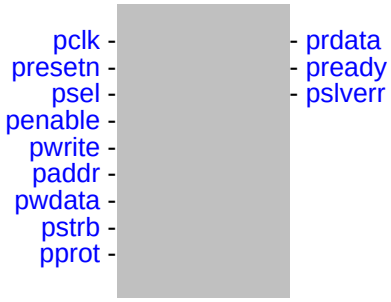


Table 4-3 APB Slave Interface Signals

Port Name	I/O	Description
pclk	I	APB clock for the bus interface unit. **Note:** ic_clk frequency must be greater than or equal to pclk frequency. The configuration parameter IC_CLK_TYPE indicates the relationship between pclk and ic_clk. It can be asynchronous (1) or identical (0). **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
presetn	I	An APB interface domain reset. The signal is asserted asynchronously, but is deasserted synchronously after the rising edge of pclk. The synchronization must be provided external to this component. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
psel	I	APB peripheral select that lasts for two pclk cycles. When asserted, indicates that the peripheral has been selected for a read/write operation. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-3 APB Slave Interface Signals (Continued)

Port Name	I/O	Description
penable	I	APB enable control. Asserted for a single pclk cycle and used for timing read/write operations. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pwrite	I	APB write control. When high, indicates a write access to the peripheral; when low, indicates a read access. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
paddr[IC_ADDR_SLICE_LHS:0]	I	APB address bus. Uses lower 7 bits of the address bus for register decode. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
pwdata[(APB_DATA_WIDTH-1):0]	I	APB write data bus. Driven by the bus master (DW_ahb to DW_apb bridge) during write cycles. Can be 8, 16, or 32 bits wide depending on APB_DATA_WIDTH parameter. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
prdata[(APB_DATA_WIDTH-1):0]	O	APB readback data. Driven by the selected peripheral during read cycles. Can be 8, 16, or 32 bits wide depending on APB_DATA_WIDTH parameter. **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-3 APB Slave Interface Signals (Continued)

Port Name	I/O	Description
pready	O	The APB ready signal, used to extend the APB transfer and it is also used to indicate the end of a transaction when there is a high in the access phase of a transaction. **Exists:** (SLAVE_INTERFACE_TYPE>0) **Synchronous To:** pclk **Registered:** (SLAVE_INTERFACE_TYPE>0 && SLVERR_RESP_EN==1) ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pslverr	O	APB slave error response signal. The signal issues an error when some error condition occurs, as specified in databook. **Exists:** (SLAVE_INTERFACE_TYPE>0) **Synchronous To:** pclk **Registered:** (SLAVE_INTERFACE_TYPE>0 && SLVERR_RESP_EN==1) ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pstrb[((APB_DATA_WIDTH/8)-1):0]	I	APB4 Write strobe bus. A high on individual bits in the pstrb bus indicate that the corresponding incoming write data byte on APB bus is to be updated in the addressed register. **Exists:** (SLAVE_INTERFACE_TYPE>1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pprot[2:0]	I	APB4 Protection type. This signal is ignored internally in DW_apb_i2c. **Exists:** (SLAVE_INTERFACE_TYPE>1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.4 DMA Interface Signals

Table 4-4 DMA Interface Signals

Port Name	I/O	Description
dma_tx_ack	I	Optional. DMA Transmit Acknowledgement. Sent by the DMA Controller to acknowledge the end of each APB transfer burst to the transmit FIFO. **Exists:** (IC_HAS_DMA==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
dma_tx_req	O	Optional. Transmit FIFO DMA Request. Asserted when the transmit FIFO requires service from the DMA Controller; that is, the transmit FIFO is at or below the watermark level. - 0 not requesting - 1 requesting Software must set up the DMA controller with the number of words to be transferred when a request is made. When using the DW_ahb_dmac, this value is programmed in the DEST_MSIZE field of the CTLx register. **Exists:** (IC_HAS_DMA==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
dma_tx_single	O	Optional. DMA Transmit FIFO Single Signal. This DMA status output informs the DMA Controller that there is at least one free entry in the transmit FIFO. This output does not request a DMA transfer. - 0: Transmit FIFO is full - 1: Transmit FIFO is not full **Exists:** (IC_HAS_DMA==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-4 DMA Interface Signals (Continued)

Port Name	I/O	Description
dma_rx_ack	I	Optional. DMA Receive Acknowledgement. Sent by the DMAcontroller to acknowledge the end of each APB transfer burst from the receive FIFO. **Exists:** (IC_HAS_DMA==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
dma_rx_req	O	Optional. Receive FIFO DMA Request. Asserted when the receive FIFO requires service from the DMA Controller; that is, the receive FIFO is at or above the watermark level. - 0 not requesting - 1 requesting Software must set up the DMA controller with the number of words to be transferred when a request is made. When using the DW_ahb_dmac, this value is programmed in the SRC_MSIZE field of the CTLx register. **Exists:** (IC_HAS_DMA==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
dma_rx_single	O	Optional. DMA Receive FIFO Single Signal. This DMA status output informs the DMA Controller that there is at least one valid data entry in the receive FIFO. This output does not request a DMA transfer. - 0: Receive FIFO is empty - 1: Receive FIFO is not empty **Exists:** (IC_HAS_DMA==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 SMBus Interface Signals

Table 4-5 SMBus Interface Signals

Port Name	I/O	Description
ic_smbsus_in_n	I	Incoming SMBus Suspend signal. This is the input SMBSUS signal. Double-registered for metastability synchronization. This signal is asynchronous to pclk. **Exists:** (IC_SMBUS_SUSPEND_ALERT==1) **Synchronous To:** Asynchronous **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
ic_smbalert_in_n	I	Incoming SMBus Alert signal. This is the input SMBALERT signal. Double-registered for metastability synchronization. This signal is asynchronous to pclk. **Exists:** (IC_SMBUS_SUSPEND_ALERT==1) **Synchronous To:** Asynchronous **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
ic_smbsus_out_n	O	Outgoing SMBus Suspend Signal. This signal is used to suspend the SMBus system, if DW_apb_i2c is used as SMBus Host. **Exists:** (IC_SMBUS_SUSPEND_ALERT==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
ic_smbalert_oe	O	Outgoing SMBus Alert Signal. This signal is used to intimate the Host that slave wants to talk, if DW_apb_i2c is used as SMBus Slave. **Exists:** (IC_SMBUS_SUSPEND_ALERT==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.6 I2C Debug Signals

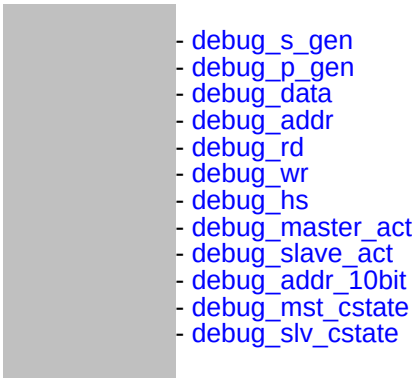


Table 4-6 I2C Debug Signals

Port Name	I/O	Description
debug_s_gen	O	In the master mode of operation, this signal is set to 1 when DW_apb_i2c is driving a START condition on the bus. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
debug_p_gen	O	In the master mode of operation, this signal is set to 1 when DW_apb_i2c is driving a STOP condition on the bus. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
debug_data	O	In the master or slave mode of operation, this signal is set to 1 when a byte of data is actively being read or written by DW_apb_i2c. This bit remains 1 until the transaction has completed. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6 I2C Debug Signals (Continued)

Port Name	I/O	Description
debug_addr	O	In the master or slave mode of operation, this signal is set to 1 when the addressing phase is active on the I2C bus. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_rd	O	In the master mode of operation, this signal is set to 1 whenever the master is receiving data. This bit remains 1 until the transfer is complete or until the direction changes. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_wr	O	In the master mode of operation, this signal is set to 1 whenever the master is transmitting data. This bit remains 1 until the transfer is complete or the direction changes. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_hs	O	In the master mode of operation, this signal is set to 1 when DW_apb_i2c is performing high-speed mode transfers. This bit is set after the high-speed master code is transmitted and remains 1 until the master leaves high-speed mode. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_master_act	O	This bit is set to 1 when the master module is active. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6 I2C Debug Signals (Continued)

Port Name	I/O	Description
debug_slave_act	O	This bit is set to 1 when the slave module is active. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_addr_10bit	O	In the Slave mode of operation, this signal is set if 10-bit addressing is enabled and if the slave has received a matching 10-bit address with respect to IC_SAR register. This signal is not applicable in Master Mode. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_mst_cstate[4:0]	O	Master FSM state vector. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_slv_cstate[3:0]	O	Slave FSM state vector. **Exists:** Always **Synchronous To:** ic_clk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

5 Register Descriptions

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

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

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

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

Exists Expressions

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

Offset

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

Memory Access Attributes

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

Table 5-1 Possible Read and Write Behaviors

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

Table 5-2 Memory Access Examples

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

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

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

Component Banks/Blocks

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

Table 5-4 Address Banks/Blocks for Memory Map: DW_apb_i2c_mem_map

Address Block	Description
DW_apb_i2c_addr_block1 on page 144	DW_apb_i2c address block **Exists:** Always

5.1 DW_apb_i2c_mem_map/DW_apb_i2c_addr_block1 Registers

DW_apb_i2c address block. Follow the link for the register to see a detailed description of the register.

Table 5-5 Registers for Address Block: DW_apb_i2c_mem_map/DW_apb_i2c_addr_block1

Register	Offset	Description
IC_CON on page 148	0x0	I2C Control Register. This register can be written only when the DW_apb_i2c is disabled, which corresponds...
IC_TAR on page 156	0x4	I2C Target Address Register If the configuration parameter I2C_DYNAMIC_TAR_UPDATE is set to 'No'...
IC_SAR on page 160	0x8	I2C Slave Address Register
IC_HS_MADDR on page 161	0xc	I2C High Speed Master Mode Code Address Register
IC_DATA_CMD on page 162	0x10	I2C Rx/Tx Data Buffer and Command Register; this is the register the CPU writes to when filling...
IC_SS_SCL_HCNT on page 166	0x14	Standard Speed I2C Clock SCL High Count Register
IC_UFM_SCL_HCNT on page 168	0x14	Ultra-Fast Speed I2C Clock SCL High Count Register
IC_SS_SCL_LCNT on page 170	0x18	Standard Speed I2C Clock SCL Low Count Register
IC_UFM_SCL_LCNT on page 172	0x18	Ultra-Fast Speed I2C Clock SCL Low Count Register
IC_FS_SCL_HCNT on page 174	0x1c	Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register
IC_UFM_TBUF_CNT on page 176	0x1c	Ultra-Fast Speed mode TBuf Idle Count Register
IC_FS_SCL_LCNT on page 178	0x20	Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register
IC_HS_SCL_HCNT on page 180	0x24	High Speed I2C Clock SCL High Count Register
IC_HS_SCL_LCNT on page 182	0x28	High Speed I2C Clock SCL Low Count Register
IC_INTR_STAT on page 184	0x2c	I2C Interrupt Status Register Each bit in this register has a corresponding mask bit in the IC_INTR_MASK...
IC_INTR_MASK on page 189	0x30	I2C Interrupt Mask Register. These bits mask their corresponding interrupt status bits. This register...
IC_RAW_INTR_STAT on page 193	0x34	I2C Raw Interrupt Status Register Unlike the IC_INTR_STAT register, these bits are not masked so...
IC_RX_TL on page 202	0x38	I2C Receive FIFO Threshold Register
IC_TX_TL on page 203	0x3c	I2C Transmit FIFO Threshold Register
IC_CLR_INTR on page 204	0x40	Clear Combined and Individual Interrupt Register
IC_CLR_RX_UNDER on page 205	0x44	Clear RX_UNDER Interrupt Register

Table 5-5 Registers for Address Block: DW_apb_i2c_mem_map/DW_apb_i2c_addr_block1 (Continued)

Register	Offset	Description
IC_CLR_RX_OVER on page 206	0x48	Clear RX_OVER Interrupt Register
IC_CLR_TX_OVER on page 207	0x4c	Clear TX_OVER Interrupt Register
IC_CLR_RD_REQ on page 208	0x50	Clear RD_REQ Interrupt Register
IC_CLR_TX_ABRT on page 209	0x54	Clear TX_ABRT Interrupt Register
IC_CLR_RX_DONE on page 210	0x58	Clear RX_DONE Interrupt Register
IC_CLR_ACTIVITY on page 211	0x5c	Clear ACTIVITY Interrupt Register
IC_CLR_STOP_DET on page 212	0x60	Clear STOP_DET Interrupt Register
IC_CLR_START_DET on page 213	0x64	Clear START_DET Interrupt Register
IC_CLR_GEN_CALL on page 214	0x68	Clear GEN_CALL Interrupt Register
IC_ENABLE on page 215	0x6c	I2C Enable Register
IC_STATUS on page 220	0x70	I2C Status Register This is a read-only register used to indicate the current transfer status...
IC_TXFLR on page 227	0x74	I2C Transmit FIFO Level Register This register contains the number of valid data entries in the...
IC_RXFLR on page 228	0x78	I2C Receive FIFO Level Register This register contains the number of valid data entries in the receive...
IC_SDA_HOLD on page 229	0x7c	I2C SDA Hold Time Length Register The bits [15:0] of this register are used to control the hold...
IC_TX_ABRT_SOURCE on page 231	0x80	I2C Transmit Abort Source Register This register has 32 bits that indicate the source of the TX_ABRT...
IC_SLV_DATA_NACK_ONLY on page 240	0x84	Generate Slave Data NACK Register The register is used to generate a NACK for the data part of...
IC_DMA_CR on page 242	0x88	DMA Control Register This register is only valid when DW_apb_i2c is configured with a set of DMA...
IC_DMA_TDLR on page 244	0x8c	DMA Transmit Data Level Register This register is only valid when the DW_apb_i2c is configured...
IC_DMA_RDLR on page 245	0x90	I2C Receive Data Level Register This register is only valid when DW_apb_i2c is configured with...
IC_SDA_SETUP on page 246	0x94	I2C SDA Setup Register This register controls the amount of time delay (in terms of number of ic_clk...
IC_ACK_GENERAL_CALL on page 248	0x98	I2C ACK General Call Register The register controls whether DW_apb_i2c responds with a ACK or NACK...

Table 5-5 Registers for Address Block: DW_apb_i2c_mem_map/DW_apb_i2c_addr_block1 (Continued)

Register	Offset	Description
IC_ENABLE_STATUS on page 249	0x9c	I2C Enable Status Register The register is used to report the DW_apb_i2c hardware status when the...
IC_FS_SPKLEN on page 253	0xa0	I2C SS, FS or FM+ spike suppression limit This register is used to store the duration, measured...
IC_UFM_SPKLEN on page 254	0xa0	I2C UFM spike suppression limit This register is used to store the duration, measured in ic_clk...
IC_HS_SPKLEN on page 256	0xa4	I2C HS spike suppression limit register This register is used to store the duration, measured in...
IC_CLR_RESTART_DET on page 258	0xa8	Clear RESTART_DET Interrupt Register
IC_SCL_STUCK_AT_LOW_TIMEOUT on page 259	0xac	I2C SCL Stuck at Low Timeout This register is used to store the duration, measured in ic_clk cycles,...
IC_SDA_STUCK_AT_LOW_TIMEOUT on page 260	0xb0	I2C SDA Stuck at Low Timeout This register is used to store the duration, measured in ic_clk cycles,...
IC_CLR_SCL_STUCK_DET on page 261	0xb4	Clear SCL Stuck at Low Detect Interrupt Register
IC_DEVICE_ID on page 262	0xb8	I2C Device-ID Register This Register contains the Device-ID of the component which includes 12-bits...
IC_SMBUS_CLK_LOW_SEXT on page 263	0xbc	SMBus Slave Clock Extend Timeout Register This Register contains the Timeout value used to determine...
IC_SMBUS_CLK_LOW_MEXT on page 264	0xc0	SMBus Master Clock Extend Timeout Register This Register contains the Timeout value used to determine...
IC_SMBUS_THIGH_MAX_IDLE_COUNT on page 265	0xc4	SMBus Master THigh MAX Bus-idle count Register This register programs the Bus-idle time period...
IC_SMBUS_INTR_STAT on page 267	0xc8	SMBUS Interrupt Status Register Each bit in this register has a corresponding mask bit in the IC_SMBUS_INTR_MASK...
IC_SMBUS_INTR_MASK on page 271	0xcc	SMBus Interrupt Mask Register
IC_SMBUS_RAW_INTR_STAT on page 275	0xd0	SMBus Raw Interrupt Status Register Unlike the IC_SMBUS_INTR_STAT register, these bits are not...
IC_CLR_SMBUS_INTR on page 280	0xd4	SMBus Clear Interrupt Register
IC_OPTIONAL_SAR on page 283	0xd8	I2C Optional Slave Address Register Optional Slave address for I2C in SMBus Mode. A same restriction...
IC_SMBUS_UDID_LSB on page 284	0xdc	SMBUS ARP UDID LSB Register This Register can be written only when the DW_apb_i2c is disabled,...
IC_SMBUS_UDID_WORD0 on page 285	0xdc	SMBUS UDID WORD0 Register This Register can be written only when the DW_apb_i2c is disabled, which...

Table 5-5 Registers for Address Block: DW_apb_i2c_mem_map/DW_apb_i2c_addr_block1 (Continued)

Register	Offset	Description
IC_SMBUS_UDID_WORD1 on page 286	0xe0	SMBUS UDID WORD1 Register This Register can be written only when the DW_apb_i2c is disabled, which...
IC_SMBUS_UDID_WORD2 on page 287	0xe4	SMBUS UDID WORD2 Register This Register can be written only when the DW_apb_i2c is disabled, which...
IC_SMBUS_UDID_WORD3 on page 288	0xe8	SMBUS UDID WORD3 Register This Register can be written only when the DW_apb_i2c is disabled, which...
REG_TIMEOUT_RST on page 289	0xf0	Name: Register timeout counter reset register Size: REG_TIMEOUT_WIDTH bits Address: 0xF0 Read/Write...
IC_COMP_PARAM_1 on page 291	0xf4	Component Parameter Register 1 Note This is a constant read-only register that contains encoded...
IC_COMP_VERSION on page 294	0xf8	I2C Component Version Register
IC_COMP_TYPE on page 295	0xfc	I2C Component Type Register

5.1.1 IC_CON

- **Name:** I2C Control Register
- **Description:** I2C Control Register. This register can be written only when the DW_apb_i2c is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect.

  Read/Write Access:

  - If configuration parameter I2C_DYNAMIC_TAR_UPDATE=1, bit 4 is read only.
  - If configuration parameter IC_RX_FULL_HLD_BUS_EN =0, bit 9 is read only.
  - If configuration parameter IC_STOP_DET_IF_MASTER_ACTIVE =0, bit 10 is read only.
  - If configuration parameter IC_BUS_CLEAR_FEATURE=0, bit 11 is read only
  - If configuration parameter IC_OPTIONAL_SAR=0, bit 16 is read only
  - If configuration parameter IC_SMBUS=0, bit 17 is read only
  - If configuration parameter IC_SMBUS_ARP=0, bits 18 and 19 are read only.
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always

31:20	19	18	17	16	15:12	11	10	9	8	7	6	5	4	3	2:1	0
RSVD_IC_CON_2	SMBUS_PERSISTENT_SLV_ADDR_EN	SMBUS_ARP_EN	SMBUS_SLAVE_QUICK_EN	OPTIONAL_SAR_CTRL	RSVD_IC_CON_1	BUS_CLEAR_FEATURE_CTRL	STOP_DET_IF_MASTER_ACTIVE	RX_FIFO_FULL_HLD_CTRL	TX_EMPTY_CTRL	STOP_DET_IFADDRESSED	IC_SLAVE_DISABLE	IC_RESTART_EN	IC_10BITADDR_MASTER	IC_10BITADDR_SLAVE	SPEED	MASTER_MODE

Table 5-6 Fields for Register: IC_CON

Bits	Name	Memory Access	Description
31:20	RSVD_IC_CON_2	R	IC_CON_2 Reserved bits - Read Only **Exists:** Always
19	SMBUS_PERSISTENT_SLV_ADDR_EN	R/W	The bit controls to enable DW_apb_i2c slave as persistent or non persistent slave. If the slave is non-PSA then DW_apb_i2c slave device clears the Address valid flag for both General and Directed Reset ARP command else the address valid flag will always set to 1. This bit is applicable only in Slave mode. **Values:** ■ 0x1 (ENABLED): SMBus Persistent Slave address control is enabled. ■ 0x0 (DISABLED): SMBus Persistent Slave address control is disabled. **Value After Reset:** IC_PERSISTANT_SLV_ADDR_DEFAULT **Exists:** IC_SMBUS_ARP==1
18	SMBUS_ARP_EN	R/W	This bit controls whether DW_apb_i2c should enable Address Resolution Logic in SMBus Mode. The Slave mode will decode the Address Resolution Protocol commands and respond to it. The DW_apb_i2c slave also includes the generation/validity of PEC byte for Address Resolution Protocol commands. This bit is applicable only in Slave mode. **Values:** ■ 0x1 (ENABLED): SMBus ARP control is enabled. ■ 0x0 (DISABLED): SMBus ARP control is disabled. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1

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

Bits	Name	Memory Access	Description
17	SMBUS_SLAVE_QUICK_EN	R/W	If this bit is set to 1, DW_apb_i2c slave only receives Quick commands in SMBus Mode. If this bit is set to 0, DW_apb_i2c slave receives all bus protocols but not Quick commands. This bit is applicable only in slave mode. **Values:** ■ 0x1 (ENABLED): SMBus SLave is enabled to receive Quick command. ■ 0x0 (DISABLED): SMBus SLave is disabled to receive Quick command. **Value After Reset:** 0x0 **Exists:** IC_SMBUS==1
16	OPTIONAL_SAR_CTRL	R/W	Enables the usage of IC_OPTIONAL_SAR register. If IC_OPTIONAL_SAR =1, IC_OPTIONAL_SAR value is used as additional slave address. User must program a valid address in IC_OPTIONAL_SAR before writing 1 to this field. If IC_OPTIONAL_SAR =0, IC_OPTIONAL_SAR value is not used as additional slave address. In this mode only one I2C slave address is used. **Values:** ■ 0x1 (ENABLED): Optional SAR Address Register is enabled. ■ 0x0 (DISABLED): Optional SAR Address Register is disabled. **Value After Reset:** 0x0 **Exists:** IC_OPTIONAL_SAR==1
15:12	RSVD_IC_CON_1	R	IC_CON_1 Reserved bits - Read Only **Exists:** Always
11	BUS_CLEAR_FEATURE_CTRL	R/W	In Master mode: ■ 1'b1: Bus Clear Feature is enabled. ■ 1'b0: Bus Clear Feature is Disabled. In Slave mode, this register bit is not applicable. **Values:** ■ 0x1 (ENABLED): Bus Clear Feature ois enabled. ■ 0x0 (DISABLED): Bus Clear Feature is disabled. **Value After Reset:** 0x0 **Exists:** IC_BUS_CLEAR_FEATURE==1

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

Bits	Name	Memory Access	Description
10	STOP_DET_IF_MASTER_ACTIVE	* Varies	In Master mode: ■ 1'b1: issues the STOP_DET interrupt only when master is active. ■ 1'b0: issues the STOP_DET irrespective of whether master is active or not. **Values:** ■ 0x1 (ENABLED): Master issues the STOP_DET interrupt only when master is active ■ 0x0 (DISABLED): Master issues the STOP_DET interrupt irrespective of whether master is active or not **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** "(IC_STOP_DET_IF_MASTER_ACTIVE==1) ? \"read-write\" : \"read-only\""
9	RX_FIFO_FULL_HLD_CTRL	* Varies	This bit controls whether DW_apb_i2c should hold the bus when the Rx FIFO is physically full to its RX_BUFFER_DEPTH, as described in the IC_RX_FULL_HLD_BUS_EN parameter. **Values:** ■ 0x1 (ENABLED): Hold bus when RX_FIFO is full ■ 0x0 (DISABLED): Overflow when RX_FIFO is full **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** "(IC_RX_FULL_HLD_BUS_EN==1) ? \"read-write\" : \"read-only\""
8	TX_EMPTY_CTRL	R/W	This bit controls the generation of the TX_EMPTY interrupt, as described in the IC_RAW_INTR_STAT register. **Values:** ■ 0x1 (ENABLED): Controlled generation of TX_EMPTY interrupt ■ 0x0 (DISABLED): Default behaviour of TX_EMPTY interrupt **Value After Reset:** 0x0 **Exists:** Always

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

Bits	Name	Memory Access	Description
7	STOP_DET_IFADDRESSED	R/W	In slave mode: ■ 1'b1: issues the STOP_DET interrrupt only when it is addressed. ■ 0'b0: issues the STOP_DET irrespective of whether it's addressed or not. **NOTE:** During a general call address, this slave does not issue the STOP_DET interrupt if STOP_DET_IF_ADDRESSED = 1'b1, even if the slave responds to the general call address by generating ACK. The STOP_DET interrupt is generated only when the transmitted address matches the slave address (SAR). **Values:** ■ 0x1 (ENABLED): slave issues STOP_DET intr only if addressed ■ 0x0 (DISABLED): slave issues STOP_DET intr always **Value After Reset:** 0x0 **Exists:** Always
6	IC_SLAVE_DISABLE	R/W	This bit controls whether I2C has its slave disabled, which means once the presetn signal is applied, then this bit takes on the value of the configuration parameter IC_SLAVE_DISABLE. You have the choice of having the slave enabled or disabled after reset is applied, which means software does not have to configure the slave. By default, the slave is always enabled (in reset state as well). If you need to disable it after reset, set this bit to 1. If this bit is set (slave is disabled), DW_apb_i2c functions only as a master and does not perform any action that requires a slave. **NOTE:** Software should ensure that if this bit is written with 0, then bit 0 should also be written with a 0. **Values:** ■ 0x1 (SLAVE_DISABLED): Slave mode is disabled ■ 0x0 (SLAVE_ENABLED): Slave mode is enabled **Value After Reset:** IC_SLAVE_DISABLE **Exists:** Always

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

Bits	Name	Memory Access	Description
5	IC_RESTART_EN	R/W	Determines whether RESTART conditions may be sent when acting as a master. Some older slaves do not support handling RESTART conditions; however, RESTART conditions are used in several DW_apb_i2c operations. When RESTART is disabled, the master is prohibited from performing the following functions: ■ Sending a START BYTE ■ Performing any high-speed mode operation ■ High-speed mode operation ■ Performing direction changes in combined format mode ■ Performing a read operation with a 10-bit address By replacing RESTART condition followed by a STOP and a subsequent START condition, split operations are broken down into multiple DW_apb_i2c transfers. If the above operations are performed, it will result in setting bit 6 (TX_ABRT) of the IC_RAW_INTR_STAT register. . **Values:** ■ 0x1 (ENABLED): Master restart enabled ■ 0x0 (DISABLED): Master restart disabled **Value After Reset:** IC_RESTART_EN **Exists:** Always
4	IC_10BITADDR_MASTER	R/W	If the I2C_DYNAMIC_TAR_UPDATE configuration parameter is set to 'No' (0), this bit is named IC_10BITADDR_MASTER and controls whether the DW_apb_i2c starts its transfers in 7- or 10-bit addressing mode when acting as a master. If I2C_DYNAMIC_TAR_UPDATE is set to 'Yes' (1), the function of this bit is handled by bit 12 of IC_TAR register, and becomes a read-only copy called IC_10BITADDR_MASTER_rd_only. ■ 0: 7-bit addressing ■ 1: 10-bit addressing **Values:** ■ 0x1 (ADDR_10BITS): Master 10Bit addressing mode ■ 0x0 (ADDR_7BITS): Master 7Bit addressing mode **Value After Reset:** IC_10BITADDR_MASTER **Exists:** I2C_DYNAMIC_TAR_UPDATE == 0

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

Bits	Name	Memory Access	Description
3	IC_10BITADDR_SLAVE	R/W	When acting as a slave, this bit controls whether the DW_apb_i2c responds to 7- or 10-bit addresses. ■ 0: 7-bit addressing. The DW_apb_i2c ignores transactions that involve 10-bit addressing; for 7-bit addressing, only the lower 7 bits of the IC_SAR register are compared. ■ 1: 10-bit addressing. The DW_apb_i2c responds to only 10-bit addressing transfers that match the full 10 bits of the IC_SAR register. **Values:** ■ 0x1 (ADDR_10BITS): Slave 10Bit addressing ■ 0x0 (ADDR_7BITS): Slave 7Bit addressing **Value After Reset:** IC_10BITADDR_SLAVE **Exists:** Always
2:1	SPEED	R/W	These bits control at which speed the DW_apb_i2c operates; its setting is relevant only if one is operating the DW_apb_i2c in master mode. Hardware protects against illegal values being programmed by software. These bits must be programmed appropriately for slave mode also, as it is used to capture correct value of spike filter as per the speed mode. This register should be programmed only with a value in the range of 1 to IC_MAX_SPEED_MODE; otherwise, hardware updates this register with the value of IC_MAX_SPEED_MODE. 1: standard mode (100 kbit/s) 2: fast mode (<=400 kbit/s) or fast mode plus (<=1000Kbit/s) 3: high speed mode (3.4 Mbit/s) **Note:** This field is not applicable when IC_ULTRA_FAST_MODE=1 **Values:** ■ 0x1 (STANDARD): Standard Speed mode of operation ■ 0x2 (FAST): Fast or Fast Plus mode of operation ■ 0x3 (HIGH): High Speed mode of operation **Value After Reset:** IC_MAX_SPEED_MODE **Exists:** Always

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

Bits	Name	Memory Access	Description
0	MASTER_MODE	R/W	This bit controls whether the DW_apb_i2c master is enabled. **NOTE:** Software should ensure that if this bit is written with '1' then bit 6 should also be written with a '1'. **Values:** ■ 0x1 (ENABLED): Master mode is enabled ■ 0x0 (DISABLED): Master mode is disabled **Value After Reset:** IC_MASTER_MODE **Exists:** Always

5.1.2 IC_TAR

- **Name:** I2C Target Address Register
- **Description:** I2C Target Address Register

  If the configuration parameter I2C_DYNAMIC_TAR_UPDATE is set to 'No' (0), this register is 12 bits wide, and bits 31:12 are reserved. This register can be written to only when IC_ENABLE[0] is set to 0.

  However, if I2C_DYNAMIC_TAR_UPDATE = 1, then the register becomes 13 bits wide. In this case, writes to IC_TAR succeed when one of the following conditions are true:

  - DW_apb_i2c is NOT enabled (IC_ENABLE[0] is set to 0); or
  - DW_apb_i2c is enabled (IC_ENABLE[0]=1); AND DW_apb_i2c is NOT engaged in any Master (tx, rx) operation (IC_STATUS[5]=0); AND DW_apb_i2c is enabled to operate in Master mode (IC_CON[0]=1); AND there are NO entries in the TX FIFO (IC_STATUS[2]=1)

  You can change the TAR address dynamically without losing the bus, only if the following conditions are met.

  - DW_apb_i2c is enabled (IC_ENABLE[0]=1); AND IC_EMPTYFIFO_HOLD_MASTER_EN configuration parameter is set to 1; AND DW_apb_i2c is enabled to operate in Master mode (IC_CON[0]=1); AND there are NO entries in the Tx FIFO and the master is in HOLD state (IC_INTR_STAT[13]=1).

  Note: If the software or application is aware that the DW_apb_i2c is not using the TAR address for the pending commands in the Tx FIFO, then it is possible to update the TAR address even while the Tx FIFO has entries (IC_STATUS[2]= 0).

  - It is not necessary to perform any write to this register if DW_apb_i2c is enabled as an I2C slave only.
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** Always

31:17	16	15:14	13	12	11	10	9:0
RSVD_IC_TAR_2	SMBUS_QUICK_CMD	RSVD_IC_TAR_1	DEVICE_ID	IC_10BITADDR_MASTER	SPECIAL	GC_OR_START	IC_TAR

Table 5-7 Fields for Register: IC_TAR

Bits	Name	Memory Access	Description
31:17	RSVD_IC_TAR_2	R	IC_TAR_2 Reserved bits - Read Only **Exists:** Always
16	SMBUS_QUICK_CMD	R/W	If bit 11 (SPECIAL) is set to 1, then this bit indicates whether a Quick command is to be performed by the DW_apb_i2c. **Values:** ■ 0x1 (ENABLED): Enables programming of QUICK-CMD transmission ■ 0x0 (DISABLED): Disables programming of QUICK-CMD transmission **Value After Reset:** 0x0 **Exists:** IC_SMBUS == 1
15:14	RSVD_IC_TAR_1	R	IC_TAR_1 Reserved bits - Read Only **Exists:** Always
13	DEVICE_ID	R/W	If bit 11 (SPECIAL) is set to 1, then this bit indicates whether a Device-ID of a particular slave mentioned in IC_TAR[9:0] is to be performed by the DW_apb_i2c Master. ■ 0: Device-ID is not performed and checks ic_tar[10] to perform either general call or START byte command ■ 1: Device-ID transfer is performed and bytes based on the number of read commands in the Tx-FIFO are received from the targeted slave and put in the Rx-FIFO. **Values:** ■ 0x1 (ENABLED): Enables programming of DEVICE-ID transmission ■ 0x0 (DISABLED): Disables programming of DEVICE-ID transmission **Value After Reset:** 0x0 **Exists:** IC_DEVICE_ID == 1

Table 5-7 Fields for Register: IC_TAR (Continued)

Bits	Name	Memory Access	Description
12	IC_10BITADDR_MASTER	R/W	This bit controls whether the DW_apb_i2c starts its transfers in 7- or 10-bit addressing mode when acting as a master. ■ 0: 7-bit addressing ■ 1: 10-bit addressing **Values:** ■ 0x1 (ADDR_10BITS): Address 10Bit transmission format ■ 0x0 (ADDR_7BITS): Address 7Bit transmission format **Value After Reset:** IC_10BITADDR_MASTER **Exists:** I2C_DYNAMIC_TAR_UPDATE
11	SPECIAL	R/W	This bit indicates whether software performs a Device-ID or General Call or START BYTE command. ■ 0: ignore bit 10 GC_OR_START and use IC_TAR normally ■ 1: perform special I2C command as specified in Device_ID or GC_OR_START bit **Values:** ■ 0x1 (ENABLED): Enables programming of GENERAL_CALL or START_BYTE transmission ■ 0x0 (DISABLED): Disables programming of GENERAL_CALL or START_BYTE transmission **Value After Reset:** 0x0 **Exists:** Always

Table 5-7 Fields for Register: IC_TAR (Continued)

Bits	Name	Memory Access	Description
10	GC_OR_START	R/W	If bit 11 (SPECIAL) is set to 1 and bit 13(Device-ID) is set to 0, then this bit indicates whether a General Call or START byte command is to be performed by the DW_apb_i2c. ■ 0: General Call Address - after issuing a General Call, only writes may be performed. Attempting to issue a read command results in setting bit 6 (TX_ABRT) of the IC_RAW_INTR_STAT register. The DW_apb_i2c remains in General Call mode until the SPECIAL bit value (bit 11) is cleared. ■ 1: START BYTE **Values:** ■ 0x1 (START_BYTE): START byte transmission ■ 0x0 (GENERAL_CALL): GENERAL_CALL byte transmission **Value After Reset:** 0x0 **Exists:** Always
9:0	IC_TAR	R/W	This is the target address for any master transaction. When transmitting a General Call, these bits are ignored. To generate a START BYTE, the CPU needs to write only once into these bits. If the IC_TAR and IC_SAR are the same, loopback exists but the FIFOs are shared between master and slave, so full loopback is not feasible. Only one direction loopback mode is supported (simplex), not duplex. A master cannot transmit to itself; it can transmit to only a slave. **Value After Reset:** IC_DEFAULT_TAR_SLAVE_ADDR **Exists:** Always

5.1.3 IC_SAR

- **Name:** I2C Slave Address Register
- **Description:** I2C Slave Address Register
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always

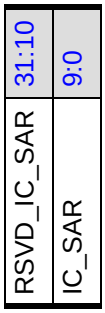


Table 5-8 Fields for Register: IC_SAR

Bits	Name	Memory Access	Description
31:10	RSVD_IC_SAR	R	IC_SAR Reserved bits - Read Only **Exists:** Always
9:0	IC_SAR	R/W	The IC_SAR holds the slave address when the I2C is operating as a slave. For 7-bit addressing, only IC_SAR[6:0] is used. This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. **Note:** The default values cannot be any of the reserved address locations: that is, 0x00 to 0x07, or 0x78 to 0x7f. The correct operation of the device is not guaranteed if you program the IC_SAR or IC_TAR to a reserved value. Refer to Table "I2C/SMBus Definition of Bits in First Byte" for a complete list of these reserved values. **Value After Reset:** IC_DEFAULT_SLAVE_ADDR **Exists:** Always

5.1.4 IC_HS_MADDR

- **Name:** I2C High Speed Master Mode Code Address Register
- **Description:** I2C High Speed Master Mode Code Address Register
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** IC_MAX_SPEED_MODE==3

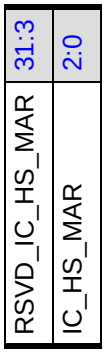


Table 5-9 Fields for Register: IC_HS_MADDR

Bits	Name	Memory Access	Description
31:3	RSVD_IC_HS_MAR	R	IC_HS_MAR Reserved bits - Read Only **Exists:** Always
2:0	IC_HS_MAR	R/W	This bit field holds the value of the I2C HS mode master code. HS-mode master codes are reserved 8-bit codes (00001xxx) that are not used for slave addressing or other purposes. Each master has its unique master code; up to eight high-speed mode masters can be present on the same I2C bus system. Valid values are from 0 to 7. This register goes away and becomes read-only returning 0's if the IC_MAX_SPEED_MODE configuration parameter is set to either Standard (1) or Fast (2). This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. **Value After Reset:** IC_HS_MASTER_CODE **Exists:** Always

5.1.5 IC_DATA_CMD

- **Name:** I2C Rx/Tx Data Buffer and Command Register
- **Description:** I2C Rx/Tx Data Buffer and Command Register; this is the register the CPU writes to when filling the TX FIFO and the CPU reads from when retrieving bytes from RX FIFO.

  The size of the register changes as follows:

  Write:

  - 11 bits when IC_EMPTYFIFO_HOLD_MASTER_EN=1
  - 9 bits when IC_EMPTYFIFO_HOLD_MASTER_EN=0

  Read:

  - 12 bits when IC_FIRST_DATA_BYTE_STATUS = 1
  - 8 bits when IC_FIRST_DATA_BYTE_STATUS = 0

  Note: In order for the DW_apb_i2c to continue acknowledging reads, a read command should be written for every byte that is to be received; otherwise the DW_apb_i2c will stop acknowledging.
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** Always

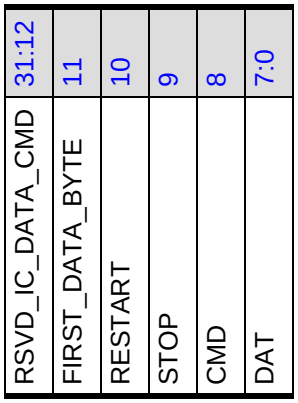


Table 5-10 Fields for Register: IC_DATA_CMD

Bits	Name	Memory Access	Description
31:12	RSVD_IC_DATA_CMD	R	IC_DATA_CMD Reserved bits - Read Only **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
11	FIRST_DATA_BYTE	R	Indicates the first data byte received after the address phase for receive transfer in Master receiver or Slave receiver mode. **NOTE:** In case of APB_DATA_WIDTH=8, 1. The user has to perform two APB Reads to IC_DATA_CMD in order to get status on 11 bit. 2. Inorder to read the 11 bit, the user has to perform the first data byte read [7:0] (offset 0x10) and then perform the second read[15:8](offset 0x11) in order to know the status of 11 bit (whether the data received in previous read is a first data byte or not). 3. The 11th bit is an optional read field, user can ignore 2nd byte read [15:8] (offset 0x11) if not interested in FIRST_DATA_BYTE status. **Values:** ■ 0x1 (ACTIVE): Non sequential data byte received ■ 0x0 (INACTIVE): Sequential data byte received **Value After Reset:** 0x0 **Exists:** IC_FIRST_DATA_BYTE_STATUS == 1 **Volatile:** true
10	RESTART	W	This bit controls whether a RESTART is issued before the byte is sent or received. This bit is available only if IC_EMPTYFIFO_HOLD_MASTER_EN is configured to 1. 1 - If IC_RESTART_EN is 1, a RESTART is issued before the data is sent/received (according to the value of CMD), regardless of whether or not the transfer direction is changing from the previous command; if IC_RESTART_EN is 0, a STOP followed by a START is issued instead. 0 - If IC_RESTART_EN is 1, a RESTART is issued only if the transfer direction is changing from the previous command; if IC_RESTART_EN is 0, a STOP followed by a START is issued instead. **Values:** ■ 0x1 (ENABLE): Issue RESTART before this command ■ 0x0 (DISABLE): Donot Issue RESTART before this command **Value After Reset:** 0x0 **Exists:** IC_EMPTYFIFO_HOLD_MASTER_EN **Volatile:** true

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

Bits	Name	Memory Access	Description
9	STOP	W	This bit controls whether a STOP is issued after the byte is sent or received. This bit is available only if IC_EMPTYFIFO_HOLD_MASTER_EN is configured to 1. ■ 1 - STOP is issued after this byte, regardless of whether or not the Tx FIFO is empty. If the Tx FIFO is not empty, the master immediately tries to start a new transfer by issuing a START and arbitrating for the bus. ■ 0 - STOP is not issued after this byte, regardless of whether or not the Tx FIFO is empty. If the Tx FIFO is not empty, the master continues the current transfer by sending/receiving data bytes according to the value of the CMD bit. If the Tx FIFO is empty, the master holds the SCL line low and stalls the bus until a new command is available in the Tx FIFO. **Values:** ■ 0x1 (ENABLE): Issue STOP after this command ■ 0x0 (DISABLE): Donot Issue STOP after this command **Value After Reset:** 0x0 **Exists:** IC_EMPTYFIFO_HOLD_MASTER_EN **Volatile:** true

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

Bits	Name	Memory Access	Description
8	CMD	W	This bit controls whether a read or a write is performed. This bit does not control the direction when the DW_apb_i2c acts as a slave. It controls only the direction when it acts as a master. When a command is entered in the TX FIFO, this bit distinguishes the write and read commands. In slave-receiver mode, this bit is a "don't care" because writes to this register are not required. In slave-transmitter mode, a "0" indicates that the data in IC_DATA_CMD is to be transmitted. When programming this bit, you should remember the following: attempting to perform a read operation after a General Call command has been sent results in a TX_ABRT interrupt (bit 6 of the IC_RAW_INTR_STAT register), unless bit 11 (SPECIAL) in the IC_TAR register has been cleared. If a "1" is written to this bit after receiving a RD_REQ interrupt, then a TX_ABRT interrupt occurs. **Values:** ■ 0x1 (READ): Master Read Command ■ 0x0 (WRITE): Master Write Command **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
7:0	DAT	R/W	This register contains the data to be transmitted or received on the I2C bus. If you are writing to this register and want to perform a read, bits 7:0 (DAT) are ignored by the DW_apb_i2c. However, when you read this register, these bits return the value of data received on the DW_apb_i2c interface. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.6 IC_SS_SCL_HCNT

- **Name:** Standard Speed I2C Clock SCL High Count Register
- **Description:** Standard Speed I2C Clock SCL High Count Register
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** IC_ULTRA_FAST_MODE==0

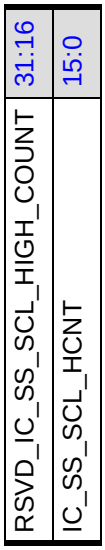


Table 5-11 Fields for Register: IC_SS_SCL_HCNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_SS_SCL_HIGH_COUNT	R	IC_SS_SCL_HCNT Reserved bits - Read Only **Exists:** Always

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

Bits	Name	Memory Access	Description
15:0	IC_SS_SCL_HCNT	* Varies	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high-period count for standard speed. For more information, refer to "IC_CLK Frequency Configuration". This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in 6 being set. For designs with APB_DATA_WIDTH = 8, the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed. When the configuration parameter IC_HC_COUNT_VALUES is set to 1, this register is read only. **NOTE:** This register must not be programmed to a value higher than 65525, because DW_apb_i2c uses a 16-bit counter to flag an I2C bus idle condition when this counter reaches a value of IC_SS_SCL_HCNT + 10. **Value After Reset:** IC_SS_SCL_HIGH_COUNT **Exists:** Always **Memory Access:** "(IC_HC_COUNT_VALUES==1) ? \"read-only\" : \"read-write\""

5.1.7 IC_UFM_SCL_HCNT

- **Name:** Ultra-Fast Speed I2C Clock SCL High Count Register
- **Description:** Ultra-Fast Speed I2C Clock SCL High Count Register
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** IC_ULTRA_FAST_MODE==1

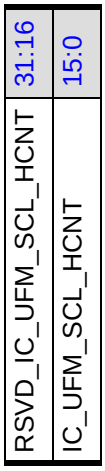


Table 5-12 Fields for Register: IC_UFM_SCL_HCNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_UFM_SCL_HCNT	R	IC_UFM_SCL_HCNT Reserved bits - Read Only **Exists:** Always

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

Bits	Name	Memory Access	Description
15:0	IC_UFM_SCL_HCNT	* Varies	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high-period count for Ultra-Fast speed. For more information, refer to "IC_CLK Frequency Configuration". This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 3; hardware prevents values less than this being written, and if attempted results in 3 being set. For designs with APB_DATA_WIDTH = 8, the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed. When the configuration parameter IC_HC_COUNT_VALUES is set to 1, this register is read only. **Value After Reset:** IC_UFM_SCL_HIGH_COUNT **Exists:** Always **Memory Access:** "(IC_HC_COUNT_VALUES==1) ? \"read-only\" : \"read-write\""

5.1.8 IC_SS_SCL_LCNT

- **Name:** Standard Speed I2C Clock SCL Low Count Register
- **Description:** Standard Speed I2C Clock SCL Low Count Register
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** IC_ULTRA_FAST_MODE==0

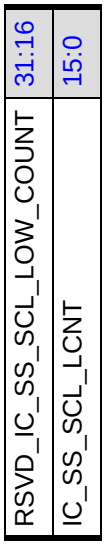


Table 5-13 Fields for Register: IC_SS_SCL_LCNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_SS_SCL_LOW_COUNT	R	RSVD_IC_SS_SCL_LOW_COUNT Reserved bits - Read Only **Exists:** Always

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

Bits	Name	Memory Access	Description
15:0	IC_SS_SCL_LCNT	* Varies	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period count for standard speed. For more information, refer to "IC_CLK Frequency Configuration" This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 8; hardware prevents values less than this being written, and if attempted, results in 8 being set. For designs with APB_DATA_WIDTH = 8, the order of programming is important to ensure the correct operation of DW_apb_i2c. The lower byte must be programmed first, and then the upper byte is programmed. When the configuration parameter IC_HC_COUNT_VALUES is set to 1, this register is read only. **Value After Reset:** IC_SS_SCL_LOW_COUNT **Exists:** Always **Memory Access:** "(IC_HC_COUNT_VALUES==1) ? \"read-only\" : \"read-write\""

5.1.9 IC_UFM_SCL_LCNT

- **Name:** Ultra-Fast Speed I2C Clock SCL Low Count Register
- **Description:** Ultra-Fast Speed I2C Clock SCL Low Count Register
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** IC_ULTRA_FAST_MODE==1

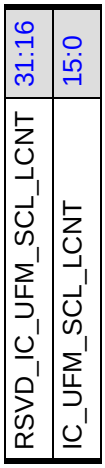


Table 5-14 Fields for Register: IC_UFM_SCL_LCNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_UFM_SCL_LCNT	R	IC_UFM_SCL_LCNT Reserved bits - Read Only **Exists:** Always

Table 5-14 Fields for Register: IC_UFM_SCL_LCNT (Continued)

Bits	Name	Memory Access	Description
15:0	IC_UFM_SCL_LCNT	* Varies	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period count for Ultra-Fast speed. This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 5; hardware prevents values less than this being written, and if attempted, results in 5 being set. For designs with APB_DATA_WIDTH = 8, the order of programming is important to ensure the correct operation of DW_apb_i2c. The lower byte must be programmed first, and then the upper byte is programmed. When the configuration parameter IC_HC_COUNT_VALUES is set to 1, this register is read only. **Value After Reset:** IC_UFM_SCL_LOW_COUNT **Exists:** Always **Memory Access:** "(IC_HC_COUNT_VALUES==1) ? \"read-only\" : \"read-write\""

5.1.10 IC_FS_SCL_HCNT

- **Name:** Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register
- **Description:** Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** IC_MAX_SPEED_MODE!=1

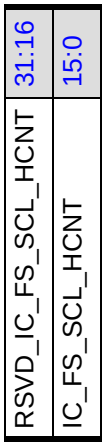


Table 5-15 Fields for Register: IC_FS_SCL_HCNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_FS_SCL_HCNT	R	IC_FS_SCL_HCNT Reserved bits - Read Only **Exists:** Always

Table 5-15 Fields for Register: IC_FS_SCL_HCNT (Continued)

Bits	Name	Memory Access	Description
15:0	IC_FS_SCL_HCNT	* Varies	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high-period count for fast mode or fast mode plus. It is used in high-speed mode to send the Master Code and START BYTE or General CALL. For more information, refer to "IC_CLK Frequency Configuration". This register goes away and becomes read-only returning 0s if IC_MAX_SPEED_MODE = standard. This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in 6 being set. For designs with APB_DATA_WIDTH == 8 the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed. **Value After Reset:** IC_FS_SCL_HIGH_COUNT **Exists:** Always **Memory Access:** "(IC_HC_COUNT_VALUES==1) ? \"read-only\" : \"read-write\""

5.1.11 IC_UFM_TBUF_CNT

- **Name:** Ultra-Fast Speed mode TBuf Idle Count Register
- **Description:** Ultra-Fast Speed mode TBuf Idle Count Register
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** IC_ULTRA_FAST_MODE==1

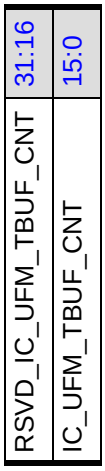


Table 5-16 Fields for Register: IC_UFM_TBUF_CNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_UFM_TBUF_CNT	R	IC_UFM_TBUF_CNT Reserved bits - Read Only **Exists:** Always

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

Bits	Name	Memory Access	Description
15:0	IC_UFM_TBUF_CNT	R/W	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the Bus-Free time between a STOP and STOP condition count for Ultra-Fast speed. This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. For designs with APB_DATA_WIDTH = 8, the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first and then the upper byte is programmed. When the configuration parameter. **NOTE:** The DW_apb_i2c will add 9 ic_clks after tBuf time is expired to generate START on the Bus. **Value After Reset:** IC_UFM_TBUF_CNT_DEFAULT **Exists:** Always

5.1.12 IC_FS_SCL_LCNT

- **Name:** Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register
- **Description:** Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register
- **Size:** 32 bits
- **Offset:** 0x20
- **Exists:** IC_MAX_SPEED_MODE!=1

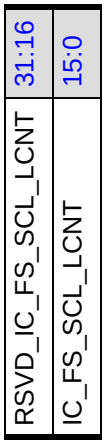


Table 5-17 Fields for Register: IC_FS_SCL_LCNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_FS_SCL_LCNT	R	IC_FS_SCL_LCNT Reserved bits - Read Only **Exists:** Always

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

Bits	Name	Memory Access	Description
15:0	IC_FS_SCL_LCNT	* Varies	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period count for fast speed. It is used in high-speed mode to send the Master Code and START BYTE or General CALL. For more information, refer to "IC_CLK Frequency Configuration". This register goes away and becomes read-only returning 0s if IC_MAX_SPEED_MODE = standard. This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 8; hardware prevents values less than this being written, and if attempted results in 8 being set. For designs with APB_DATA_WIDTH = 8 the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed. If the value is less than 8 then the count value gets changed to 8. When the configuration parameter IC_HC_COUNT_VALUES is set to 1, this register is read only. **Value After Reset:** IC_FS_SCL_LOW_COUNT **Exists:** Always **Memory Access:** "(IC_HC_COUNT_VALUES==1) ? \"read-only\" : \"read-write\""

5.1.13 IC_HS_SCL_HCNT

- **Name:** High Speed I2C Clock SCL High Count Register
- **Description:** High Speed I2C Clock SCL High Count Register
- **Size:** 32 bits
- **Offset:** 0x24
- **Exists:** IC_MAX_SPEED_MODE==3

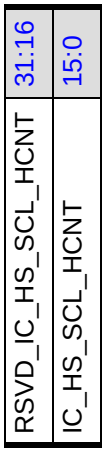


Table 5-18 Fields for Register: IC_HS_SCL_HCNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_HS_SCL_HCNT	R	IC_HS_SCL_HCNT Reserved bits - Read Only **Exists:** Always

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

Bits	Name	Memory Access	Description
15:0	IC_HS_SCL_HCNT	* Varies	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high period count for high speed.refer to "IC_CLK Frequency Configuration". The SCL High time depends on the loading of the bus. For 100pF loading, the SCL High time is 60ns; for 400pF loading, the SCL High time is 120ns. This register goes away and becomes read-only returning 0s if IC_MAX_SPEED_MODE != high. This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in 6 being set. For designs with APB_DATA_WIDTH = 8 the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed. **Value After Reset:** IC_HS_SCL_HIGH_COUNT **Exists:** Always **Memory Access:** "(IC_HC_COUNT_VALUES==1) ? \"read-only\" : \"read-write\""

5.1.14 IC_HS_SCL_LCNT

- **Name:** High Speed I2C Clock SCL Low Count Register
- **Description:** High Speed I2C Clock SCL Low Count Register
- **Size:** 32 bits
- **Offset:** 0x28
- **Exists:** IC_MAX_SPEED_MODE==3

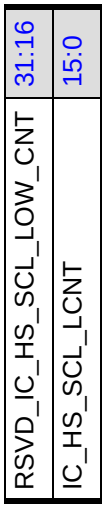


Table 5-19 Fields for Register: IC_HS_SCL_LCNT

Bits	Name	Memory Access	Description
31:16	RSVD_IC_HS_SCL_LOW_CNT	R	IC_HS_SCL_LCNT Reserved bits - Read Only **Exists:** Always

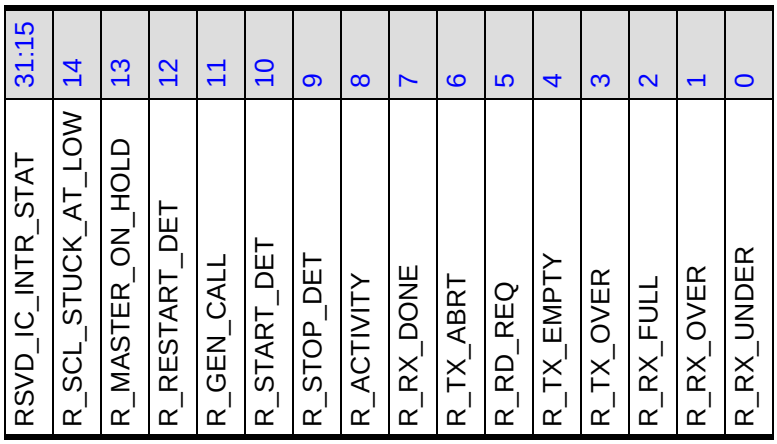
Table 5-19 Fields for Register: IC_HS_SCL_LCNT (Continued)

Bits	Name	Memory Access	Description
15:0	IC_HS_SCL_LCNT	* Varies	This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period count for high speed. For more information, refer to "IC_CLK Frequency Configuration". The SCL low time depends on the loading of the bus. For 100pF loading, the SCL low time is 160ns; for 400pF loading, the SCL low time is 320ns. This register goes away and becomes read-only returning 0s if IC_MAX_SPEED_MODE != high. This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 8; hardware prevents values less than this being written, and if attempted results in 8 being set. For designs with APB_DATA_WIDTH == 8 the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed. If the value is less than 8 then the count value gets changed to 8. **Value After Reset:** IC_HS_SCL_LOW_COUNT **Exists:** Always **Memory Access:** "(IC_HC_COUNT_VALUES==1) ? \"read-only\" : \"read-write\""

5.1.15 IC_INTR_STAT

- **Name:** I2C Interrupt Status Register
- **Description:** I2C Interrupt Status Register

  Each bit in this register has a corresponding mask bit in the IC_INTR_MASK register. These bits are cleared by reading the matching interrupt clear register. The unmasked raw versions of these bits are available in the IC_RAW_INTR_STAT register.
- **Size:** 32 bits
- **Offset:** 0x2c
- **Exists:** Always



31:15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD_IC_INTR_STAT	R_SCL_STUCK_AT_LOW	R_MASTER_ON_HOLD	R_RESTART_DET	R_GEN_CALL	R_START_DET	R_STOP_DET	R_ACTIVITY	R_RX_DONE	R_TX_ABRT	R_RD_REQ	R_TX_EMPTY	R_TX_OVER	R_RX_FULL	R_RX_OVER	R_RX_UNDER

Table 5-20 Fields for Register: IC_INTR_STAT

Bits	Name	Memory Access	Description
31:15	RSVD_IC_INTR_STAT	R	IC_INTR_STAT Reserved bits - Read Only **Exists:** Always **Volatile:** true
14	R_SCL_STUCK_AT_LOW	R	See IC_RAW_INTR_STAT for a detailed description of R_SCL_STUCK_AT_LOW bit. **Values:** ■ 0x1 (ACTIVE): R_SCL_STUCK_AT_LOW interrupt is active ■ 0x0 (INACTIVE): R_SCL_STUCK_AT_LOW interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_BUS_CLEAR_FEATURE==1 **Volatile:** true

Table 5-20 Fields for Register: IC_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
13	R_MASTER_ON_HOLD	R	See IC_RAW_INTR_STAT for a detailed description of R_MASTER_ON_HOLD bit. **Values:** ■ 0x1 (ACTIVE): R_MASTER_ON_HOLD interrupt is active ■ 0x0 (INACTIVE): R_MASTER_ON_HOLD interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
12	R_RESTART_DET	R	See IC_RAW_INTR_STAT for a detailed description of R_RESTART_DET bit. **Values:** ■ 0x1 (ACTIVE): R_RESTART_DET interrupt is active ■ 0x0 (INACTIVE): R_RESTART_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
11	R_GEN_CALL	R	See IC_RAW_INTR_STAT for a detailed description of R_GEN_CALL bit. **Values:** ■ 0x1 (ACTIVE): R_GEN_CALL interrupt is active ■ 0x0 (INACTIVE): R_GEN_CALL interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
10	R_START_DET	R	See IC_RAW_INTR_STAT for a detailed description of R_START_DET bit. **Values:** ■ 0x1 (ACTIVE): R_START_DET interrupt is active ■ 0x0 (INACTIVE): R_START_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-20 Fields for Register: IC_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
9	R_STOP_DET	R	See IC_RAW_INTR_STAT for a detailed description of R_STOP_DET bit. **Values:** ■ 0x1 (ACTIVE): R_STOP_DET interrupt is active ■ 0x0 (INACTIVE): R_STOP_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
8	R_ACTIVITY	R	See IC_RAW_INTR_STAT for a detailed description of R_ACTIVITY bit. **Values:** ■ 0x1 (ACTIVE): R_ACTIVITY interrupt is active ■ 0x0 (INACTIVE): R_ACTIVITY interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
7	R_RX_DONE	R	See IC_RAW_INTR_STAT for a detailed description of R_RX_DONE bit. **Values:** ■ 0x1 (ACTIVE): R_RX_DONE interrupt is active ■ 0x0 (INACTIVE): R_RX_DONE interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
6	R_TX_ABRT	R	See IC_RAW_INTR_STAT for a detailed description of R_TX_ABRT bit. **Values:** ■ 0x1 (ACTIVE): R_TX_ABRT interrupt is active ■ 0x0 (INACTIVE): R_TX_ABRT interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-20 Fields for Register: IC_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
5	R_RD_REQ	R	See IC_RAW_INTR_STAT for a detailed description of R_RD_REQ bit. **Values:** ■ 0x1 (ACTIVE): R_RD_REQ interrupt is active ■ 0x0 (INACTIVE): R_RD_REQ interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
4	R_TX_EMPTY	R	See IC_RAW_INTR_STAT for a detailed description of R_TX_EMPTY bit. **Values:** ■ 0x1 (ACTIVE): R_TX_EMPTY interrupt is active ■ 0x0 (INACTIVE): R_TX_EMPTY interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
3	R_TX_OVER	R	See IC_RAW_INTR_STAT for a detailed description of R_TX_OVER bit. **Values:** ■ 0x1 (ACTIVE): R_TX_OVER interrupt is active ■ 0x0 (INACTIVE): R_TX_OVER interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
2	R_RX_FULL	R	See IC_RAW_INTR_STAT for a detailed description of R_RX_FULL bit. **Values:** ■ 0x1 (ACTIVE): R_RX_FULL interrupt is active ■ 0x0 (INACTIVE): R_RX_FULL interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-20 Fields for Register: IC_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
1	R_RX_OVER	R	See IC_RAW_INTR_STAT for a detailed description of R_RX_OVER bit. **Values:** ■ 0x1 (ACTIVE): R_RX_OVER interrupt is active ■ 0x0 (INACTIVE): R_RX_OVER interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	R_RX_UNDER	R	See IC_RAW_INTR_STAT for a detailed description of R_RX_UNDER bit. **Values:** ■ 0x1 (ACTIVE): RX_UNDER interrupt is active ■ 0x0 (INACTIVE): RX_UNDER interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.16 IC_INTR_MASK

- **Name:** I2C Interrupt Mask Register
- **Description:** I2C Interrupt Mask Register.

  These bits mask their corresponding interrupt status bits. This register is active low; a value of 0 masks the interrupt, whereas a value of 1 unmasks the interrupt.
- **Size:** 32 bits
- **Offset:** 0x30
- **Exists:** Always

31:15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD_IC_INTR_STAT	M_SCL_STUCK_AT_LOW	M_MASTER_ON_HOLD	M_RESTART_DET	M_GEN_CALL	M_START_DET	M_STOP_DET	M_ACTIVITY	M_RX_DONE	M_TX_ABRT	M_RD_REQ	M_TX_EMPTY	M_TX_OVER	M_RX_FULL	M_RX_OVER	M_RX_UNDER

Table 5-21 Fields for Register: IC_INTR_MASK

Bits	Name	Memory Access	Description
31:15	RSVD_IC_INTR_STAT	R	IC_INTR_STAT Reserved bits - Read Only **Exists:** Always
14	M_SCL_STUCK_AT_LOW	R/W	This bit masks the R_SCL_STUCK_AT_LOW interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): SCL_STUCK_AT_LOW interrupt is unmasked ■ 0x0 (ENABLED): SCL_STUCK_AT_LOW interrupt is masked **Value After Reset:** "(IC_BUS_CLEAR_FEATURE_EN) ? \"0x1\" : \"0x0\"" **Exists:** IC_BUS_CLEAR_FEATURE==1

Table 5-21 Fields for Register: IC_INTR_MASK (Continued)

Bits	Name	Memory Access	Description
13	M_MASTER_ON_HOLD	R/W	This bit masks the R_MASTER_ON_HOLD interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): MASTER_ON_HOLD interrupt is unmasked ■ 0x0 (ENABLED): MASTER_ON_HOLD interrupt is masked **Value After Reset:** 0x0 **Exists:** I2C_DYNAMIC_TAR_UPDATE == 1 && IC_EMPTYFIFO_HOLD_MASTER_EN == 1
12	M_RESTART_DET	R/W	This bit masks the R_RESTART_DET interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): RESTART_DET interrupt is unmasked ■ 0x0 (ENABLED): RESTART_DET interrupt is masked **Value After Reset:** 0x0 **Exists:** IC_SLV_RESTART_DET_EN == 1
11	M_GEN_CALL	R/W	This bit masks the R_GEN_CALL interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): GEN_CALL interrupt is unmasked ■ 0x0 (ENABLED): GEN_CALL interrupt is masked **Value After Reset:** 0x1 **Exists:** Always
10	M_START_DET	R/W	This bit masks the R_START_DET interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): START_DET interrupt is unmasked ■ 0x0 (ENABLED): START_DET interrupt is masked **Value After Reset:** 0x0 **Exists:** Always
9	M_STOP_DET	R/W	This bit masks the R_STOP_DET interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): STOP_DET interrupt is unmasked ■ 0x0 (ENABLED): STOP_DET interrupt is masked **Value After Reset:** 0x0 **Exists:** Always

Table 5-21 Fields for Register: IC_INTR_MASK (Continued)

Bits	Name	Memory Access	Description
8	M_ACTIVITY	R/W	This bit masks the R_ACTIVITY interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): ACTIVITY interrupt is unmasked ■ 0x0 (ENABLED): ACTIVITY interrupt is masked **Value After Reset:** 0x0 **Exists:** Always
7	M_RX_DONE	R/W	This bit masks the R_RX_DONE interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): RX_DONE interrupt is unmasked ■ 0x0 (ENABLED): RX_DONE interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_ULTRA_FAST_MODE == 0
6	M_TX_ABRT	R/W	This bit masks the R_TX_ABRT interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): TX_ABORT interrupt is unmasked ■ 0x0 (ENABLED): TX_ABORT interrupt is masked **Value After Reset:** 0x1 **Exists:** Always
5	M_RD_REQ	R/W	This bit masks the R_RD_REQ interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): RD_REQ interrupt is unmasked ■ 0x0 (ENABLED): RD_REQ interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_ULTRA_FAST_MODE == 0
4	M_TX_EMPTY	R/W	This bit masks the R_TX_EMPTY interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): TX_EMPTY interrupt is unmasked ■ 0x0 (ENABLED): TX_EMPTY interrupt is masked **Value After Reset:** 0x1 **Exists:** Always

Table 5-21 Fields for Register: IC_INTR_MASK (Continued)

Bits	Name	Memory Access	Description
3	M_TX_OVER	R/W	This bit masks the R_TX_OVER interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): TX_OVER interrupt is unmasked ■ 0x0 (ENABLED): TX_OVER interrupt is masked **Value After Reset:** 0x1 **Exists:** Always
2	M_RX_FULL	R/W	This bit masks the R_RX_FULL interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): RX_FULL interrupt is unmasked ■ 0x0 (ENABLED): RX_FULL interrupt is masked **Value After Reset:** 0x1 **Exists:** Always
1	M_RX_OVER	R/W	This bit masks the R_RX_OVER interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): RX_OVER interrupt is unmasked ■ 0x0 (ENABLED): RX_OVER interrupt is masked **Value After Reset:** 0x1 **Exists:** Always
0	M_RX_UNDER	R/W	This bit masks the R_RX_UNDER interrupt in IC_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): RX_UNDER interrupt is unmasked ■ 0x0 (ENABLED): RX_UNDER interrupt is masked **Value After Reset:** 0x1 **Exists:** Always

5.1.17 IC_RAW_INTR_STAT

- **Name:** I2C Raw Interrupt Status Register
- **Description:** I2C Raw Interrupt Status Register

  Unlike the IC_INTR_STAT register, these bits are not masked so they always show the true status of the DW_apb_i2c.
- **Size:** 32 bits
- **Offset:** 0x34
- **Exists:** Always

31:15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD_IC_RAW_INTR_STAT	SCL_STUCK_AT_LOW	MASTER_ON_HOLD	RESTART_DET	GEN_CALL	START_DET	STOP_DET	ACTIVITY	RX_DONE	TX_ABRT	RD_REQ	TX_EMPTY	TX_OVER	RX_FULL	RX_OVER	RX_UNDER

Table 5-22 Fields for Register: IC_RAW_INTR_STAT

Bits	Name	Memory Access	Description
31:15	RSVD_IC_RAW_INTR_STAT	R	IC_RAW_INTR_STAT Reserved bits - Read Only **Exists:** Always **Volatile:** true

Table 5-22 Fields for Register: IC_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
14	SCL_STUCK_AT_LOW	R	Indicates whether the SCL Line is stuck at low for the IC_SCL_STUCK_LOW_TIMEOUT number of ic_clk periods. Enabled only when IC_BUS_CLEAR_FEATURE=1 and IC_ULTRA_FAST_MODE=0. **Values:** ■ 0x1 (ACTIVE): SCL_STUCK_AT_LOW interrupt is active ■ 0x0 (INACTIVE): SCL_STUCK_AT_LOW interrupt is inactive. **Value After Reset:** 0x0 **Exists:** IC_BUS_CLEAR_FEATURE==1 **Volatile:** true
13	MASTER_ON_HOLD	R	Indicates whether master is holding the bus and TX FIFO is empty. Enabled only when I2C_DYNAMIC_TAR_UPDATE=1 and IC_EMPTYFIFO_HOLD_MASTER_EN=1. **Values:** ■ 0x1 (ACTIVE): MASTER_ON_HOLD interrupt is active ■ 0x0 (INACTIVE): MASTER_ON_HOLD interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
12	RESTART_DET	R	Indicates whether a RESTART condition has occurred on the I2C interface when DW_apb_i2c is operating in Slave mode and the slave is being addressed. Enabled only when IC_SLV_RESTART_DET_EN=1. **Note:** However, in high-speed mode or during a START BYTE transfer, the RESTART comes before the address field as per the I2C protocol. In this case, the slave is not the addressed slave when the RESTART is issued, therefore DW_apb_i2c does not generate the RESTART_DET interrupt. **Values:** ■ 0x1 (ACTIVE): RESTART_DET interrupt is active ■ 0x0 (INACTIVE): RESTART_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-22 Fields for Register: IC_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
11	GEN_CALL	R	Set only when a General Call address is received and it is acknowledged. It stays set until it is cleared either by disabling DW_apb_i2c or when the CPU reads bit 0 of the IC_CLR_GEN_CALL register. DW_apb_i2c stores the received data in the Rx buffer. **Values:** ■ 0x1 (ACTIVE): GEN_CALL interrupt is active ■ 0x0 (INACTIVE): GEN_CALL interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
10	START_DET	R	Indicates whether a START or RESTART condition has occurred on the I2C interface regardless of whether DW_apb_i2c is operating in slave or master mode. **Values:** ■ 0x1 (ACTIVE): START_DET interrupt is active ■ 0x0 (INACTIVE): START_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-22 Fields for Register: IC_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
9	STOP_DET	R	Indicates whether a STOP condition has occurred on the I2C interface regardless of whether DW_apb_i2c is operating in slave or master mode. **In Slave Mode:** ■ If IC_CON[7]=1'b1 (STOP_DET_IFADDRESSED), the STOP_DET interrupt will be issued only if slave is addressed. **Note:** During a general call address, this slave does not issue a STOP_DET interrupt if STOP_DET_IF_ADDRESSED=1'b1, even if the slave responds to the general call address by generating ACK. The STOP_DET interrupt is generated only when the transmitted address matches the slave address (SAR). ■ If IC_CON[7]=1'b0 (STOP_DET_IFADDRESSED), the STOP_DET interrupt is issued irrespective of whether it is being addressed. **In Master Mode:** ■ If IC_CON[10]=1'b1 (STOP_DET_IF_MASTER_ACTIVE),the STOP_DET interrupt will be issued only if Master is active. ■ If IC_CON[10]=1'b0 (STOP_DET_IFADDRESSED),the STOP_DET interrupt will be issued irrespective of whether master is active or not. **Values:** ■ 0x1 (ACTIVE): STOP_DET interrupt is active ■ 0x0 (INACTIVE): STOP_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-22 Fields for Register: IC_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
8	ACTIVITY	R	This bit captures DW_apb_i2c activity and stays set until it is cleared. There are four ways to clear it: ■ Disabling the DW_apb_i2c ■ Reading the IC_CLR_ACTIVITY register ■ Reading the IC_CLR_INTR register ■ System reset Once this bit is set, it stays set unless one of the four methods is used to clear it. Even if the DW_apb_i2c module is idle, this bit remains set until cleared, indicating that there was activity on the bus. **Values:** ■ 0x1 (ACTIVE): RAW_INTR_ACTIVITY interrupt is active ■ 0x0 (INACTIVE): RAW_INTR_ACTIVITY interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
7	RX_DONE	R	When the DW_apb_i2c is acting as a slave-transmitter, this bit is set to 1 if the master does not acknowledge a transmitted byte. This occurs on the last byte of the transmission, indicating that the transmission is done. **Values:** ■ 0x1 (ACTIVE): RX_DONE interrupt is active ■ 0x0 (INACTIVE): RX_DONE interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true

Table 5-22 Fields for Register: IC_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
6	TX_ABRT	R	This bit indicates if DW_apb_i2c, as an I2C transmitter, is unable to complete the intended actions on the contents of the transmit FIFO. This situation can occur both as an I2C master or an I2C slave, and is referred to as a 'transmit abort'. When this bit is set to 1, the IC_TX_ABRT_SOURCE register indicates the reason why the transmit abort takes places. **Note:** The DW_apb_i2c flushes/resets/empties only the TX_FIFO whenever there is a transmit abort caused by any of the events tracked by the IC_TX_ABRT_SOURCE register. The Tx FIFO remains in this flushed state until the register IC_CLR_TX_ABRT is read. Once this read is performed, the Tx FIFO is then ready to accept more data bytes from the APB interface. RX FIFO flush because of TX_ABRT is controlled by the coreConsultant parameter IC_AVOID_RX_FIFO_FLUSH_ON_TX_ABRT. **Values:** ■ 0x1 (ACTIVE): TX_ABRT interrupt is active ■ 0x0 (INACTIVE): TX_ABRT interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
5	RD_REQ	R	This bit is set to 1 when DW_apb_i2c is acting as a slave and another I2C master is attempting to read data from DW_apb_i2c. The DW_apb_i2c holds the I2C bus in a wait state (SCL=0) until this interrupt is serviced, which means that the slave has been addressed by a remote master that is asking for data to be transferred. The processor must respond to this interrupt and then write the requested data to the IC_DATA_CMD register. This bit is set to 0 just after the processor reads the IC_CLR_RD_REQ register. **Values:** ■ 0x1 (ACTIVE): RD_REQ interrupt is active ■ 0x0 (INACTIVE): RD_REQ interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true

Table 5-22 Fields for Register: IC_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
4	TX_EMPTY	R	The behavior of the TX_EMPTY interrupt status differs based on the TX_EMPTY_CTRL selection in the IC_CON register. ■ When TX_EMPTY_CTRL = 0: This bit is set to 1 when the transmit buffer is at or below the threshold value set in the IC_TX_TL register. ■ When TX_EMPTY_CTRL = 1: This bit is set to 1 when the transmit buffer is at or below the threshold value set in the IC_TX_TL register and the transmission of the address/data from the internal shift register for the most recently popped command is completed. It is automatically cleared by hardware when the buffer level goes above the threshold. When IC_ENABLE[0] is set to 0, the TX FIFO is flushed and held in reset. There the TX FIFO looks like it has no data within it, so this bit is set to 1, provided there is activity in the master or slave state machines. When there is no longer any activity, then with ic_en=0, this bit is set to 0. **Values:** ■ 0x1 (ACTIVE): TX_EMPTY interrupt is active ■ 0x0 (INACTIVE): TX_EMPTY interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
3	TX_OVER	R	Set during transmit if the transmit buffer is filled to IC_TX_BUFFER_DEPTH and the processor attempts to issue another I2C command by writing to the IC_DATA_CMD register. When the module is disabled, this bit keeps its level until the master or slave state machines go into idle, and when ic_en goes to 0, this interrupt is cleared. **Values:** ■ 0x1 (ACTIVE): TX_OVER interrupt is active ■ 0x0 (INACTIVE): TX_OVER interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-22 Fields for Register: IC_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
2	RX_FULL	R	Set when the receive buffer reaches or goes above the RX_TL threshold in the IC_RX_TL register. It is automatically cleared by hardware when buffer level goes below the threshold. If the module is disabled (IC_ENABLE[0]=0), the RX FIFO is flushed and held in reset; therefore the RX FIFO is not full. So this bit is cleared once the IC_ENABLE bit 0 is programmed with a 0, regardless of the activity that continues. **Values:** ■ 0x1 (ACTIVE): RX_FULL interrupt is active ■ 0x0 (INACTIVE): RX_FULL interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
1	RX_OVER	R	Set if the receive buffer is completely filled to IC_RX_BUFFER_DEPTH and an additional byte is received from an external I2C device. The DW_apb_i2c acknowledges this, but any data bytes received after the FIFO is full are lost. If the module is disabled (IC_ENABLE[0]=0), this bit keeps its level until the master or slave state machines go into idle, and when ic_en goes to 0, this interrupt is cleared. **Note:** If the configuration parameter IC_RX_FULL_HLD_BUS_EN is enabled and bit 9 of the IC_CON register (RX_FIFO_FULL_HLD_CTRL) is programmed to HIGH, then the RX_OVER interrupt never occurs, because the Rx FIFO never overflows. **Values:** ■ 0x1 (ACTIVE): RX_OVER interrupt is active ■ 0x0 (INACTIVE): RX_OVER interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-22 Fields for Register: IC_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
0	RX_UNDER	R	Set if the processor attempts to read the receive buffer when it is empty by reading from the IC_DATA_CMD register. If the module is disabled (IC_ENABLE[0]=0), this bit keeps its level until the master or slave state machines go into idle, and when ic_en goes to 0, this interrupt is cleared. **Values:** ■ 0x1 (ACTIVE): RX_UNDER interrupt is active ■ 0x0 (INACTIVE): RX_UNDER interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.18 IC_RX_TL

- **Name:** I2C Receive FIFO Threshold Register
- **Description:** I2C Receive FIFO Threshold Register
- **Size:** 32 bits
- **Offset:** 0x38
- **Exists:** Always

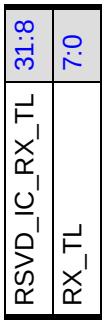


Table 5-23 Fields for Register: IC_RX_TL

Bits	Name	Memory Access	Description
31:8	RSVD_IC_RX_TL	R	IC_RX_TL Reserved bits - Read Only **Exists:** Always
7:0	RX_TL	R/W	Receive FIFO Threshold Level. Controls the level of entries (or above) that triggers the RX_FULL interrupt (bit 2 in IC_RAW_INTR_STAT register). The valid range is 0-255, with the additional restriction that hardware does not allow this value to be set to a value larger than the depth of the buffer. If an attempt is made to do that, the actual value set will be the maximum depth of the buffer. A value of 0 sets the threshold for 1 entry, and a value of 255 sets the threshold for 256 entries. **Value After Reset:** IC_RX_TL **Exists:** Always

5.1.19 IC_TX_TL

- **Name:** I2C Transmit FIFO Threshold Register
- **Description:** I2C Transmit FIFO Threshold Register
- **Size:** 32 bits
- **Offset:** 0x3c
- **Exists:** Always

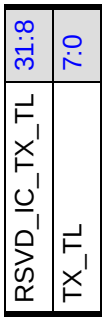


Table 5-24 Fields for Register: IC_TX_TL

Bits	Name	Memory Access	Description
31:8	RSVD_IC_TX_TL	R	IC_TX_TL Reserved bits - Read Only **Exists:** Always
7:0	TX_TL	R/W	Transmit FIFO Threshold Level. Controls the level of entries (or below) that trigger the TX_EMPTY interrupt (bit 4 in IC_RAW_INTR_STAT register). The valid range is 0-255, with the additional restriction that it may not be set to value larger than the depth of the buffer. If an attempt is made to do that, the actual value set will be the maximum depth of the buffer. A value of 0 sets the threshold for 0 entries, and a value of 255 sets the threshold for 255 entries. **Value After Reset:** IC_TX_TL **Exists:** Always

5.1.20 IC_CLR_INTR

- **Name:** Clear Combined and Individual Interrupt Register
- **Description:** Clear Combined and Individual Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x40
- **Exists:** Always

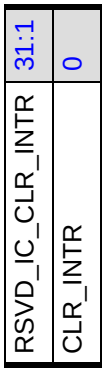


Table 5-25 Fields for Register: IC_CLR_INTR

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_INTR	R	CLR_INTR Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_INTR	R	Read this register to clear the combined interrupt, all individual interrupts, and the IC_TX_ABRT_SOURCE register. This bit does not clear hardware clearable interrupts but software clearable interrupts. Refer to Bit 9 of the IC_TX_ABRT_SOURCE register for an exception to clearing IC_TX_ABRT_SOURCE. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.21 IC_CLR_RX_UNDER

- **Name:** Clear RX_UNDER Interrupt Register
- **Description:** Clear RX_UNDER Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x44
- **Exists:** Always

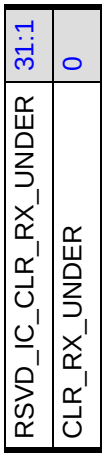


Table 5-26 Fields for Register: IC_CLR_RX_UNDER

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_RX_UNDER	R	IC_CLR_RX_UNDER Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_RX_UNDER	R	Read this register to clear the RX_UNDER interrupt (bit 0) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.22 IC_CLR_RX_OVER

- **Name:** Clear RX_OVER Interrupt Register
- **Description:** Clear RX_OVER Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x48
- **Exists:** Always

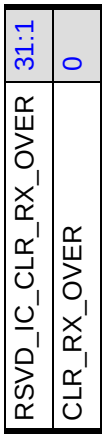


Table 5-27 Fields for Register: IC_CLR_RX_OVER

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_RX_OVER	R	IC_CLR_RX_OVER Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_RX_OVER	R	Read this register to clear the RX_OVER interrupt (bit 1) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.23 IC_CLR_TX_OVER

- **Name:** Clear TX_OVER Interrupt Register
- **Description:** Clear TX_OVER Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x4c
- **Exists:** Always

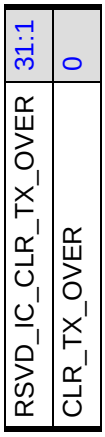


Table 5-28 Fields for Register: IC_CLR_TX_OVER

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_TX_OVER	R	IC_CLR_TX_OVER Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_TX_OVER	R	Read this register to clear the TX_OVER interrupt (bit 3) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.24 IC_CLR_RD_REQ

- **Name:** Clear RD_REQ Interrupt Register
- **Description:** Clear RD_REQ Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x50
- **Exists:** IC_ULTRA_FAST_MODE==0

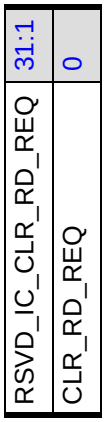


Table 5-29 Fields for Register: IC_CLR_RD_REQ

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_RD_REQ	R	IC_CLR_RD_REQ Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_RD_REQ	R	Read this register to clear the RD_REQ interrupt (bit 5) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.25 IC_CLR_TX_ABRT

- **Name:** Clear TX_ABRT Interrupt Register
- **Description:** Clear TX_ABRT Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x54
- **Exists:** Always

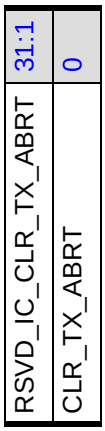


Table 5-30 Fields for Register: IC_CLR_TX_ABRT

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_TX_ABRT	R	IC_CLR_TX_ABRT Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_TX_ABRT	R	Read this register to clear the TX_ABRT interrupt (bit 6) of the IC_RAW_INTR_STAT register, and the IC_TX_ABRT_SOURCE register. This also releases the TX FIFO from the flushed/reset state, allowing more writes to the TX FIFO. Refer to Bit 9 of the IC_TX_ABRT_SOURCE register for an exception to clearing IC_TX_ABRT_SOURCE. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.26 IC_CLR_RX_DONE

- **Name:** Clear RX_DONE Interrupt Register
- **Description:** Clear RX_DONE Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x58
- **Exists:** IC_ULTRA_FAST_MODE==0

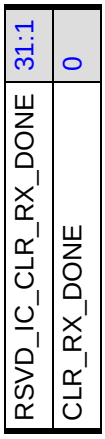


Table 5-31 Fields for Register: IC_CLR_RX_DONE

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_RX_DONE	R	IC_CLR_RX_DONE Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_RX_DONE	R	Read this register to clear the RX_DONE interrupt (bit 7) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.27 IC_CLR_ACTIVITY

- **Name:** Clear ACTIVITY Interrupt Register
- **Description:** Clear ACTIVITY Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x5c
- **Exists:** Always

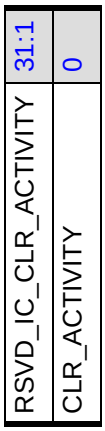


Table 5-32 Fields for Register: IC_CLR_ACTIVITY

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_ACTIVITY	R	IC_CLR_ACTIVITY Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_ACTIVITY	R	Reading this register clears the ACTIVITY interrupt if the I2C is not active anymore. If the I2C module is still active on the bus, the ACTIVITY interrupt bit continues to be set. It is automatically cleared by hardware if the module is disabled and if there is no further activity on the bus. The value read from this register to get status of the ACTIVITY interrupt (bit 8) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.28 IC_CLR_STOP_DET

- **Name:** Clear STOP_DET Interrupt Register
- **Description:** Clear STOP_DET Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x60
- **Exists:** Always

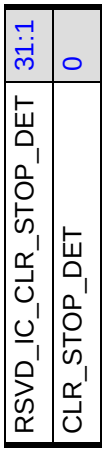


Table 5-33 Fields for Register: IC_CLR_STOP_DET

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_STOP_DET	R	IC_CLR_STOP_DET Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_STOP_DET	R	Read this register to clear the STOP_DET interrupt (bit 9) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.29 IC_CLR_START_DET

- **Name:** Clear START_DET Interrupt Register
- **Description:** Clear START_DET Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x64
- **Exists:** Always

Table 5-34 Fields for Register: IC_CLR_START_DET

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_START_DET	R	IC_CLR_START_DET Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_START_DET	R	Read this register to clear the START_DET interrupt (bit 10) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.30 IC_CLR_GEN_CALL

- **Name:** Clear GEN_CALL Interrupt Register
- **Description:** Clear GEN_CALL Interrupt Register
- **Size:** 32 bits
- **Offset:** 0x68
- **Exists:** Always

Table 5-35 Fields for Register: IC_CLR_GEN_CALL

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_GEN_CALL	R	IC_CLR_GEN_CALL Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_GEN_CALL	R	Read this register to clear the GEN_CALL interrupt (bit 11) of IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.31 IC_ENABLE

- **Name:** I2C ENABLE Register
- **Description:** I2C Enable Register
- **Size:** 32 bits
- **Offset:** 0x6c
- **Exists:** Always

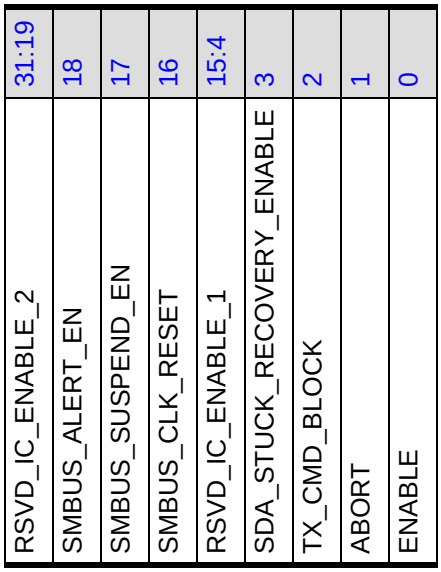


Table 5-36 Fields for Register: IC_ENABLE

Bits	Name	Memory Access	Description
31:19	RSVD_IC_ENABLE_2	R	IC_ENABLE Reserved bits - Read Only **Exists:** Always

Table 5-36 Fields for Register: IC_ENABLE (Continued)

Bits	Name	Memory Access	Description
18	SMBUS_ALERT_EN	R/W	The SMBUS_ALERT_CTRL register bit is used to control assertion of SMBALERT signal. - 1: Assert SMBALERT signal This register bit is auto-cleared after detection of Acknowledgement from master for Alert Response address. **Values:** ■ 0x1 (ALERT_ENABLED): Slave initates the Alert signal to indicate SMBus Host ■ 0x0 (SUSPEND_DISABLED): Slave will not initates the Alert signal to indicate SMBus Host. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1
17	SMBUS_SUSPEND_EN	R/W	The SMBUS_SUSPEND_EN register bit is used to control assertion and de-assertion of SMBSUS signal. ■ 0: De-assert SMBSUS signal ■ 1: Assert SMBSUS signal **Values:** ■ 0x1 (ENABLED): Host/Master initates the SMBUS system to enter Suspend Mode. ■ 0x0 (DISABLED): Host/Master will not initates the SMBUS system to enter Suspend Mode. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1
16	SMBUS_CLK_RESET	R/W	This bit is used in SMBus Host mode to initiate the SMBus Master Clock Reset. This bit should be enabled only when Master is in idle. Whenever this bit is enabled, the SMBCLK is held low for the IC_SCL_STUCK_TIMEOUT ic_clk cycles to reset the SMBus slave devices. **Values:** ■ 0x1 (ENABLED): Master initates the SMBUS Clock Reset Mechanism. ■ 0x0 (DISABLED): Master will not initates SMBUS Clock Reset Mechanism. **Value After Reset:** 0x0 **Exists:** IC_SMBUS==1
15:4	RSVD_IC_ENABLE_1	R	RSVD_IC_ENABLE_1 Reserved bits - Read Only **Exists:** Always

Table 5-36 Fields for Register: IC_ENABLE (Continued)

Bits	Name	Memory Access	Description
3	SDA_STUCK_RECOVERY_ENABLE	R/W	If SDA is stuck at low indicated through the TX_ABORT interrupt (IC_TX_ABRT_SOURCE[17]), then this bit is used as a control knob to initiate the SDA Recovery Mechanism (that is, send at most 9 SCL clocks and STOP to release the SDA line) and then this bit gets auto clear **Values:** ■ 0x1 (SDA_STUCK_RECOVERY_ENABLED): Master initates the SDA stuck at low recovery mechanism. ■ 0x0 (SDA_STUCK_RECOVERY_DISABLED): Master disabled the SDA stuck at low recovery mechanism. **Value After Reset:** 0x0 **Exists:** IC_BUS_CLEAR_FEATURE==1
2	TX_CMD_BLOCK	R/W	In Master mode: ■ 1'b1: Blocks the transmission of data on I2C bus even if Tx FIFO has data to transmit. ■ 1'b0: The transmission of data starts on I2C bus automatically, as soon as the first data is available in the Tx FIFO. **Note:** To block the execution of Master commands, set the TX_CMD_BLOCK bit only when Tx FIFO is empty (IC_STATUS[2]==1) and Master is in Idle state (IC_STATUS[5] == 0). Any further commands put in the Tx FIFO are not executed until TX_CMD_BLOCK bit is unset. **Values:** ■ 0x1 (BLOCKED): Tx Command execution blocked ■ 0x0 (NOT_BLOCKED): Tx Command execution not blocked **Value After Reset:** IC_TX_CMD_BLOCK_DEFAULT **Exists:** Always

Table 5-36 Fields for Register: IC_ENABLE (Continued)

Bits	Name	Memory Access	Description
1	ABORT	R/W	When set, the controller initiates the transfer abort. ■ 0: ABORT not initiated or ABORT done ■ 1: ABORT operation in progress The software can abort the I2C transfer in master mode by setting this bit. The software can set this bit only when ENABLE is already set; otherwise, the controller ignores any write to ABORT bit. The software cannot clear the ABORT bit once set. In response to an ABORT, the controller issues a STOP and flushes the Tx FIFO after completing the current transfer, then sets the TX_ABORT interrupt after the abort operation. The ABORT bit is cleared automatically after the abort operation. For a detailed description on how to abort I2C transfers, refer to "Aborting I2C Transfers". **Values:** ■ 0x1 (ENABLED): ABORT operation in progress ■ 0x0 (DISABLE): ABORT operation not in progress **Value After Reset:** 0x0 **Exists:** Always

Table 5-36 Fields for Register: IC_ENABLE (Continued)

Bits	Name	Memory Access	Description
0	ENABLE	R/W	Controls whether the DW_apb_i2c is enabled. ■ 0: Disables DW_apb_i2c (TX and RX FIFOs are held in an erased state) ■ 1: Enables DW_apb_i2c Software can disable DW_apb_i2c while it is active. However, it is important that care be taken to ensure that DW_apb_i2c is disabled properly. A recommended procedure is described in "Disabling DW_apb_i2c". When DW_apb_i2c is disabled, the following occurs: ■ The TX FIFO and RX FIFO get flushed. ■ Status bits in the IC_INTR_STAT register are still active until DW_apb_i2c goes into IDLE state. If the module is transmitting, it stops as well as deletes the contents of the transmit buffer after the current transfer is complete. If the module is receiving, the DW_apb_i2c stops the current transfer at the end of the current byte and does not acknowledge the transfer. In systems with asynchronous pclk and ic_clk when IC_CLK_TYPE parameter set to asynchronous (1), there is a two ic_clk delay when enabling or disabling the DW_apb_i2c. For a detailed description on how to disable DW_apb_i2c, refer to "Disabling DW_apb_i2c" **Values:** ■ 0x1 (ENABLED): I2C is enabled ■ 0x0 (DISABLED): I2C is disabled **Value After Reset:** 0x0 **Exists:** Always

5.1.32 IC_STATUS

- **Name:** I2C STATUS Register
- **Description:** I2C Status Register

  This is a read-only register used to indicate the current transfer status and FIFO status. The status register may be read at any time. None of the bits in this register request an interrupt.

  When the I2C is disabled by writing 0 in bit 0 of the IC_ENABLE register:

  - Bits 1 and 2 are set to 1
  - Bits 3 and 10 are set to 0

  When the master or slave state machines goes to idle and ic_en=0:

  - Bits 5 and 6 are set to 0
- **Size:** 32 bits
- **Offset:** 0x70
- **Exists:** Always

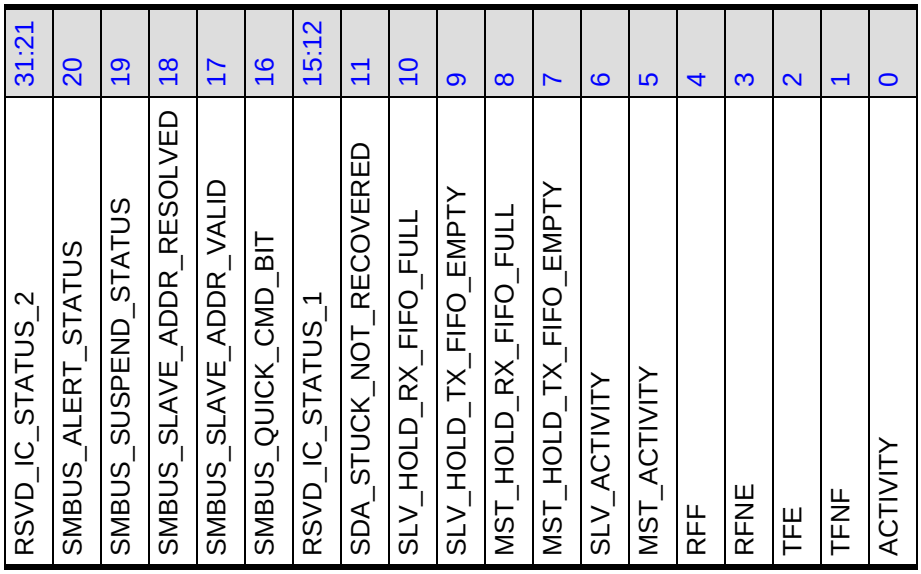


Table 5-37 Fields for Register: IC_STATUS

Bits	Name	Memory Access	Description
31:21	RSVD_IC_STATUS_2	R	IC_STATUS Reserved bits - Read Only **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
20	SMBUS_ALERT_STATUS	R	This bit indicates the status of the SMBus Alert signal (ic_smbalert_in_n). This signal is asserted when the SMBus Alert signal is asserted by the SMBus Device. **Values:** ■ 0x1 (ACTIVE): SMBUS Alert is asserted. ■ 0x0 (INACTIVE): SMBUS Alert is not asserted. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1 **Volatile:** true
19	SMBUS_SUSPEND_STATUS	R	This bit indicates the status of the SMBus Suspend signal (ic_smbsus_in_n). This signal is asserted when the SMBus Suspend signal is asserted by the SMBus Host. **Values:** ■ 0x1 (ACTIVE): SMBUS System is in Suspended mode. ■ 0x0 (INACTIVE): SMBUS System is not in Suspended mode. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1 **Volatile:** true
18	SMBUS_SLAVE_ADDR_RESOLVED	R	This bit indicates whether the slave address (ic_sar) is resolved by the ARP Master. **Values:** ■ 0x1 (ACTIVE): SMBUS Slave Address is Resolved. ■ 0x0 (INACTIVE): SMBUS Slave Address is not Resolved. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
17	SMBUS_SLAVE_ADDR_VALID	R	This bit indicates whether the slave address (ic_sar) is valid or not. **Values:** ■ 0x1 (ACTIVE): SMBUS Slave Address is Valid. ■ 0x0 (INACTIVE): SMBUS SLave Address is not valid. **Value After Reset:** IC_PERSISTANT_SLV_ADDR_DEFAULT **Exists:** IC_SMBUS_ARP==1 **Volatile:** true

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

Bits	Name	Memory Access	Description
16	SMBUS_QUICK_CMD_BIT	R	This bit indicates the R/W bit of the Quick command received. This bit will be cleared after the user has read this bit. **Values:** ■ 0x1 (ACTIVE): SMBUS QUICK CMD Read/write is set to 1. ■ 0x0 (INACTIVE): SMBUS QUICK CMD Read/write is set to 0. **Value After Reset:** 0x0 **Exists:** IC_SMBUS==1 **Volatile:** true
15:12	RSVD_IC_STATUS_1	R	RSVD_IC_STATUS_1 Reserved bits - Read Only **Exists:** Always **Volatile:** true
11	SDA_STUCK_NOT_RECOVERED	R	This bit indicates that SDA stuck at low is not recovered after the recovery mechanism. In Slave mode, this register bit is not applicable. **Values:** ■ 0x1 (ACTIVE): SDA Stuck at low is recovered after recovery mechanism. ■ 0x0 (INACTIVE): SDA Stuck at low is not recovered after recovery mechanism. **Value After Reset:** 0x0 **Exists:** IC_BUS_CLEAR_FEATURE==1 **Volatile:** true
10	SLV_HOLD_RX_FIFO_FULL	R	This bit indicates the BUS Hold in Slave mode due to Rx FIFO is Full and an additional byte has been received (This kind of Bus hold is applicable if IC_RX_FULL_HLD_BUS_EN is set to 1). **Values:** ■ 0x1 (ACTIVE): Slave holds the bus due to Rx FIFO is full ■ 0x0 (INACTIVE): Slave is not holding the bus or Bus hold is not due to Rx FIFO is full **Value After Reset:** 0x0 **Exists:** IC_STAT_FOR_CLK_STRETCH == 1 **Volatile:** true

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

Bits	Name	Memory Access	Description
9	SLV_HOLD_TX_FIFO_EMPTY	R	This bit indicates the BUS Hold in Slave mode for the Read request when the Tx FIFO is empty. The Bus is in hold until the Tx FIFO has data to Transmit for the read request. **Values:** ■ 0x1 (ACTIVE): Slave holds the bus due to Tx FIFO is empty ■ 0x0 (INACTIVE): Slave is not holding the bus or Bus hold is not due to Tx FIFO is empty **Value After Reset:** 0x0 **Exists:** IC_STAT_FOR_CLK_STRETCH == 1 **Volatile:** true
8	MST_HOLD_RX_FIFO_FULL	R	This bit indicates the BUS Hold in Master mode due to Rx FIFO is Full and additional byte has been received (This kind of Bus hold is applicable if IC_RX_FULL_HLD_BUS_EN is set to 1). **Values:** ■ 0x1 (ACTIVE): Master holds the bus due to Rx FIFO is full ■ 0x0 (INACTIVE): Master is not holding the bus or Bus hold is not due to Rx FIFO is full **Value After Reset:** 0x0 **Exists:** IC_STAT_FOR_CLK_STRETCH == 1 **Volatile:** true
7	MST_HOLD_TX_FIFO_EMPTY	R	If the IC_EMPTYFIFO_HOLD_MASTER_EN parameter is set to 1, the DW_apb_i2c master stalls the write transfer when Tx FIFO is empty, and the the last byte does not have the Stop bit set. This bit indicates the BUS hold when the master holds the bus because of the Tx FIFO being empty, and the the previous transferred command does not have the Stop bit set. (This kind of Bus hold is applicable if IC_EMPTYFIFO_HOLD_MASTER_EN is set to 1). **Values:** ■ 0x1 (ACTIVE): Master holds the bus due to Tx FIFO is empty ■ 0x0 (INACTIVE): Master is not holding the bus or Bus hold is not due to Tx FIFO is empty **Value After Reset:** 0x0 **Exists:** IC_STAT_FOR_CLK_STRETCH == 1 **Volatile:** true

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

Bits	Name	Memory Access	Description
6	SLV_ACTIVITY	R	Slave FSM Activity Status. When the Slave Finite State Machine (FSM) is not in the IDLE state, this bit is set. ■ 0: Slave FSM is in IDLE state so the Slave part of DW_apb_i2c is not Active ■ 1: Slave FSM is not in IDLE state so the Slave part of DW_apb_i2c is Active **Values:** ■ 0x1 (ACTIVE): Slave not idle ■ 0x0 (IDLE): Slave is idle **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
5	MST_ACTIVITY	R	Master FSM Activity Status. When the Master Finite State Machine (FSM) is not in the IDLE state, this bit is set. ■ 0: Master FSM is in IDLE state so the Master part of DW_apb_i2c is not Active ■ 1: Master FSM is not in IDLE state so the Master part of DW_apb_i2c is Active **Note:** IC_STATUS[0]-that is, ACTIVITY bit-is the OR of SLV_ACTIVITY and MST_ACTIVITY bits. **Values:** ■ 0x1 (ACTIVE): Master not idle ■ 0x0 (IDLE): Master is idle **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
4	RFF	R	Receive FIFO Completely 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. ■ 0: Receive FIFO is not full ■ 1: Receive FIFO is full **Values:** ■ 0x1 (FULL): Rx FIFO is full ■ 0x0 (NOT_FULL): Rx FIFO not full **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
3	RFNE	R	Receive FIFO Not Empty. This bit is set when the receive FIFO contains one or more entries; it is cleared when the receive FIFO is empty. ■ 0: Receive FIFO is empty ■ 1: Receive FIFO is not empty **Values:** ■ 0x1 (NOT_EMPTY): Rx FIFO not empty ■ 0x0 (EMPTY): Rx FIFO is empty **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
2	TFE	R	Transmit FIFO Completely Empty. When the transmit FIFO is completely empty, this bit is set. When it contains one or more valid entries, this bit is cleared. This bit field does not request an interrupt. ■ 0: Transmit FIFO is not empty ■ 1: Transmit FIFO is empty **Values:** ■ 0x1 (EMPTY): Tx FIFO is empty ■ 0x0 (NON_EMPTY): Tx FIFO not empty **Value After Reset:** 0x1 **Exists:** Always **Volatile:** true
1	TFNF	R	Transmit FIFO Not Full. Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full. ■ 0: Transmit FIFO is full ■ 1: Transmit FIFO is not full **Values:** ■ 0x1 (NOT_FULL): Tx FIFO not full ■ 0x0 (FULL): Tx FIFO is full **Value After Reset:** 0x1 **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
0	ACTIVITY	R	I2C Activity Status. **Values:** ■ 0x1 (ACTIVE): I2C is active ■ 0x0 (INACTIVE): I2C is idle **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.33 IC_TXFLR

- **Name:** I2C Transmit FIFO Level Register
- **Description:** I2C Transmit FIFO Level Register

  This register contains the number of valid data entries in the transmit FIFO buffer. It is cleared whenever:

  - The I2C is disabled
  - There is a transmit abort - that is, TX_ABRT bit is set in the IC_RAW_INTR_STAT register
  - The slave bulk transmit mode is aborted

  The register increments whenever data is placed into the transmit FIFO and decrements when data is taken from the transmit FIFO.
- **Size:** 32 bits
- **Offset:** 0x74
- **Exists:** Always

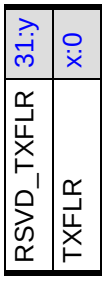


Table 5-38 Fields for Register: IC_TXFLR

Bits	Name	Memory Access	Description
31:y	RSVD_TXFLR	R	TXFLR Register field Reserved bits - Read Only **Exists:** Always **Volatile:** true **Range Variable[y]:** TX_ABW_P1
x:0	TXFLR	R	Transmit FIFO Level. Contains the number of valid data entries in the transmit FIFO. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** TX_ABW_P1 - 1

5.1.34 IC_RXFLR

- **Name:** I2C Receive FIFO Level Register
- **Description:** I2C Receive FIFO Level Register

  This register contains the number of valid data entries in the receive FIFO buffer. It is cleared whenever:

  - The I2C is disabled
  - Whenever there is a transmit abort caused by any of the events tracked in IC_TX_ABRT_SOURCE

  The register increments whenever data is placed into the receive FIFO and decrements when data is taken from the receive FIFO.
- **Size:** 32 bits
- **Offset:** 0x78
- **Exists:** Always

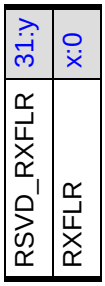


Table 5-39 Fields for Register: IC_RXFLR

Bits	Name	Memory Access	Description
31:y	RSVD_RXFLR	R	RXFLR Reserved bits - Read Only **Exists:** Always **Volatile:** true **Range Variable[y]:** RX_ABW_P1
x:0	RXFLR	R	Receive FIFO Level. Contains the number of valid data entries in the receive FIFO. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** RX_ABW_P1 - 1

5.1.35 IC_SDA_HOLD

- **Name:** I2C SDA Hold Time Length Register
- **Description:** I2C SDA Hold Time Length Register

  The bits [15:0] of this register are used to control the hold time of SDA during transmit in both slave and master mode (after SCL goes from HIGH to LOW).

  The bits [23:16] of this register are used to extend the SDA transition (if any) whenever SCL is HIGH in the receiver in either master or slave mode.

  Writes to this register succeed only when IC_ENABLE[0]=0.

  The values in this register are in units of ic_clk period. The value programmed in IC_SDA_TX_HOLD must be greater than the minimum hold time in each mode one cycle in master mode, seven cycles in slave mode for the value to be implemented.

  The programmed SDA hold time during transmit (IC_SDA_TX_HOLD) cannot exceed at any time the duration of the low part of scl. Therefore the programmed value cannot be larger than N_SCL_LOW-2, where N_SCL_LOW is the duration of the low part of the scl period measured in ic_clk cycles.

- **Size:** 32 bits
- **Offset:** 0x7c
- **Exists:** Always

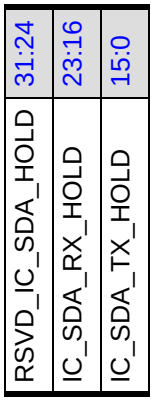


Table 5-40 Fields for Register: IC_SDA_HOLD

Bits	Name	Memory Access	Description
31:24	RSVD_IC_SDA_HOLD	R	IC_SDA_HOLD Reserved bits - Read Only **Exists:** Always

Table 5-40 Fields for Register: IC_SDA_HOLD (Continued)

Bits	Name	Memory Access	Description
23:16	IC_SDA_RX_HOLD	R/W	Sets the required SDA hold time in units of ic_clk period, when DW_apb_i2c acts as a receiver. **Value After Reset:** IC_DEFAULT_SDA_RX_HOLD **Exists:** Always
15:0	IC_SDA_TX_HOLD	R/W	Sets the required SDA hold time in units of ic_clk period, when DW_apb_i2c acts as a transmitter. **Value After Reset:** IC_DEFAULT_SDA_TX_HOLD **Exists:** Always

5.1.36 IC_TX_ABRT_SOURCE

- **Name:** I2C Transmit Abort Source Register
- **Description:** I2C Transmit Abort Source Register

  This register has 32 bits that indicate the source of the TX_ABRT bit. Except for Bit 9, this register is cleared whenever the IC_CLR_TX_ABRT register or the IC_CLR_INTR register is read. To clear Bit 9, the source of the ABRT_SBYTE_NORSTRT must be fixed first; RESTART must be enabled (IC_CON[5]=1), the SPECIAL bit must be cleared (IC_TAR[11]), or the GC_OR_START bit must be cleared (IC_TAR[10]).

  Once the source of the ABRT_SBYTE_NORSTRT is fixed, then this bit can be cleared in the same manner as other bits in this register. If the source of the ABRT_SBYTE_NORSTRT is not fixed before attempting to clear this bit, Bit 9 clears for one cycle and is then re-asserted.
- **Size:** 32 bits
- **Offset:** 0x80
- **Exists:** Always

31:23	22:21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TX_FLUSH_CNT	RSVD_IC_TX_ABRT_SOURCE	ABRT_DEVICE_WRITE	ABRT_DEVICE_SLVADDR_NOACK	ABRT_DEVICE_NOACK	ABRT_SDA_STUCK_AT_LOW	ABRT_USER_ABRT	ABRT_SLVRD_INTX	ABRT_SLV_ARBLOST	ABRT_SLVFLUSH_TXFIFO	ARB_LOST	ABRT_MASTER_DIS	ABRT_10B_RD_NORSTRT	ABRT_SBYTE_NORSTRT	ABRT_HS_NORSTRT	ABRT_SBYTE_ACKDET	ABRT_HS_ACKDET	ABRT_GCALL_READ	ABRT_GCALL_NOACK	ABRT_TXDATA_NOACK	ABRT_10ADDR2_NOACK	ABRT_10ADDR1_NOACK	ABRT_7B_ADDR_NOACK

Table 5-41 Fields for Register: IC_TX_ABRT_SOURCE

Bits	Name	Memory Access	Description
31:23	TX_FLUSH_CNT	R	This field indicates the number of Tx FIFO Data Commands which are flushed due to TX_ABRT interrupt. It is cleared whenever I2C is disabled. **Role of DW_apb_i2c:** Master-Transmitter or Slave-Transmitter **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
22:21	RSVD_IC_TX_ABRT_SOURCE	R	IC_TX_ABRT_SOURCE Reserved bits - Read Only **Exists:** Always **Volatile:** true
20	ABRT_DEVICE_WRITE	R	This is a master-mode-only bit. Master is initiating the DEVICE_ID transfer and the Tx-FIFO consists of write commands. **Role of DW_apb_i2c:** Master **Values:** ■ 0x1 (ACTIVE): This abort is generated because of NOACK for Slave address ■ 0x0 (INACTIVE): This abort is not generated **Value After Reset:** 0x0 **Exists:** IC_DEVICE_ID == 1 **Volatile:** true
19	ABRT_DEVICE_SLVADDR_NOACK	R	This is a master-mode-only bit. Master is initiating the DEVICE_ID transfer and the slave address sent was not acknowledged by any slave. **Role of DW_apb_i2c:** Master **Values:** ■ 0x1 (ACTIVE): This abort is generated because of NOACK for Slave address ■ 0x0 (INACTIVE): This abort is not generated **Value After Reset:** 0x0 **Exists:** IC_DEVICE_ID == 1 **Volatile:** true

Table 5-41 Fields for Register: IC_TX_ABRT_SOURCE (Continued)

Bits	Name	Memory Access	Description
18	ABRT_DEVICE_NOACK	R	This is a master-mode-only bit. Master is initiating the DEVICE_ID transfer and the device id sent was not acknowledged by any slave. **Role of DW_apb_i2c:** Master **Values:** ■ 0x1 (ACTIVE): This abort is generated because of NOACK for DEVICE-ID ■ 0x0 (INACTIVE): This abort is not generated **Value After Reset:** 0x0 **Exists:** IC_DEVICE_ID == 1 **Volatile:** true
17	ABRT_SDA_STUCK_AT_LOW	R	This is a master-mode-only bit. Master detects the SDA Stuck at low for the IC_SDA_STUCK_AT_LOW_TIMEOUT value of ic_clks. **Role of DW_apb_i2c:** Master **Values:** ■ 0x1 (ACTIVE): This abort is generated because of Sda stuck at low for IC_SDA_STUCK_AT_LOW_TIMEOUT value of ic_clks ■ 0x0 (INACTIVE): This abort is not generated **Value After Reset:** 0x0 **Exists:** IC_BUS_CLEAR_FEATURE == 1 **Volatile:** true
16	ABRT_USER_ABRT	R	This is a master-mode-only bit. Master has detected the transfer abort (IC_ENABLE[1]) **Role of DW_apb_i2c:** Master-Transmitter **Values:** ■ 0x1 (ABRT_USER_ABRT_GENERATED): Transfer abort detected by master ■ 0x0 (ABRT_USER_ABRT_VOID): Transfer abort detected by master- scenario not present **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-41 Fields for Register: IC_TX_ABRT_SOURCE (Continued)

Bits	Name	Memory Access	Description
15	ABRT_SLVRD_INTX	R	1: When the processor side responds to a slave mode request for data to be transmitted to a remote master and user writes a 1 in CMD (bit 8) of IC_DATA_CMD register. **Role of DW_apb_i2c:** Slave-Transmitter **Values:** ■ 0x1 (ABRT_SLVRD_INTX_GENERATED): Slave trying to transmit to remote master in read mode ■ 0x0 (ABRT_SLVRD_INTX_VOID): Slave trying to transmit to remote master in read mode- scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
14	ABRT_SLV_ARBLOST	R	This field indicates that a Slave has lost the bus while transmitting data to a remote master. IC_TX_ABRT_SOURCE[12] is set at the same time. **Note:** Even though the slave never 'owns' the bus, something could go wrong on the bus. This is a fail safe check. For instance, during a data transmission at the low-to-high transition of SCL, if what is on the data bus is not what is supposed to be transmitted, then DW_apb_i2c no longer own the bus. **Role of DW_apb_i2c:** Slave-Transmitter **Values:** ■ 0x1 (ABRT_SLV_ARBLOST_GENERATED): Slave lost arbitration to remote master ■ 0x0 (ABRT_SLV_ARBLOST_VOID): Slave lost arbitration to remote master- scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true

Table 5-41 Fields for Register: IC_TX_ABRT_SOURCE (Continued)

Bits	Name	Memory Access	Description
13	ABRT_SLVFLUSH_TXFIFO	R	This field specifies that the Slave has received a read command and some data exists in the TX FIFO, so the slave issues a TX_ABRT interrupt to flush old data in TX FIFO. **Role of DW_apb_i2c:** Slave-Transmitter **Values:** ■ 0x1 (ABRT_SLVFLUSH_TXFIFO_GENERATED): Slave flushes existing data in TX-FIFO upon getting read command ■ 0x0 (ABRT_SLVFLUSH_TXFIFO_VOID): Slave flushes existing data in TX-FIFO upon getting read command- scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
12	ARB_LOST	R	This field specifies that the Master has lost arbitration, or if IC_TX_ABRT_SOURCE[14] is also set, then the slave transmitter has lost arbitration. **Role of DW_apb_i2c:** Master-Transmitter or Slave-Transmitter **Values:** ■ 0x1 (ABRT_LOST_GENERATED): Master or Slave-Transmitter lost arbitration ■ 0x0 (ABRT_LOST_VOID): Master or Slave-Transmitter lost arbitration- scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
11	ABRT_MASTER_DIS	R	This field indicates that the User tries to initiate a Master operation with the Master mode disabled. **Role of DW_apb_i2c:** Master-Transmitter or Master-Receiver **Values:** ■ 0x1 (ABRT_MASTER_DIS_GENERATED): User intitating master operation when MASTER disabled ■ 0x0 (ABRT_MASTER_DIS_VOID): User initiating master operation when MASTER disabled- scenario not present **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-41 Fields for Register: IC_TX_ABRT_SOURCE (Continued)

Bits	Name	Memory Access	Description
10	ABRT_10B_RD_NORSTRT	R	This field indicates that the restart is disabled (IC_RESTART_EN bit (IC_CON[5]) =0) and the master sends a read command in 10-bit addressing mode. **Role of DW_apb_i2c:** Master-Receiver **Values:** ■ 0x1 (ABRT_10B_RD_GENERATED): Master trying to read in 10Bit addressing mode when RESTART disabled ■ 0x0 (ABRT_10B_RD_VOID): Master not trying to read in 10Bit addressing mode when RESTART disabled **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
9	ABRT_SBYTE_NORSTRT	R	To clear Bit 9, the source of the ABRT_SBYTE_NORSTRT must be fixed first; restart must be enabled (IC_CON[5]=1), the SPECIAL bit must be cleared (IC_TAR[11]), or the GC_OR_START bit must be cleared (IC_TAR[10]). Once the source of the ABRT_SBYTE_NORSTRT is fixed, then this bit can be cleared in the same manner as other bits in this register. If the source of the ABRT_SBYTE_NORSTRT is not fixed before attempting to clear this bit, bit 9 clears for one cycle and then gets reasserted. When this field is set to 1, the restart is disabled (IC_RESTART_EN bit (IC_CON[5]) =0) and the user is trying to send a START Byte. **Role of DW_apb_i2c:** Master **Values:** ■ 0x1 (ABRT_SBYTE_NORSTRT_GENERATED): User trying to send START byte when RESTART disabled ■ 0x0 (ABRT_SBYTE_NORSTRT_VOID): User trying to send START byte when RESTART disabled- scenario not present **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-41 Fields for Register: IC_TX_ABRT_SOURCE (Continued)

Bits	Name	Memory Access	Description
8	ABRT_HS_NORSTRT	R	This field indicates that the restart is disabled (IC_RESTART_EN bit (IC_CON[5]) =0) and the user is trying to use the master to transfer data in High Speed mode. **Role of DW_apb_i2c:** Master-Transmitter or Master-Receiver **Values:** ■ 0x1 (ABRT_HS_NORSTRT_GENERATED): User trying to switch Master to HS mode when RESTART disabled ■ 0x0 (ABRT_HS_NORSTRT_VOID): User trying to switch Master to HS mode when RESTART disabled- scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
7	ABRT_SBYTE_ACKDET	R	This field indicates that the Master has sent a START Byte and the START Byte was acknowledged (wrong behavior). **Role of DW_apb_i2c:** Master **Values:** ■ 0x1 (ABRT_SBYTE_ACKDET_GENERATED): ACK detected for START byte ■ 0x0 (ABRT_SBYTE_ACKDET_VOID): ACK detected for START byte- scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
6	ABRT_HS_ACKDET	R	This field indicates that the Master is in High Speed mode and the High Speed Master code was acknowledged (wrong behavior). **Role of DW_apb_i2c:** Master **Values:** ■ 0x1 (ABRT_HS_ACK_GENERATED): HS Master code ACKed in HS Mode ■ 0x0 (ABRT_HS_ACK_VOID): HS Master code ACKed in HS Mode- scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true

Table 5-41 Fields for Register: IC_TX_ABRT_SOURCE (Continued)

Bits	Name	Memory Access	Description
5	ABRT_GCALL_READ	R	This field indicates that DW_apb_i2c in the master mode has sent a General Call but the user programmed the byte following the General Call to be a read from the bus (IC_DATA_CMD[9] is set to 1). **Role of DW_apb_i2c:** Master-Transmitter **Values:** ■ 0x1 (ABRT_GCALL_READ_GENERATED): GCALL is followed by read from bus ■ 0x0 (ABRT_GCALL_READ_VOID): GCALL is followed by read from bus-scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
4	ABRT_GCALL_NOACK	R	This field indicates that DW_apb_i2c in master mode has sent a General Call and no slave on the bus acknowledged the General Call. **Role of DW_apb_i2c:** Master-Transmitter **Values:** ■ 0x1 (ABRT_GCALL_NOACK_GENERATED): GCALL not ACKed by any slave ■ 0x0 (ABRT_GCALL_NOACK_VOID): GCALL not ACKed by any slave-scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
3	ABRT_TXDATA_NOACK	R	This field indicates the master-mode only bit. When the master receives an acknowledgement for the address, but when it sends data byte(s) following the address, it did not receive an acknowledge from the remote slave(s). **Role of DW_apb_i2c:** Master-Transmitter **Values:** ■ 0x1 (ABRT_TXDATA_NOACK_GENERATED): Transmitted data not ACKed by addressed slave ■ 0x0 (ABRT_TXDATA_NOACK_VOID): Transmitted data non-ACKed by addressed slave-scenario not present **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true

Table 5-41 Fields for Register: IC_TX_ABRT_SOURCE (Continued)

Bits	Name	Memory Access	Description
2	ABRT_10ADDR2_NOACK	R	This field indicates that the Master is in 10-bit address mode and that the second address byte of the 10-bit address was not acknowledged by any slave. **Role of DW_apb_i2c:** Master-Transmitter or Master-Receiver **Values:** ■ 0x1 (ACTIVE): Byte 2 of 10Bit Address not ACKed by any slave ■ 0x0 (INACTIVE): This abort is not generated **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
1	ABRT_10ADDR1_NOACK	R	This field indicates that the Master is in 10-bit address mode and the first 10-bit address byte was not acknowledged by any slave. **Reset value:** 0x0 **Role of DW_apb_i2c:** Master-Transmitter or Master-Receiver **Values:** ■ 0x1 (ACTIVE): Byte 1 of 10Bit Address not ACKed by any slave ■ 0x0 (INACTIVE): This abort is not generated **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true
0	ABRT_7B_ADDR_NOACK	R	This field indicates that the Master is in 7-bit addressing mode and the address sent was not acknowledged by any slave. **Role of DW_apb_i2c:** Master-Transmitter or Master-Receiver **Values:** ■ 0x1 (ACTIVE): This abort is generated because of NOACK for 7-bit address ■ 0x0 (INACTIVE): This abort is not generated **Value After Reset:** 0x0 **Exists:** IC_ULTRA_FAST_MODE==0 **Volatile:** true

5.1.37 IC_SLV_DATA_NACK_ONLY

- **Name:** Generate Slave Data NACK Register
- **Description:** Generate Slave Data NACK Register

  The register is used to generate a NACK for the data part of a transfer when DW_apb_i2c is acting as a slave-receiver. This register only exists when the IC_SLV_DATA_NACK_ONLY parameter is set to 1. When this parameter disabled, this register does not exist and writing to the register's address has no effect.

  A write can occur on this register if both of the following conditions are met:

  - DW_apb_i2c is disabled (IC_ENABLE[0] = 0)
  - Slave part is inactive (IC_STATUS[6] = 0)

  Note: The IC_STATUS[6] is a register read-back location for the internal slv_activity signal; the user should poll this before writing the ic_slv_data_nack_only bit.
- **Size:** 32 bits
- **Offset:** 0x84
- **Exists:** [<functionof> "(IC_SLV_DATA_NACK_ONLY==0) ? 0 : 1"]

Table 5-42 Fields for Register: IC_SLV_DATA_NACK_ONLY

Bits	Name	Memory Access	Description
31:1	RSVD_IC_SLV_DATA_NACK_ONLY	R	IC_SLV_DATA_NACK_ONLY Reserved bits - Read Only **Exists:** Always

Table 5-42 Fields for Register: IC_SLV_DATA_NACK_ONLY (Continued)

Bits	Name	Memory Access	Description
0	NACK	R/W	Generate NACK. This NACK generation only occurs when DW_apb_i2c is a slave-receiver. If this register is set to a value of 1, it can only generate a NACK after a data byte is received; hence, the data transfer is aborted and the data received is not pushed to the receive buffer. When the register is set to a value of 0, it generates NACK/ACK, depending on normal criteria. ■ 1: generate NACK after data byte received ■ 0: generate NACK/ACK normally **Values:** ■ 0x1 (ENABLED): Slave reciever generates NACK upon data reception only ■ 0x0 (DISABLED): Slave reciever generates NACK normally **Value After Reset:** 0x0 **Exists:** Always

5.1.38 IC_DMA_CR

- **Name:** DMA Control Register
- **Description:** DMA Control Register

  This register is only valid when DW_apb_i2c is configured with a set of DMA Controller interface signals (IC_HAS_DMA = 1). When DW_apb_i2c is not configured for DMA operation, this register does not exist and writing to the register's address has no effect and reading from this register address will return zero. The register is used to enable the DMA Controller interface operation. There is a separate bit for transmit and receive. This can be programmed regardless of the state of IC_ENABLE.
- **Size:** 32 bits
- **Offset:** 0x88
- **Exists:** [<functionof> "(IC_HAS_DMA==1) ? 1 : 0"]

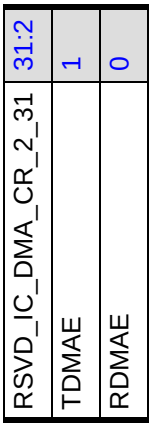


Table 5-43 Fields for Register: IC_DMA_CR

Bits	Name	Memory Access	Description
31:2	RSVD_IC_DMA_CR_2_31	R	RSVD_IC_DMA_CR_2_31 Reserved bits - Read Only **Exists:** Always
1	TDMAE	R/W	Transmit DMA Enable. This bit enables/disables the transmit FIFO DMA channel. **Values:** ■ 0x1 (ENABLED): Transmit FIFO DMA channel enabled ■ 0x0 (DISABLED): transmit FIFO DMA channel disabled **Value After Reset:** 0x0 **Exists:** Always

Table 5-43 Fields for Register: IC_DMA_CR (Continued)

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

5.1.39 IC_DMA_TDLR

- **Name:** DMA Transmit Data Level Register
- **Description:** DMA Transmit Data Level Register

  This register is only valid when the DW_apb_i2c is configured with a set of DMA interface signals (IC_HAS_DMA = 1). When DW_apb_i2c is not configured for DMA operation, this register does not exist; writing to its address has no effect; reading from its address returns zero.
- **Size:** 32 bits
- **Offset:** 0x8c
- **Exists:** IC_HAS_DMA==1

Table 5-44 Fields for Register: IC_DMA_TDLR

Bits	Name	Memory Access	Description
31:y	RSVD_DMA_TDLR	R	DMA_TDLR Reserved bits - Read Only **Exists:** Always **Range Variable[y]:** TX_ABW
x:0	DMATDL	R/W	Transmit Data Level. This bit field controls the level at which a DMA request is made by the transmit logic. It is equal to the watermark level; that is, the dma_tx_req signal is generated when the number of valid data entries in the transmit FIFO is equal to or below this field value, and TDMAE = 1. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** TX_ABW - 1

5.1.40 IC_DMA_RDLR

- **Name:** DMA Transmit Data Level Register
- **Description:** I2C Receive Data Level Register

  This register is only valid when DW_apb_i2c is configured with a set of DMA interface signals (IC_HAS_DMA = 1). When DW_apb_i2c is not configured for DMA operation, this register does not exist; writing to its address has no effect; reading from its address returns zero.
- **Size:** 32 bits
- **Offset:** 0x90
- **Exists:** [<functionof> "(IC_HAS_DMA==1) ? 1 : 0"]

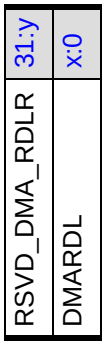


Table 5-45 Fields for Register: IC_DMA_RDLR

Bits	Name	Memory Access	Description
31:y	RSVD_DMA_RDLR	R	DMA_RDLR Reserved bits - Read Only **Exists:** Always **Range Variable[y]:** RX_ABW
x:0	DMARDL	R/W	Receive Data Level. This bit field controls the level at which a DMA request is made by the receive logic. The watermark level = DMARDL+1; that is, dma_rx_req is generated when the number of valid data entries in the receive FIFO is equal to or more than this field value + 1, and RDMAE =1. For instance, when DMARDL is 0, then dma_rx_req is asserted when 1 or more data entries are present in the receive FIFO. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** RX_ABW - 1

5.1.41 IC_SDA_SETUP

- **Name:** I2C SDA Setup Register
- **Description:** I2C SDA Setup Register

  This register controls the amount of time delay (in terms of number of ic_clk clock periods) introduced in the rising edge of SCL - relative to SDA changing - when DW_apb_i2c services a read request in a slave-transmitter operation. The relevant I2C requirement is tSU:DAT (note 4) as detailed in the I2C Bus Specification. This register must be programmed with a value equal to or greater than 2.

  Writes to this register succeed only when IC_ENABLE[0] = 0.

  Note: The length of setup time is calculated using [(IC_SDA_SETUP - 1) * (ic_clk_period)], so if the user requires 10 ic_clk periods of setup time, they should program a value of 11. The IC_SDA_SETUP register is only used by the DW_apb_i2c when operating as a slave transmitter.
- **Size:** 32 bits
- **Offset:** 0x94
- **Exists:** IC_ULTRA_FAST_MODE==0

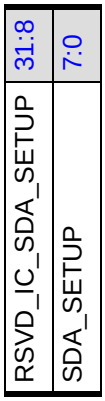


Table 5-46 Fields for Register: IC_SDA_SETUP

Bits	Name	Memory Access	Description
31:8	RSVD_IC_SDA_SETUP	R	IC_SDA_SETUP Reserved bits - Read Only **Exists:** Always

Table 5-46 Fields for Register: IC_SDA_SETUP (Continued)

Bits	Name	Memory Access	Description
7:0	SDA_SETUP	R/W	SDA Setup. It is recommended that if the required delay is 1000ns, then for an ic_clk frequency of 10 MHz, IC_SDA_SETUP should be programmed to a value of 11. IC_SDA_SETUP must be programmed with a minimum value of 2. **Value After Reset:** IC_DEFAULT_SDA_SETUP **Exists:** Always

5.1.42 IC_ACK_GENERAL_CALL

- **Name:** I2C ACK General Call Register
- **Description:** I2C ACK General Call Register

  The register controls whether DW_apb_i2c responds with a ACK or NACK when it receives an I2C General Call address.

  This register is applicable only when the DW_apb_i2c is in slave mode.
- **Size:** 32 bits
- **Offset:** 0x98
- **Exists:** IC_ULTRA_FAST_MODE==0

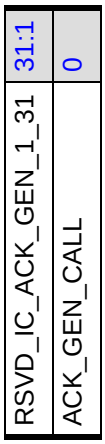


Table 5-47 Fields for Register: IC_ACK_GENERAL_CALL

Bits	Name	Memory Access	Description
31:1	RSVD_IC_ACK_GEN_1_31	R	RSVD_IC_ACK_GEN_1_31 Reserved bits - Read Only **Exists:** Always
0	ACK_GEN_CALL	R/W	ACK General Call. When set to 1, DW_apb_i2c responds with a ACK (by asserting ic_data_oe) when it receives a General Call. Otherwise, DW_apb_i2c responds with a NACK (by negating ic_data_oe). **Values:** ■ 0x1 (ENABLED): Generate ACK for a General Call ■ 0x0 (DISABLED): Generate NACK for General Call **Value After Reset:** IC_DEFAULT_ACK_GENERAL_CALL **Exists:** Always

5.1.43 IC_ENABLE_STATUS

- **Name:** I2C Enable Status Register
- **Description:** I2C Enable Status Register

  The register is used to report the DW_apb_i2c hardware status when the IC_ENABLE[0] register is set from 1 to 0; that is, when DW_apb_i2c is disabled.

  If IC_ENABLE[0] has been set to 1, bits 2:1 are forced to 0, and bit 0 is forced to 1.

  If IC_ENABLE[0] has been set to 0, bits 2:1 is only be valid as soon as bit 0 is read as '0'.

  Note: When IC_ENABLE[0] has been set to 0, a delay occurs for bit 0 to be read as 0 because disabling the DW_apb_i2c depends on I2C bus activities.
- **Size:** 32 bits
- **Offset:** 0x9c
- **Exists:** Always

Table 5-48 Fields for Register: IC_ENABLE_STATUS

Bits	Name	Memory Access	Description
31:3	RSVD_IC_ENABLE_STATUS	R	IC_ENABLE_STATUS Reserved bits - Read Only **Exists:** Always **Volatile:** true

Table 5-48 Fields for Register: IC_ENABLE_STATUS (Continued)

Bits	Name	Memory Access	Description
2	SLV_RX_DATA_LOST	R	Slave Received Data Lost. This bit indicates if a Slave-Receiver operation has been aborted with at least one data byte received from an I2C transfer due to the setting bit 0 of IC_ENABLE from 1 to 0. When read as 1, DW_apb_i2c is deemed to have been actively engaged in an aborted I2C transfer (with matching address) and the data phase of the I2C transfer has been entered, even though a data byte has been responded with a NACK. **Note:** If the remote I2C master terminates the transfer with a STOP condition before the DW_apb_i2c has a chance to NACK a transfer, and IC_ENABLE[0] has been set to 0, then this bit is also set to 1. When read as 0, DW_apb_i2c is deemed to have been disabled without being actively involved in the data phase of a Slave-Receiver transfer. **Note:** The CPU can safely read this bit when IC_EN (bit 0) is read as 0. **Values:** ■ 0x1 (ACTIVE): Slave RX Data is lost ■ 0x0 (INACTIVE): Slave RX Data is not lost **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-48 Fields for Register: IC_ENABLE_STATUS (Continued)

Bits	Name	Memory Access	Description
1	SLV_DISABLED_WHILE_BUSY	R	Slave Disabled While Busy (Transmit, Receive). This bit indicates if a potential or active Slave operation has been aborted due to the setting bit 0 of the IC_ENABLE register from 1 to 0. This bit is set when the CPU writes a 0 to the IC_ENABLE register while: (a) DW_apb_i2c is receiving the address byte of the Slave-Transmitter operation from a remote master; OR, (b) address and data bytes of the Slave-Receiver operation from a remote master. When read as 1, DW_apb_i2c is deemed to have forced a NACK during any part of an I2C transfer, irrespective of whether the I2C address matches the slave address set in DW_apb_i2c (IC_SAR register) OR if the transfer is completed before IC_ENABLE is set to 0 but has not taken effect. **Note:** If the remote I2C master terminates the transfer with a STOP condition before the DW_apb_i2c has a chance to NACK a transfer, and IC_ENABLE[0] has been set to 0, then this bit will also be set to 1. When read as 0, DW_apb_i2c is deemed to have been disabled when there is master activity, or when the I2C bus is idle. **Note:** The CPU can safely read this bit when IC_EN (bit 0) is read as 0. **Values:** ■ 0x1 (ACTIVE): Slave is disabled when it is active ■ 0x0 (INACTIVE): Slave is disabled when it is idle **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-48 Fields for Register: IC_ENABLE_STATUS (Continued)

Bits	Name	Memory Access	Description
0	IC_EN	R	ic_en Status. This bit always reflects the value driven on the output port ic_en. ■ When read as 1, DW_apb_i2c is deemed to be in an enabled state. ■ When read as 0, DW_apb_i2c is deemed completely inactive. **Note:** The CPU can safely read this bit anytime. When this bit is read as 0, the CPU can safely read SLV_RX_DATA_LOST (bit 2) and SLV_DISABLED_WHILE_BUSY (bit 1). **Values:** ■ 0x1 (ENABLED): I2C enabled ■ 0x0 (DISABLED): I2C disabled **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.44 IC_FS_SPKLEN

- **Name:** I2C SS, FS or FM+ spike suppression limit
- **Description:** I2C SS, FS or FM+ spike suppression limit

  This register is used to store the duration, measured in ic_clk cycles, of the longest spike that is filtered out by the spike suppression logic w hen the component is operating in SS, FS or FM+ modes. The relevant I2C requirement is tSP (table 4) as detailed in the I2C Bus Specification. This register must be programmed with a minimum value of 1.
- **Size:** 32 bits
- **Offset:** 0xa0
- **Exists:** IC_ULTRA_FAST_MODE==0

Table 5-49 Fields for Register: IC_FS_SPKLEN

Bits	Name	Memory Access	Description
31:8	RSVD_IC_FS_SPKLEN	R	IC_FS_SPKLEN Reserved bits - Read Only **Exists:** Always
7:0	IC_FS_SPKLEN	R/W	This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in ic_clk cycles, of the longest spike in the SCL or SDA lines that will be filtered out by the spike suppression logic. This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 1; hardware prevents values less than this being written, and if attempted results in 1 being set. or more information, refer to "Spike Suppression". **Value After Reset:** IC_DEFAULT_FS_SPKLEN **Exists:** Always

5.1.45 IC_UFM_SPKLEN

- **Name:** I2C Ultra-Fast mode spike suppression limit
- **Description:** I2C UFM spike suppression limit

  This register is used to store the duration, measured in ic_clk cycles, of the longest spike that is filtered out by the spike suppression logic when the component is operating in Ultra-Fast mode. The relevant I2C requirement is tSP (table 13) as detailed in the I2C Bus Specification. This register must be programmed with a minimum value of 1.
- **Size:** 32 bits
- **Offset:** 0xa0
- **Exists:** IC_ULTRA_FAST_MODE==1

Table 5-50 Fields for Register: IC_UFM_SPKLEN

Bits	Name	Memory Access	Description
31:8	RSVD_IC_UFM_SPKLEN	R	IC_UFM_SPKLEN Reserved bits - Read Only **Exists:** Always

Table 5-50 Fields for Register: IC_UFM_SPKLEN (Continued)

Bits	Name	Memory Access	Description
7:0	IC_UFM_SPKLEN	R/W	This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in ic_clk cycles, of the longest spike in the SCL or SDA lines that will be filtered out by the spike suppression logic. This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 1; hardware prevents values less than this being written, and if attempted results in 1 being set. **Value After Reset:** IC_DEFAULT_UFM_SPKLEN **Exists:** Always

5.1.46 IC_HS_SPKLEN

- **Name:** I2C HS spike suppression limit register
- **Description:** I2C HS spike suppression limit register

  This register is used to store the duration, measured in ic_clk cycles, of the longest spike that is filtered out by the spike suppression logic when the component is operating in HS modes. The relevant I2C requirement is tSP (table 6) as detailed in the I2C Bus Specification. This register must be programmed with a minimum value of 1 and is implemented only if the component is configured to support HS mode; that is, if the IC_MAX_SPEED_MODE parameter is set to 3.
- **Size:** 32 bits
- **Offset:** 0xa4
- **Exists:** IC_HIGHSPEED_MODE_EN

Table 5-51 Fields for Register: IC_HS_SPKLEN

Bits	Name	Memory Access	Description
31:8	RSVD_IC_HS_SPKLEN	R	IC_HS_SPKLEN Reserved bits - Read Only **Exists:** Always

Table 5-51 Fields for Register: IC_HS_SPKLEN (Continued)

Bits	Name	Memory Access	Description
7:0	IC_HS_SPKLEN	R/W	This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in ic_clk cycles, of the longest spike in the SCL or SDA lines that will be filtered out by the spike suppression logic; for more information, refer to "Spike Suppression" This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 1; hardware prevents values less than this being written, and if attempted results in 1 being set. **Value After Reset:** IC_DEFAULT_HS_SPKLEN **Exists:** Always

5.1.47 IC_CLR_RESTART_DET

- **Name:** Clear RESTART_DET Interrupt Register
- **Description:** Clear RESTART_DET Interrupt Register
- **Size:** 32 bits
- **Offset:** 0xa8
- **Exists:** IC_SLV_RESTART_DET_EN == 1

Table 5-52 Fields for Register: IC_CLR_RESTART_DET

Bits	Name	Memory Access	Description
31:1	RSVD_IC_CLR_RESTART_DET	R	IC_CLR_RESTART_DET Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_RESTART_DET	R	Read this register to clear the RESTART_DET interrupt (bit 12) of IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.48 IC_SCL_STUCK_AT_LOW_TIMEOUT

- **Name:** I2C SCL Stuck at Low Timeout register
- **Description:** I2C SCL Stuck at Low Timeout

  This register is used to store the duration, measured in ic_clk cycles, used to Generate an Interrupt (SCL_STUCK_AT_LOW) if SCL is held low for the IC_SCL_STUCK_LOW_TIMEOUT duration.
- **Size:** 32 bits
- **Offset:** 0xac
- **Exists:** IC_BUS_CLEAR_FEATURE==1

Table 5-53 Fields for Register: IC_SCL_STUCK_AT_LOW_TIMEOUT

Bits	Name	Memory Access	Description
31:0	IC_SCL_STUCK_LOW_TIMEOUT	R/W	DW_apb_i2c generate the interrupt to indicate SCL stuck at low (SCL_STUCK_AT_LOW) if it detects the SCL stuck at low for the IC_SCL_STUCK_LOW_TIMEOUT in units of ic_clk period. This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. **Value After Reset:** IC_SCL_STUCK_TIMEOUT_DEFAULT **Exists:** Always

5.1.49 IC_SDA_STUCK_AT_LOW_TIMEOUT

- **Name:** I2C SDA Stuck at Low Timeout register
- **Description:** I2C SDA Stuck at Low Timeout

  This register is used to store the duration, measured in ic_clk cycles, used to Recover the Data (SDA) line through sending SCL pulses if SDA is held low for the mentioned duration.
- **Size:** 32 bits
- **Offset:** 0xb0
- **Exists:** IC_BUS_CLEAR_FEATURE==1

Table 5-54 Fields for Register: IC_SDA_STUCK_AT_LOW_TIMEOUT

Bits	Name	Memory Access	Description
31:0	IC_SDA_STUCK_LOW_TIMEOUT	R/W	DW_apb_i2c initiates the recovery of SDA line through enabling the SDA_STUCK_RECOVERY_EN (IC_ENABLE[3]) register bit, if it detects the SDA stuck at low for the IC_SDA_STUCK_LOW_TIMEOUT in units of ic_clk period. **Value After Reset:** IC_SDA_STUCK_TIMEOUT_DEFAULT **Exists:** Always

5.1.50 IC_CLR_SCL_STUCK_DET

- **Name:** Clear SCL Stuck at Low Detect interrupt Register
- **Description:** Clear SCL Stuck at Low Detect Interrupt Register
- **Size:** 32 bits
- **Offset:** 0xb4
- **Exists:** IC_BUS_CLEAR_FEATURE==1

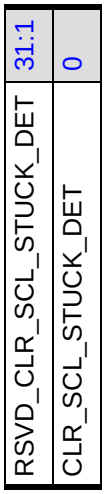


Table 5-55 Fields for Register: IC_CLR_SCL_STUCK_DET

Bits	Name	Memory Access	Description
31:1	RSVD_CLR_SCL_STUCK_DET	R	CLR_SCL_STUCK_DET Reserved bits - Read Only **Exists:** Always **Volatile:** true
0	CLR_SCL_STUCK_DET	R	Read this register to clear the SCL_STUCT_AT_LOW interrupt (bit 15) of the IC_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.51 IC_DEVICE_ID

- **Name:** I2C Device-Id register
- **Description:** I2C Device-ID Register

  This Register contains the Device-ID of the component which includes 12-bits of Manufacturer name and 9-bits of part identification and 3 bits of die-version.
- **Size:** 32 bits
- **Offset:** 0xb8
- **Exists:** IC_DEVICE_ID==1

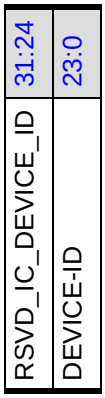


Table 5-56 Fields for Register: IC_DEVICE_ID

Bits	Name	Memory Access	Description
31:24	RSVD_IC_DEVICE_ID	R	IC_DEVICE_ID Reserved bits - Read Only **Exists:** Always
23:0	DEVICE-ID	R	Contains the Device-ID of the component assigned through the configuration parameter 'IC_DEVICE_ID_VALUE' **Value After Reset:** IC_DEVICE_ID_VALUE **Exists:** Always

5.1.52 IC_SMBUS_CLK_LOW_SEXT

- **Name:** SMBus Slave Clock Extend Timeout register
- **Description:** SMBus Slave Clock Extend Timeout Register

  This Register contains the Timeout value used to determine the Slave Clock Extend Timeout in one transfer (from START to STOP). This Register can be written only when the DW_apb_i2c is disabled, which corresponds to IC_ENABLE[0] being set to 0. This register is present only if configuration parameter IC_SMBUS is set to 1. This register is used to store the duration, measured in ic_clk cycles, used to detect the slave clock extend timeout if slave extends the clock (SCL) for the mentioned duration.
- **Size:** 32 bits
- **Offset:** 0xbc
- **Exists:** IC_SMBUS==1

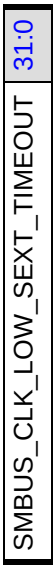


Table 5-57 Fields for Register: IC_SMBUS_CLK_LOW_SEXT

Bits	Name	Memory Access	Description
31:0	SMBUS_CLK_LOW_SEXT_TIMEOUT	R/W	This field is used to detect the Slave Clock Extend timeout (tLOW:SEXT) in master mode extended by the slave device in one message from the initial START to the STOP. The values in this register are in units of ic_clk period. **Value After Reset:** IC_SMBUS_CLK_LOW_SEXT_DEFAULT **Exists:** Always

5.1.53 IC_SMBUS_CLK_LOW_MEXT

- **Name:** SMBus Master Clock Extend Timeout register
- **Description:** SMBus Master Clock Extend Timeout Register

  This Register contains the Timeout value used to determine the Master Clock Extend Timeout in one byte of transfer. This Register can be written only when the DW_apb_i2c is disabled, which corresponds to IC_ENABLE[0] being set to 0. This register is present only if configuration parameter IC_SMBUS is set to 1. This register is used to store the duration, measured in ic_clk cycles, used to detect the Master clock extend timeout if Master extends the clock (SCL) for the mentioned duration.
- **Size:** 32 bits
- **Offset:** 0xc0
- **Exists:** IC_SMBUS==1

Table 5-58 Fields for Register: IC_SMBUS_CLK_LOW_MEXT

Bits	Name	Memory Access	Description
31:0	SMBUS_CLK_LOW_MEXT_TIMEOUT	R/W	This field is used to detect the Master extend SMBus clock (SCLK) timeout defined from START-to-ACK, ACK-to-ACK, or ACK-to-STOP in Master mode. The values in this register are in units of ic_clk period. **Value After Reset:** IC_SMBUS_CLK_LOW_MEXT_DEFAULT **Exists:** Always

5.1.54 IC_SMBUS_THIGH_MAX_IDLE_COUNT

- **Name:** SMBus Master THigh MAX Bus-idle count Register
- **Description:** SMBus Master THigh MAX Bus-idle count Register

  This register programs the Bus-idle time period used when a master has been dynamically added to the bus or when a master has generated a clock reset on the bus. This register is used to store the duration, measured in ic_clk cycles, used to detect the Bus Idle condition if SCL and SDA are held high for the mentioned duration. This register can be written only when the DW_apb_i2c is disabled, which corresponds to IC_ENABLE[0] being set to 0. This register is present only if configuration parameter IC_SMBUS is set to 1.
- **Size:** 32 bits
- **Offset:** 0xc4
- **Exists:** IC_SMBUS==1

Table 5-59 Fields for Register: IC_SMBUS_THIGH_MAX_IDLE_COUNT

Bits	Name	Memory Access	Description
31:16	RSVD_SMBUS_THIGH_MAX_BUS_IDLE_CNT	R	SMBUS_THIGH_MAX_BUS_IDLE_CNT Reserved bits - Read Only **Exists:** Always

Table 5-59 Fields for Register: IC_SMBUS_THIGH_MAX_IDLE_COUNT (Continued)

Bits	Name	Memory Access	Description
15:0	SMBUS_THIGH_MAX_BUS_IDLE_CNT	R/W	This field is used to set the required Bus-Idle time period used when a master has been dynamically added to the bus and may not have detected a state transition on the SMBCLK or SMBDAT lines. In this case, the master must wait long enough to ensure that a transfer is not currently in progress The values in this register are in units of ic_clk period. **Value After Reset:** IC_SMBUS_RST_IDLE_CNT_DEFAULT **Exists:** Always

5.1.55 IC_SMBUS_INTR_STAT

- **Name:** SMBus Interrupt Status Register
- **Description:** SMBUS Interrupt Status Register

  Each bit in this register has a corresponding mask bit in the IC_SMBUS_INTR_MASK register. These bits are cleared by writing the matching SMBus interrupt clear register(IC_CLR_SMBUS_INTR) bits. The unmasked raw versions of these bits are available in the IC_SMBUS_RAW_INTR_STAT register.
- **Size:** 32 bits
- **Offset:** 0xc8
- **Exists:** IC_SMBUS==1

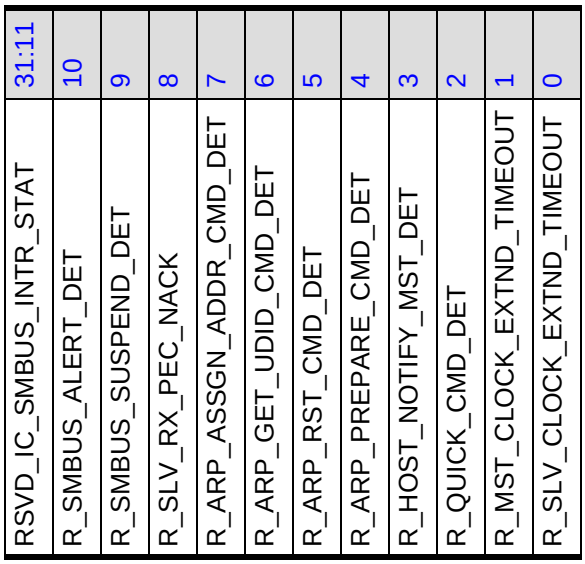


Table 5-60 Fields for Register: IC_SMBUS_INTR_STAT

Bits	Name	Memory Access	Description
31:11	RSVD_IC_SMBUS_INTR_STAT	R	IC_SMBUS_INTR_STAT Reserved bits - Read Only **Exists:** Always **Volatile:** true

Table 5-60 Fields for Register: IC_SMBUS_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
10	R_SMBUS_ALERT_DET	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_SMBUS_ALERT_DET bit. **Values:** ■ 0x1 (ACTIVE): SMBUS_ALERT_DET interrupt is active ■ 0x0 (INACTIVE): SMBUS_ALERT_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1 **Volatile:** true
9	R_SMBUS_SUSPEND_DET	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_SMBUS_SUSPEND_DET bit. **Values:** ■ 0x1 (ACTIVE): SMBUS_SUSPEND_DET interrupt is active ■ 0x0 (INACTIVE): SMBUS_SUSPEND_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1 **Volatile:** true
8	R_SLV_RX_PEC_NACK	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_SLV_RX_PEC_NACK bit. **Values:** ■ 0x1 (ACTIVE): SLV_RX_PEC_NACK interrupt is active ■ 0x0 (INACTIVE): SLV_RX_PEC_NACK interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
7	R_ARP_ASSGN_ADDR_CMD_DET	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_ARP_ASSGN_ADDR_CMD_DET bit. **Values:** ■ 0x1 (ACTIVE): ARP_ASSGN_ADDR_CMD_DET interrupt is active ■ 0x0 (INACTIVE): ARP_ASSGN_ADDR_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true

Table 5-60 Fields for Register: IC_SMBUS_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
6	R_ARP_GET_UDID_CMD_DET	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_ARP_GET_UDID_CMD_DET bit. **Values:** ■ 0x1 (ACTIVE): ARP_GET_UDID_CMD_DET interrupt is active ■ 0x0 (INACTIVE): ARP_GET_UDID_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
5	R_ARP_RST_CMD_DET	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_ARP_RST_CMD_DET bit. **Values:** ■ 0x1 (ACTIVE): ARP_RST_CMD_DET interrupt is active ■ 0x0 (INACTIVE): ARP_RST_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
4	R_ARP_PREPARE_CMD_DET	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_ARP_PREPARE_CMD_DET bit. **Values:** ■ 0x1 (ACTIVE): ARP_PREPARE_CMD_DET interrupt is active ■ 0x0 (INACTIVE): ARP_PREPARE_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
3	R_HOST_NOTIFY_MST_DET	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_HOST_NOTIFY_MST_DET bit. **Values:** ■ 0x1 (ACTIVE): HOST_NOTIFY_MST_DET interrupt is active ■ 0x0 (INACTIVE): HOST_NOTIFY_MST_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS==1 **Volatile:** true

Table 5-60 Fields for Register: IC_SMBUS_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
2	R_QUICK_CMD_DET	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_QUICK_CMD_DET bit. **Values:** ■ 0x1 (ACTIVE): QUICK_CMD_DET interrupt is active ■ 0x0 (INACTIVE): QUICK_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
1	R_MST_CLOCK_EXTND_TIMEOUT	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_MST_CLOCK_EXTND_TIMEOUT bit. **Values:** ■ 0x1 (ACTIVE): MST_CLOCK_EXTND_TIMEOUT interrupt is active ■ 0x0 (INACTIVE): MST_CLOCK_EXTND_TIMEOUT interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	R_SLV_CLOCK_EXTND_TIMEOUT	R	See IC_SMBUS_INTR_RAW_STATUS for a detailed description of R_SLV_CLOCK_EXTND_TIMEOUT bit. **Values:** ■ 0x1 (ACTIVE): SLV_CLOCK_EXTND_TIMEOUT interrupt is active ■ 0x0 (INACTIVE): SLV_CLOCK_EXTND_TIMEOUT interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.56 IC_SMBUS_INTR_MASK

- **Name:** SMBus Interrupt Mask Register
- **Description:** SMBus Interrupt Mask Register
- **Size:** 32 bits
- **Offset:** 0xcc
- **Exists:** IC_SMBUS==1

31:11	10	9	8	7	6	5	4	3	2	1	0
RSVD_IC_SMBUS_INTR_MASK	M_SMBUS_ALERT_DET	R_SMBUS_SUSPEND_DET	M_SLV_RX_PEC_NACK	M_ARP_ASSGN_ADDR_CMD_DET	M_ARP_GET_UDID_CMD_DET	M_ARP_RST_CMD_DET	M_ARP_PREPARE_CMD_DET	M_HOST_NOTIFY_MST_DET	M_QUICK_CMD_DET	M_MST_CLOCK_EXTND_TIMEOUT	M_SLV_CLOCK_EXTND_TIMEOUT

Table 5-61 Fields for Register: IC_SMBUS_INTR_MASK

Bits	Name	Memory Access	Description
31:11	RSVD_IC_SMBUS_INTR_MASK	R	IC_SMBUS_INTR_MASK Reserved bits - Read Only **Exists:** Always
10	M_SMBUS_ALERT_DET	R/W	This bit masks the R_SMBUS_ALERT_DET interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): SMBUS_ALERT_DET interrupt is unmasked ■ 0x0 (ENABLED): SMBUS_ALERT_DET interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_SMBUS_SUSPEND_ALERT==1

Table 5-61 Fields for Register: IC_SMBUS_INTR_MASK (Continued)

Bits	Name	Memory Access	Description
9	R_SMBUS_SUSPEND_DET	R/W	This bit masks the R_SMBUS_SUSPEND_DET interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): SMBUS_SUSPEND_DET interrupt is unmasked ■ 0x0 (ENABLED): SMBUS_SUSPEND_DET interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_SMBUS_SUSPEND_ALERT==1
8	M_SLV_RX_PEC_NACK	R/W	This bit masks the R_SLV_RX_PEC_NACK interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): SLV_RX_PEC_NACK interrupt is unmasked ■ 0x0 (ENABLED): SLV_RX_PEC_NACK interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_SMBUS_ARP==1
7	M_ARP_ASSGN_ADDR_CMD_DET	R/W	This bit masks the R_ARP_ASSGN_ADDR_CMD_DET interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): ARP_ASSGN_ADDR_CMD_DET interrupt is unmasked ■ 0x0 (ENABLED): ARP_ASSGN_ADDR_CMD_DET interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_SMBUS_ARP==1
6	M_ARP_GET_UDID_CMD_DET	R/W	This bit masks the R_ARP_GET_UDID_CMD_DET interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): ARP_GET_UDID_CMD_DET interrupt is unmasked ■ 0x0 (ENABLED): ARP_GET_UDID_CMD_DET interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_SMBUS_ARP==1

Table 5-61 Fields for Register: IC_SMBUS_INTR_MASK (Continued)

Bits	Name	Memory Access	Description
5	M_ARP_RST_CMD_DET	R/W	This bit masks the R_ARP_RST_CMD_DET interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): ARP_RST_CMD_DET interrupt is unmasked ■ 0x0 (ENABLED): ARP_RST_CMD_DET interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_SMBUS_ARP==1
4	M_ARP_PREPARE_CMD_DET	R/W	This bit masks the R_ARP_PREPARE_CMD_DET interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): ARP_PREPARE_CMD_DET interrupt is unmasked ■ 0x0 (ENABLED): ARP_PREPARE_CMD_DET interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_SMBUS_ARP==1
3	M_HOST_NOTIFY_MST_DET	R/W	This bit masks the R_HOST_NOTIFY_MST_DET interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): HOST_NOTIFY_MST_DET interrupt is unmasked ■ 0x0 (ENABLED): HOST_NOTIFY_MST_DET interrupt is masked **Value After Reset:** 0x1 **Exists:** IC_SMBUS==1
2	M_QUICK_CMD_DET	R/W	This bit masks the R_QUICK_CMD_DET interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): QUICK_CMD_DET interrupt is unmasked ■ 0x0 (ENABLED): QUICK_CMD_DET interrupt is masked **Value After Reset:** 0x1 **Exists:** Always

Table 5-61 Fields for Register: IC_SMBUS_INTR_MASK (Continued)

Bits	Name	Memory Access	Description
1	M_MST_CLOCK_EXTND_TIMEOUT	R/W	This bit masks the R_MST_CLOCK_EXTND_TIMEOUT interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): MST_CLOCK_EXTND_TIMEOUT interrupt is unmasked ■ 0x0 (ENABLED): MST_CLOCK_EXTND_TIMEOUT interrupt is masked **Value After Reset:** 0x1 **Exists:** Always
0	M_SLV_CLOCK_EXTND_TIMEOUT	R/W	This bit masks the R_SLV_CLOCK_EXTND_TIMEOUT interrupt in IC_SMBUS_INTR_STAT register. **Values:** ■ 0x1 (DISABLED): SLV_CLOCK_EXTND_TIMEOUT interrupt is unmasked ■ 0x0 (ENABLED): SLV_CLOCK_EXTND_TIMEOUT interrupt is masked **Value After Reset:** 0x1 **Exists:** Always

5.1.57 IC_SMBUS_RAW_INTR_STAT

- **Name:** SMBus Raw Interrupt Status Register
- **Description:** SMBus Raw Interrupt Status Register

  Unlike the IC_SMBUS_INTR_STAT register, these bits are not masked so they always show the true status of the DW_apb_i2c.
- **Size:** 32 bits
- **Offset:** 0xd0
- **Exists:** IC_SMBUS==1

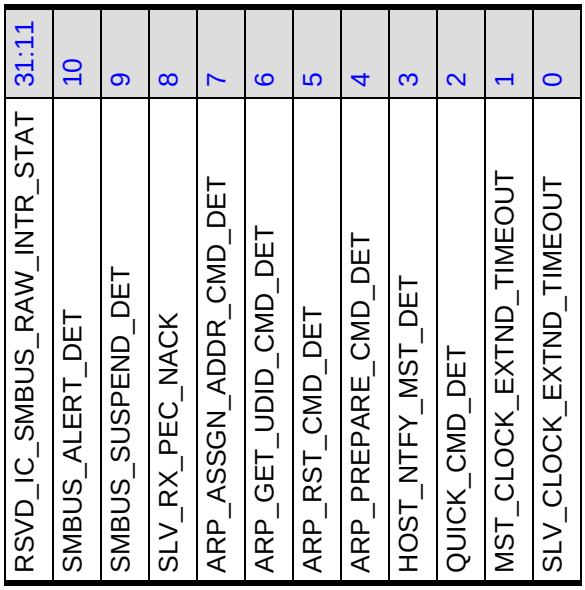


Table 5-62 Fields for Register: IC_SMBUS_RAW_INTR_STAT

Bits	Name	Memory Access	Description
31:11	RSVD_IC_SMBUS_RAW_INTR_STAT	R	IC_SMBUS_RAW_INTR_STAT Reserved bits - Read Only **Exists:** Always **Volatile:** true

Table 5-62 Fields for Register: IC_SMBUS_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
10	SMBUS_ALERT_DET	R	Indicates whether a SMBALERT (ic_smbalert_in_n) signal is driven low by the slave. **Values:** ■ 0x1 (ACTIVE): SMBUS Alert interrupt is active ■ 0x0 (INACTIVE): SMBUS Alert interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1 **Volatile:** true
9	SMBUS_SUSPEND_DET	R	Indicates whether a SMBSUS (ic_smbsus_in_n) signal is driven low by the Host. **Values:** ■ 0x1 (ACTIVE): SMBUS System Suspended interrupt is active ■ 0x0 (INACTIVE): SMBUS System Suspended interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1 **Volatile:** true
8	SLV_RX_PEC_NACK	R	Indicates whether a NACK has been sent due to PEC mismatch while working as ARP slave. **Values:** ■ 0x1 (ACTIVE): SLV_RX_PEC_NACK interrupt is active ■ 0x0 (INACTIVE): SLV_RX_PEC_NACK interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
7	ARP_ASSGN_ADDR_CMD_DET	R	Indicates whether an Assign Address ARP command has been received. **Values:** ■ 0x1 (ACTIVE): ARP_ASSGN_ADDR_CMD_DET interrupt is active ■ 0x0 (INACTIVE): ARP_ASSGN_ADDR_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true

Table 5-62 Fields for Register: IC_SMBUS_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
6	ARP_GET_UDID_CMD_DET	R	Indicates whether a Get UDID ARP command has been received. **Values:** ■ 0x1 (ACTIVE): ARP_GET_UDID_CMD_DET interrupt is active ■ 0x0 (INACTIVE): ARP_GET_UDID_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
5	ARP_RST_CMD_DET	R	Indicates whether a General or Directed Reset ARP command has been received. **Values:** ■ 0x1 (ACTIVE): ARP_RST_CMD_DET interrupt is active ■ 0x0 (INACTIVE): ARP_RST_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
4	ARP_PREPARE_CMD_DET	R	Indicates whether a prepare to ARP command has been received. **Values:** ■ 0x1 (ACTIVE): ARP_PREPARE_CMD_DET interrupt is active ■ 0x0 (INACTIVE): ARP_PREPARE_CMD_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1 **Volatile:** true
3	HOST_NTFY_MST_DET	R	Indicates whether a Notify ARP Master ARP command has been received. **Values:** ■ 0x1 (ACTIVE): HOST_NTFY_MST_DET interrupt is active ■ 0x0 (INACTIVE): HOST_NTFY_MST_DET interrupt is inactive **Value After Reset:** 0x0 **Exists:** IC_SMBUS==1 **Volatile:** true

Table 5-62 Fields for Register: IC_SMBUS_RAW_INTR_STAT (Continued)

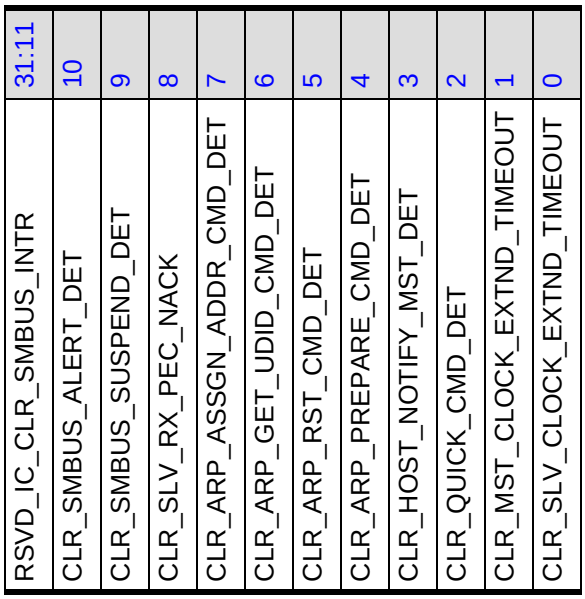
Bits	Name	Memory Access	Description
2	QUICK_CMD_DET	R	Indicates whether a Quick command has been received on the SMBus interface regardless of whether DW_apb_i2c is operating in slave or master mode. Enabled only when IC_SMBUS=1 is set to 1. **Values:** ■ 0x1 (ACTIVE): Quick Command interrupt is active ■ 0x0 (INACTIVE): Quick Command interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
1	MST_CLOCK_EXTND_TIMEOUT	R	Indicates whether the Master device transaction (START-to-ACK, ACK-to-ACK, or ACK-to-STOP) from START to STOP exceeds IC_SMBUS_CLOCK_LOW_MEXT time with in each byte of message. This bit is enabled only when: ■ IC_SMBUS=1 ■ IC_CON[0]=1 ■ IC_EMPTYFIFO_HOLD_MASTER_EN=1 or ■ IC_RX_FULL_HLD_BUS_EN=1 **Values:** ■ 0x1 (ACTIVE): Master Clock Extend Timeout interrupt is active ■ 0x0 (INACTIVE): Master Clock Extend Timeout interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-62 Fields for Register: IC_SMBUS_RAW_INTR_STAT (Continued)

Bits	Name	Memory Access	Description
0	SLV_CLOCK_EXTND_TIMEOUT	R	Indicates whether the transaction from Slave (i.e from START to STOP) exceeds IC_SMBUS_CLK_LOW_SEXT time. This bit is enabled only when: ■ IC_SMBUS=1 ■ IC_CON[0]=1 **Values:** ■ 0x1 (ACTIVE): Slave Clock Extend Timeout interrupt is active ■ 0x0 (INACTIVE): Slave Clock Extend Timeout interrupt is inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.58 IC_CLR_SMBUS_INTR

- **Name:** Clear SMBus Interrupt Register
- **Description:** SMBus Clear Interrupt Register
- **Size:** 32 bits
- **Offset:** 0xd4
- **Exists:** IC_SMBUS==1



31:11	10	9	8	7	6	5	4	3	2	1	0
RSVD_IC_CLR_SMBUS_INTR	CLR_SMBUS_ALERT_DET	CLR_SMBUS_SUSPEND_DET	CLR_SLV_RX_PEC_NACK	CLR_ARP_ASSGN_ADDR_CMD_DET	CLR_ARP_GET_UDID_CMD_DET	CLR_ARP_RST_CMD_DET	CLR_ARP_PREPARE_CMD_DET	CLR_HOST_NOTIFY_MST_DET	CLR_QUICK_CMD_DET	CLR_MST_CLOCK_EXTND_TIMEOUT	CLR_SLV_CLOCK_EXTND_TIMEOUT

Table 5-63 Fields for Register: IC_CLR_SMBUS_INTR

Bits	Name	Memory Access	Description
31:11	RSVD_IC_CLR_SMBUS_INTR	W	IC_CLR_SMBUS_INTR Reserved bits - Read Only **Exists:** Always
10	CLR_SMBUS_ALERT_DET	W	Write this register bit to clear the SMBUS_ALERT_DET interrupt (bit 10) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1
9	CLR_SMBUS_SUSPEND_DET	W	Write this register bit to clear the SMBUS_SUSPEND_DET interrupt (bit 9) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_SUSPEND_ALERT==1

Table 5-63 Fields for Register: IC_CLR_SMBUS_INTR (Continued)

Bits	Name	Memory Access	Description
8	CLR_SLV_RX_PEC_NACK	W	Write this register bit to clear the SLV_RX_PEC_NACK interrupt (bit 8) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1
7	CLR_ARP_ASSGN_ADDR_CMD_DET	W	Write this register bit to clear the ARP_ASSGN_ADDR_CMD_DET interrupt (bit 7) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1
6	CLR_ARP_GET_UDID_CMD_DET	W	Write this register bit to clear the ARP_GET_UDID_CMD_DET interrupt (bit 6) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1
5	CLR_ARP_RST_CMD_DET	W	Write this register bit to clear the ARP_RST_CMD_DET interrupt (bit 5) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1
4	CLR_ARP_PREPARE_CMD_DET	W	Write this register bit to clear the ARP_PREPARE_CMD_DET interrupt (bit 4) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** IC_SMBUS_ARP==1
3	CLR_HOST_NOTIFY_MST_DET	W	Write this register bit to clear the HOST_NOTIFY_MST_DET interrupt (bit 3) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** IC_SMBUS==1
2	CLR_QUICK_CMD_DET	W	Write this register bit to clear the QUICK_CMD_DET interrupt (bit 2) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always
1	CLR_MST_CLOCK_EXTND_TIMEOUT	W	Write this register bit to clear the MST_CLOCK_EXTND_TIMEOUT interrupt (bit 1) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always

Table 5-63 Fields for Register: IC_CLR_SMBUS_INTR (Continued)

Bits	Name	Memory Access	Description
0	CLR_SLV_CLOCK_EXTND_TIMEOUT	W	Write this register bit to clear the SLV_CLOCK_EXTND_TIMEOUT interrupt (bit 0) of the IC_SMBUS_RAW_INTR_STAT register. **Value After Reset:** 0x0 **Exists:** Always

5.1.59 IC_OPTIONAL_SAR

- **Name:** I2C Optional Slave Address Register
- **Description:** I2C Optional Slave Address Register

  Optional Slave address for I2C in SMBus Mode. A same restriction as IC_SAR applies on IC_OPTIONAL_SAR.
- **Size:** 32 bits
- **Offset:** 0xd8
- **Exists:** IC_OPTIONAL_SAR==1

Table 5-64 Fields for Register: IC_OPTIONAL_SAR

Bits	Name	Memory Access	Description
31:7	RSVD_IC_OPTIONAL_SAR	R	IC_OPTIONAL_SAR Reserved bits - Read Only **Exists:** Always
6:0	OPTIONAL_SAR	R/W	Optional Slave address for DW_apb_i2c when operating as a slave in SMBus Mode. **Value After Reset:** IC_OPTIONAL_SAR_DEFAULT **Exists:** Always

5.1.60 IC_SMBUS_UDID_LSB

- **Name:** SMBUS ARP UDID LSB Register
- **Description:** SMBUS ARP UDID LSB Register

  This Register can be written only when the DW_apb_i2c is disabled, which corresponds to IC_ENABLE[0] being set to 0. This register is present only if configuration parameter IC_SMBUS_UDID_HC is set to 1. This register is used to store the LSB 32 bit value of Slave UDID register used in Address Resolution Protocol of SMBus.
- **Size:** 32 bits
- **Offset:** 0xdc
- **Exists:** (IC_SMBUS_ARP == 1) && (IC_SMBUS_UDID_HC == 1)

Table 5-65 Fields for Register: IC_SMBUS_UDID_LSB

Bits	Name	Memory Access	Description
31:0	SMBUS_UDID_LSB	R/W	This field is used to store the LSB 32 bit value of slave unique device identifier used in Address Resolution Protocol. **Value After Reset:** IC_SMBUS_UDID_LSB_DEFAULT **Exists:** Always

5.1.61 IC_SMBUS_UDID_WORD0

- **Name:** SMBUS ARP UDID WORD0 Register
- **Description:** SMBUS UDID WORD0 Register

  This Register can be written only when the DW_apb_i2c is disabled, which corresponds to IC_ENABLE[0] being set to 0. This register is present only if configuration parameter IC_SMBUS_UDID_HC is set to 0. This register is used to store the Lower 32 bit value of Slave UDID register i.e. UDID[31:0] used in Address Resolution Protocol of SMBus.
- **Size:** 32 bits
- **Offset:** 0xdc
- **Exists:** IC_SMBUS_UDID_HC==0

Table 5-66 Fields for Register: IC_SMBUS_UDID_WORD0

Bits	Name	Memory Access	Description
31:0	SMBUS_UDID_WORD0	R/W	This field is used to store the Lower 32 bit value of slave unique device identifier used in Address Resolution Protocol. **Value After Reset:** IC_SMBUS_UDID_LSB_DEFAULT **Exists:** Always

5.1.62 IC_SMBUS_UDID_WORD1

- **Name:** SMBUS ARP UDID WORD1 Register
- **Description:** SMBUS UDID WORD1 Register

  This Register can be written only when the DW_apb_i2c is disabled, which corresponds to IC_ENABLE[0] being set to 0. This register is present only if configuration parameter IC_SMBUS_UDID_HC is set to 0. This register is used to store the Middle-Lower 32 bit value of Slave UDID register i.e. UDID[63:32] used in Address Resolution Protocol of SMBus.
- **Size:** 32 bits
- **Offset:** 0xe0
- **Exists:** IC_SMBUS_UDID_HC==0

Table 5-67 Fields for Register: IC_SMBUS_UDID_WORD1

Bits	Name	Memory Access	Description
31:0	SMBUS_UDID_WORD1	R/W	This field is used to store the Middle-Lower 32 bit value of slave unique device identifier used in Address Resolution Protocol. **Value After Reset:** IC_SMBUS_UDID_WORD1_DEFAULT **Exists:** Always

5.1.63 IC_SMBUS_UDID_WORD2

- **Name:** SMBUS ARP UDID WORD2 Register
- **Description:** SMBUS UDID WORD2 Register

  This Register can be written only when the DW_apb_i2c is disabled, which corresponds to IC_ENABLE[0] being set to 0. This register is present only if configuration parameter IC_SMBUS_UDID_HC is set to 0. This register is used to store the Middle-Upper 32 bit value of Slave UDID register i.e. UDID[95:64] used in Address Resolution Protocol of SMBus.
- **Size:** 32 bits
- **Offset:** 0xe4
- **Exists:** IC_SMBUS_UDID_HC==0

Table 5-68 Fields for Register: IC_SMBUS_UDID_WORD2

Bits	Name	Memory Access	Description
31:0	SMBUS_UDID_WORD2	R/W	This field is used to store the Middle-Upper 32 bit value of slave unique device identifier used in Address Resolution Protocol. **Value After Reset:** IC_SMBUS_UDID_WORD2_DEFAULT **Exists:** Always

5.1.64 IC_SMBUS_UDID_WORD3

- **Name:** SMBUS ARP UDID WORD3 Register
- **Description:** SMBUS UDID WORD3 Register

  This Register can be written only when the DW_apb_i2c is disabled, which corresponds to IC_ENABLE[0] being set to 0. This register is present only if configuration parameter IC_SMBUS_UDID_HC is set to 0. This register is used to store the Upper 32 bit value of Slave UDID register i.e. UDID[127:96] used in Address Resolution Protocol of SMBus.
- **Size:** 32 bits
- **Offset:** 0xe8
- **Exists:** IC_SMBUS_UDID_HC==0

Table 5-69 Fields for Register: IC_SMBUS_UDID_WORD3

Bits	Name	Memory Access	Description
31:0	SMBUS_UDID_WORD3	R/W	This field is used to store the Upper 32 bit value of slave unique device identifier used in Address Resolution Protocol. **Value After Reset:** IC_SMBUS_UDID_WORD3_DEFAULT **Exists:** Always

5.1.65 REG_TIMEOUT_RST

- **Name:** Register timeout counter reset value
- **Description:** Name: Register timeout counter reset register Size: REG_TIMEOUT_WIDTH bits Address: 0xF0 Read / Write Access: Read / Write This register keeps the timeout value of register timer counter. The reset value of the register is REG_TIMEOUT_VALUE. The default reset value can be further modified if HC_REG_TIMEOUT_VALUE = 0. The final programmed value (or the default reset value if not programmed) determines from what value the register timeout counter starts counting down. A zero on this counter will break the waited transaction with PSLVERR as high.
- **Size:** 32 bits
- **Offset:** 0xf0
- **Exists:** [<functionof> "(((SLAVE_INTERFACE_TYPE>0 && SLVERR_RESP_EN==1 && REG_TIMEOUT_WIDTH>0) ? 1 : 0)==1) ? 1 : 0"]

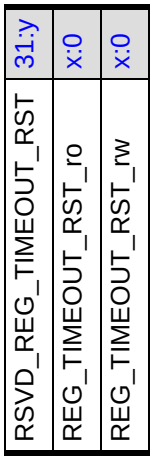


Table 5-70 Fields for Register: REG_TIMEOUT_RST

Bits	Name	Memory Access	Description
31:y	RSVD_REG_TIMEOUT_RST	R	Reserved bits - Read Only **Exists:** Always **Volatile:** true **Range Variable[y]:** REG_TIMEOUT_WIDTH
x:0	REG_TIMEOUT_RST_ro	R	This field holds reset value of REG_TIMEOUT counter register. **Value After Reset:** REG_TIMEOUT_VALUE **Exists:** [<functionof> "(HC_REG_TIMEOUT_VALUE==1) ? 1 : 0"] **Volatile:** true **Range Variable[x]:** REG_TIMEOUT_WIDTH - 1

Table 5-70 Fields for Register: REG_TIMEOUT_RST (Continued)

Bits	Name	Memory Access	Description
x:0	REG_TIMEOUT_RST_rw	R/W	This field holds reset value of REG_TIMEOUT counter register. **Value After Reset:** REG_TIMEOUT_VALUE **Exists:** [<functionof> "(HC_REG_TIMEOUT_VALUE==0) ? 1 : 0"] **Volatile:** true **Range Variable[x]:** REG_TIMEOUT_WIDTH - 1

5.1.66 IC_COMP_PARAM_1

- **Name:** Component Parameter Register 1
- **Description:** Component Parameter Register 1

  NoteThis is a constant read-only register that contains encoded information about the component's parameter settings. The reset value depends on coreConsultant parameter(s).
- **Size:** 32 bits
- **Offset:** 0xf4
- **Exists:** [<functionof> "(IC_ADD_ENCODED_PARAMS==1) ? 1 : 0"]

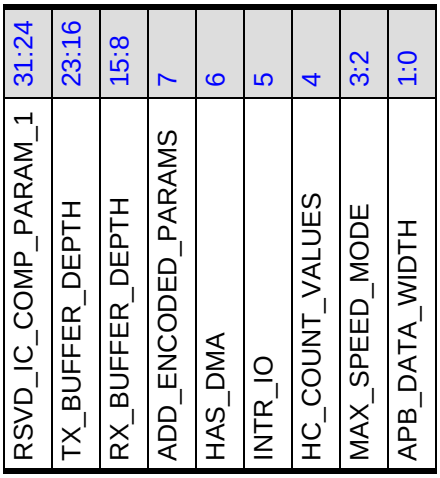


Table 5-71 Fields for Register: IC_COMP_PARAM_1

Bits	Name	Memory Access	Description
31:24	RSVD_IC_COMP_PARAM_1	R	IC_COMP_PARAM_1 Reserved bits - Read Only **Exists:** Always
23:16	TX_BUFFER_DEPTH	R	The value of this register is derived from the IC_TX_BUFFER_DEPTH coreConsultant parameter. ■ 0x00 = Reserved ■ 0x01 = 2 ■ 0x02 = 3 ■ ... ■ 0xFF = 256 **Value After Reset:** ENCODED_IC_TX_BUFFER_DEPTH **Exists:** Always

Table 5-71 Fields for Register: IC_COMP_PARAM_1 (Continued)

Bits	Name	Memory Access	Description
15:8	RX_BUFFER_DEPTH	R	The value of this register is derived from the IC_RX_BUFFER_DEPTH coreConsultant parameter. ■ 0x00: Reserved ■ 0x01: 2 ■ 0x02: 3 ■ ... ■ 0xFF: 256 **Value After Reset:** ENCODED_IC_RX_BUFFER_DEPTH **Exists:** Always
7	ADD_ENCODED_PARAMS	R	The value of this register is derived from the IC_ADD_ENCODED_PARAMS coreConsultant parameter. Reading 1 in this bit means that the capability of reading these encoded parameters via software has been included. Otherwise, the entire register is 0 regardless of the setting of any other parameters that are encoded in the bits. **Values:** ■ 0x1 (ENABLED): Enables capability of reading encoded parameters ■ 0x0 (DISBALED): Disables capability of reading encoded parameters **Value After Reset:** IC_ADD_ENCODED_PARAMS **Exists:** Always
6	HAS_DMA	R	The value of this register is derived from the IC_HAS_DMA coreConsultant parameter. **Values:** ■ 0x1 (ENABLED): DMA handshaking signals are enabled ■ 0x0 (DISABLED): DMA handshaking signals are disabled **Value After Reset:** IC_HAS_DMA **Exists:** Always
5	INTR_IO	R	The value of this register is derived from the IC_INTR_IO coreConsultant parameter. **Values:** ■ 0x1 (COMBINED): COMBINED Interrupt outputs ■ 0x0 (INDIVIDUAL): INDIVIDUAL Interrupt outputs **Value After Reset:** IC_INTR_IO **Exists:** Always

Table 5-71 Fields for Register: IC_COMP_PARAM_1 (Continued)

Bits	Name	Memory Access	Description
4	HC_COUNT_VALUES	R	The value of this register is derived from the IC_HC_COUNT VALUES coreConsultant parameter. **Values:** ■ 0x1 (ENABLED): Hard code the count values for each mode. ■ 0x0 (DISABLED): Programmable count values for each mode. **Value After Reset:** IC_HC_COUNT_VALUES **Exists:** Always
3:2	MAX_SPEED_MODE	R	The value of this register is derived from the IC_MAX_SPEED_MODE coreConsultant parameter. ■ 0x0: Reserved ■ 0x1: Standard ■ 0x2: Fast ■ 0x3: High **Values:** ■ 0x1 (STANDARD): IC MAX SPEED is STANDARD MODE ■ 0x2 (FAST): IC MAX SPEED is FAST MODE ■ 0x3 (HIGH): IC MAX SPEED is HIGH MODE **Value After Reset:** "(IC_ULTRA_FAST_MODE_EN==0) ? (IC_MAX_SPEED_MODE) : \"0x00\"" **Exists:** IC_ULTRA_FAST_MODE==0
1:0	APB_DATA_WIDTH	R	The value of this register is derived from the APB_DATA_WIDTH coreConsultant parameter. **Values:** ■ 0x0 (APB_08BITS): APB data bus width is 08 bits ■ 0x1 (APB_16BITS): APB data bus width is 16 bits ■ 0x2 (APB_32BITS): APB data bus width is 32 bits ■ 0x3 (RESERVED): Reserved bits **Value After Reset:** ENCODED_APB_DATA_WIDTH **Exists:** Always

5.1.67 IC_COMP_VERSION

- **Name:** I2C Component Version Register
- **Description:** I2C Component Version Register
- **Size:** 32 bits
- **Offset:** 0xf8
- **Exists:** Always

Table 5-72 Fields for Register: IC_COMP_VERSION

Bits	Name	Memory Access	Description
31:0	IC_COMP_VERSION	R	Specific values for this register are described in the Releases Table in the DW_apb_i2c Release Notes **Value After Reset:** IC_VERSION_ID **Exists:** Always

5.1.68 IC_COMP_TYPE

- **Name:** I2C Component Type Register
- **Description:** I2C Component Type Register
- **Size:** 32 bits
- **Offset:** 0xfc
- **Exists:** Always

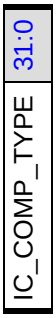


Table 5-73 Fields for Register: IC_COMP_TYPE

Bits	Name	Memory Access	Description
31:0	IC_COMP_TYPE	R	Designware Component Type number = 0x44_57_01_40. This assigned unique hex value is constant and is derived from the two ASCII letters 'DW' followed by a 16-bit unsigned number. **Value After Reset:** 0x44570140 **Exists:** Always

6 Programming the DW_apb_i2c

The DW_apb_i2c can be programmed through software registers or the DW_apb_i2c low-level software driver.

6.1 Software Registers

For information about programming the software registers in terms of DW_apb_i2c operation, see "Slave Mode Operation" on page 44 and "Master Mode Operation" on page 47. The software registers are described in more detail in "Register Descriptions" on page 141.

6.2 Software Drivers

The family of DesignWare Synthesizable Components includes a Driver Kit for the DW_apb_i2c component. This low-level Driver Kit allows you to easily program a DW_apb_i2c component and integrate your code into a larger software system. The Driver Kit provides the following benefits to IP designers:

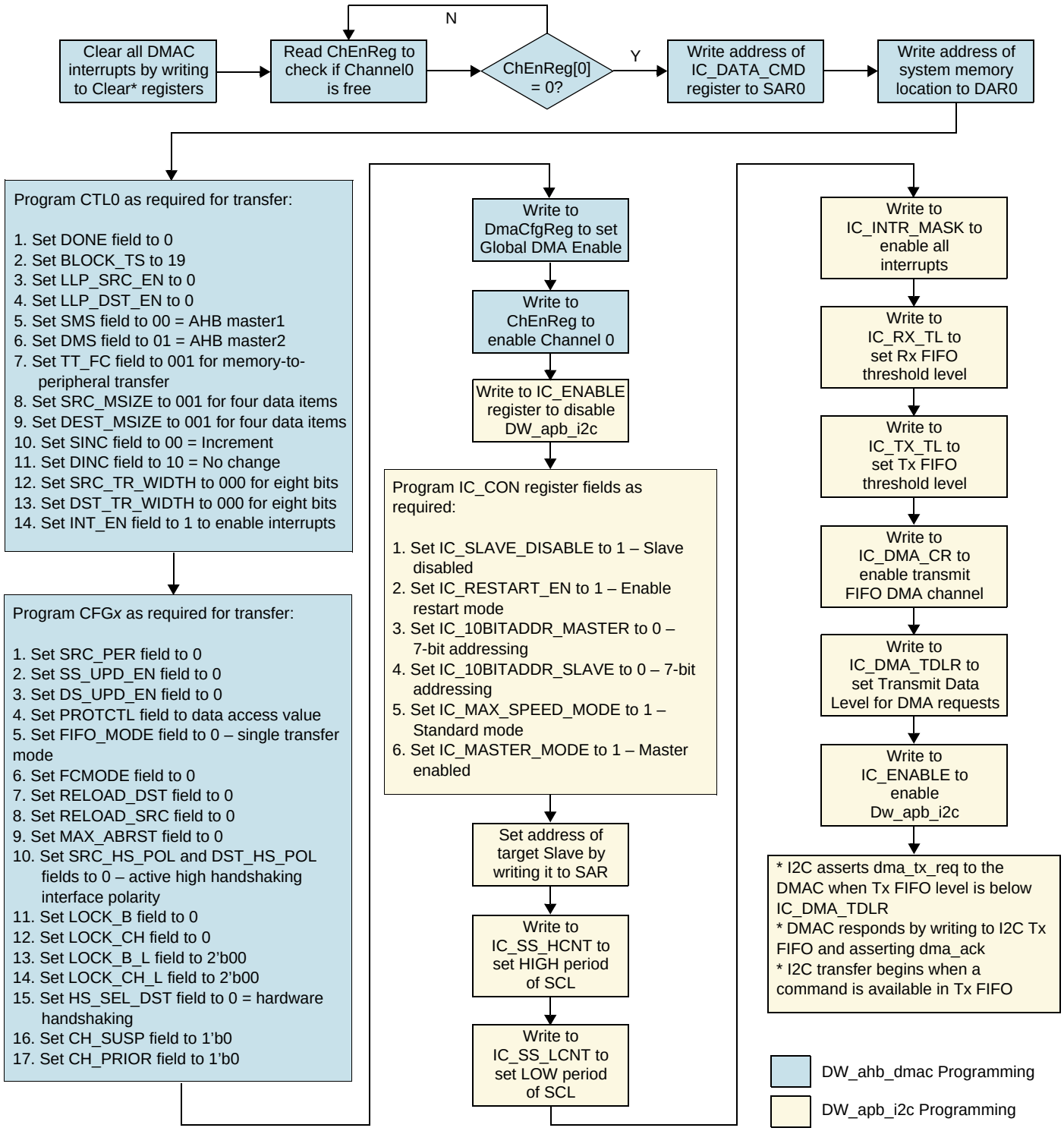
- Proven method of access to DW_apb_i2c minimizing usage errors
- Rapid software development with minimum overhead
- Detailed knowledge of DW_apb_i2c register bit fields not required
- Easy integration of DW_apb_i2c into existing software system
- Programming at register level eliminated

You must purchase a source code license (DWC-APB-Advanced-Source) to use the DW_apb_i2c Driver Kit. However, you can access some Driver Kit files and documentation in $DESIGNWARE_HOME/drivers/DW_apb_i2c/latest. For more information about the Driver Kit, see the *DW_apb_i2c Driver Kit User Guide*. For more information about purchasing the source code license and obtaining a download of the Driver Kit, contact Synopsys at designware@synopsys.com for details.

6.3 Programming Example

The flow diagram in Figure 6-1 shows an overview of programming the DW_apb_i2c.

Figure 6-1 Flowchart for DW_ahb_dmac and DW_apb_i2c Programming Example



Note When there is at least one entry in the DW_apb_i2c Rx FIFO, the DW_apb_i2c asserts dma_single to the DMAC. When the number of entries in the DW_apb_i2c Rx FIFO reaches reaches IC_DMA_RDLR, the DW_apb_i2c asserts dma_rx_req to the DMAC. In this example, in order to read nineteen data items from the DW_apb_i2c Rx FIFO, the DMAC samples dma_req for three BURST transfers of four beats of size 1 byte each, and it samples dma_single for three SINGLE transfers of size 1 byte each.

The following outlines details regarding reads and writes to/from DW_apb_i2c masters/slaves and VIP master/slaves:

- For DW_apb_i2c master writes to a slave VIP model, bear in mind when using the DMA that you are writing characters from the byte stream. However, for a write, the DW_apb_i2c needs a halfword. To use the DMA, you should write software similar to the following:

```
short int temp_array[];
char * ptr=(char *) temp_array;
foreach byte in bytes {
  store byte ptr++;
  store '0x01' write command ptr++
}
```

  a. Program the DMA to read a stream of halfwords from memory and write them to the DW_apb_i2c using the hardware interface.

  b. Program the DW_apb_i2c to do a write using the transmit DMA.

- For DW_apb_i2c master reads from a slave VIP model:

  a. Create a read command halfword.

  b. Program DMA channel 0 to do a fixed read of the read command halfword—that is, no address increment—to the DW_apb_i2c transmit buffer.

  c. Program DMA channel 1 to read the data from the read buffer and store it in memory.

  d. Program the DW_apb_i2c to do a master read by setting *both* DMA channels.

- For DW_apb_i2c slave writes from a master VIP model:

  a. Program the DW_apb_i2c to be a slave with the RX buffer DMA enabled.

  b. Program the DMA to read the buffer and store the bytes in memory.

- For DW_apb_i2c slave reads from a master VIP model:

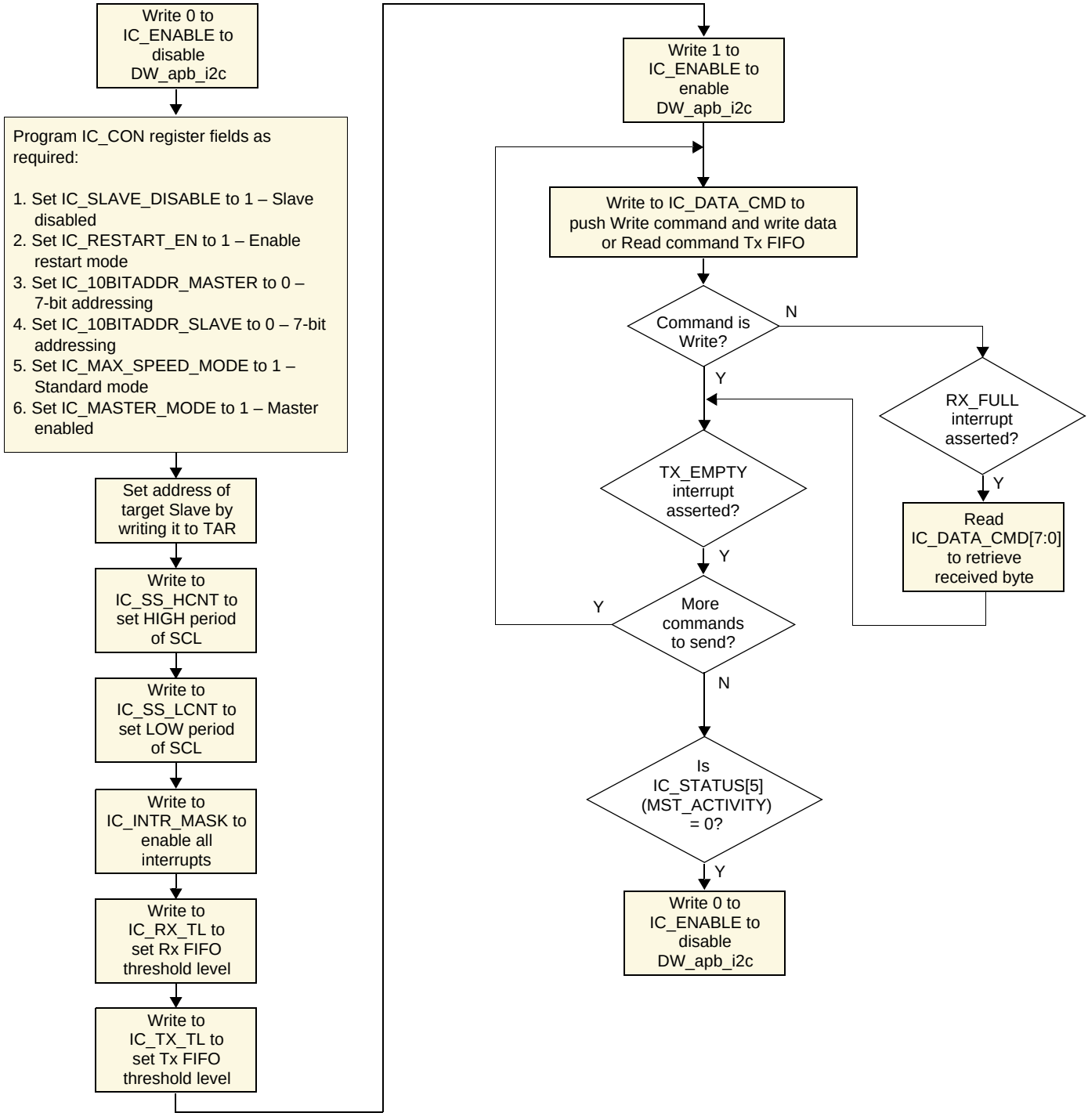
  a. Enable IC_INTR_MASK.RD_REQ; otherwise the DW_apb_i2c does not acknowledge the read.

  b. When you get the RD_REQ interrrupt, program the DMA to write the TX buffer with the read data.

  c. Program the DW_apb_i2c to enable the TX DMA.

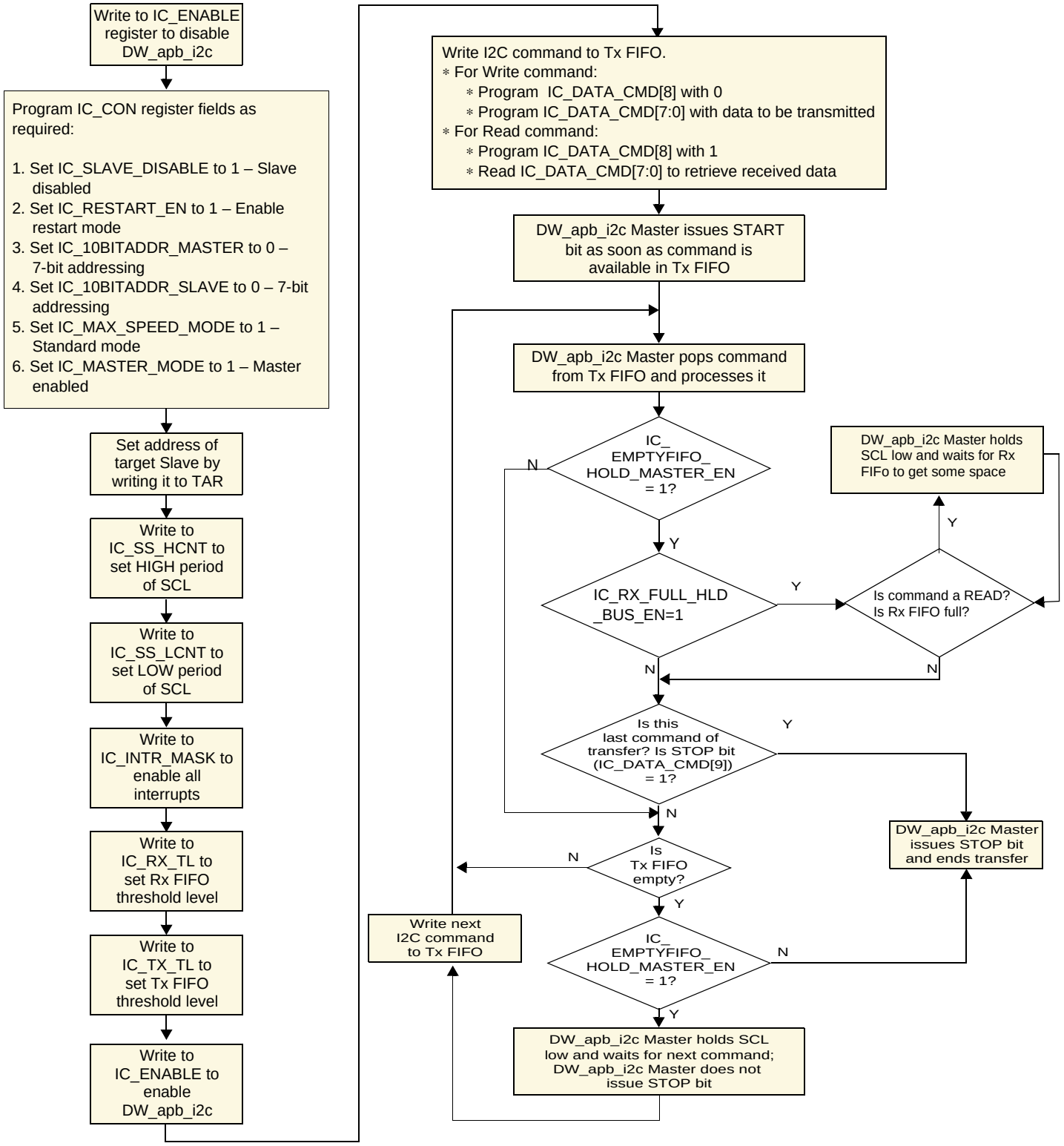
The flow diagram in Figure 6-2 shows a programming example for the DW_apb_i2c Master.

Figure 6-2 Flowchart for DW_apb_i2c Master



The flow diagram in Figure 6-3 shows a programming example for the DW_apb_i2c master in standard mode, fast mode, or fast mode plus with 7-bit addressing.

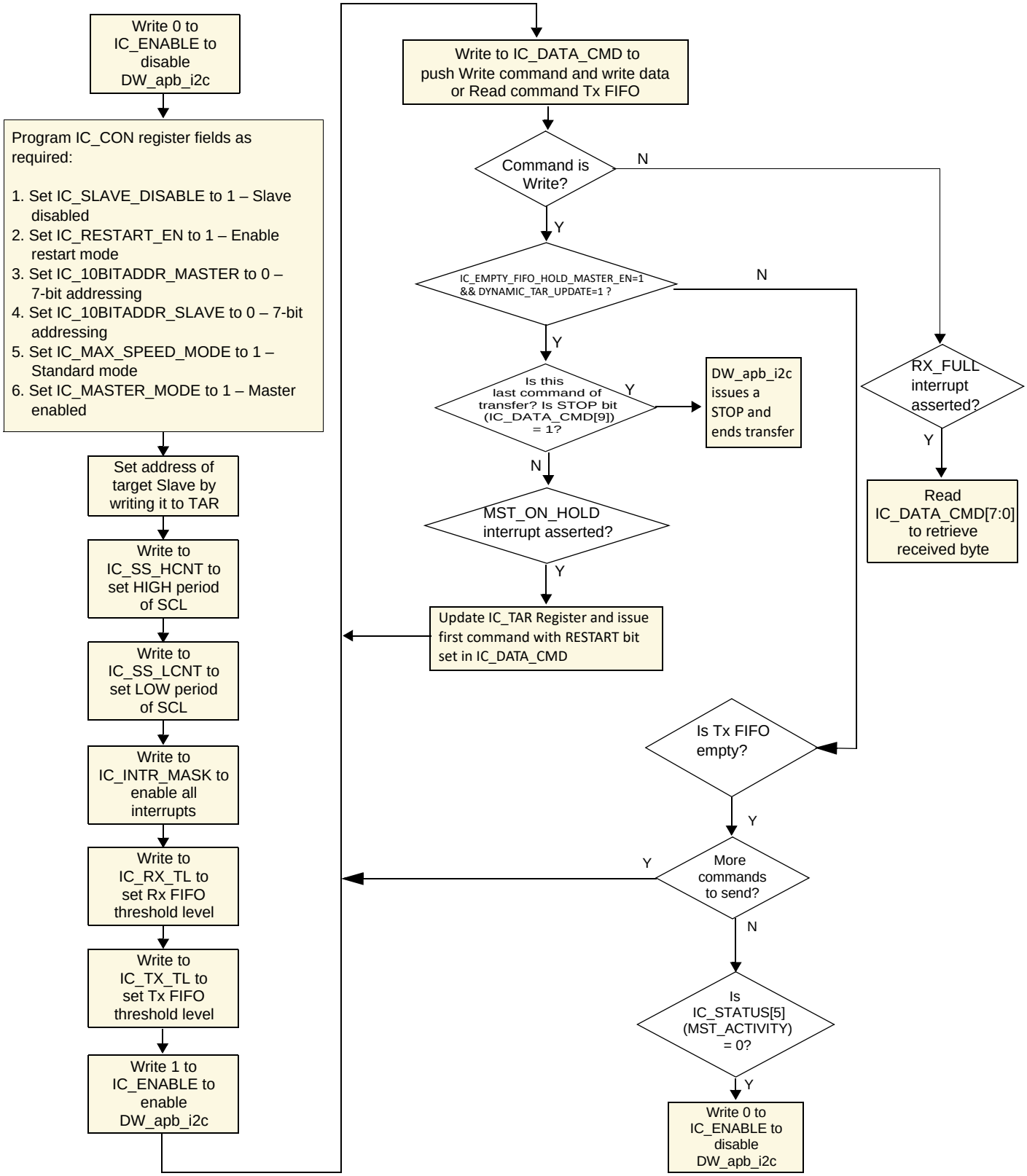
Figure 6-3 Flowchart for DW_apb_i2c Master in Standard Mode, Fast Mode, or Fast Mode Plus



The flow diagram in Figure 6-4 shows a programming example for DW_apb_i2c as master with TAR address update. This flow shows how the MST_ON_HOLD interrupt is used when the software needs information from the hardware to safely update the TAR address.

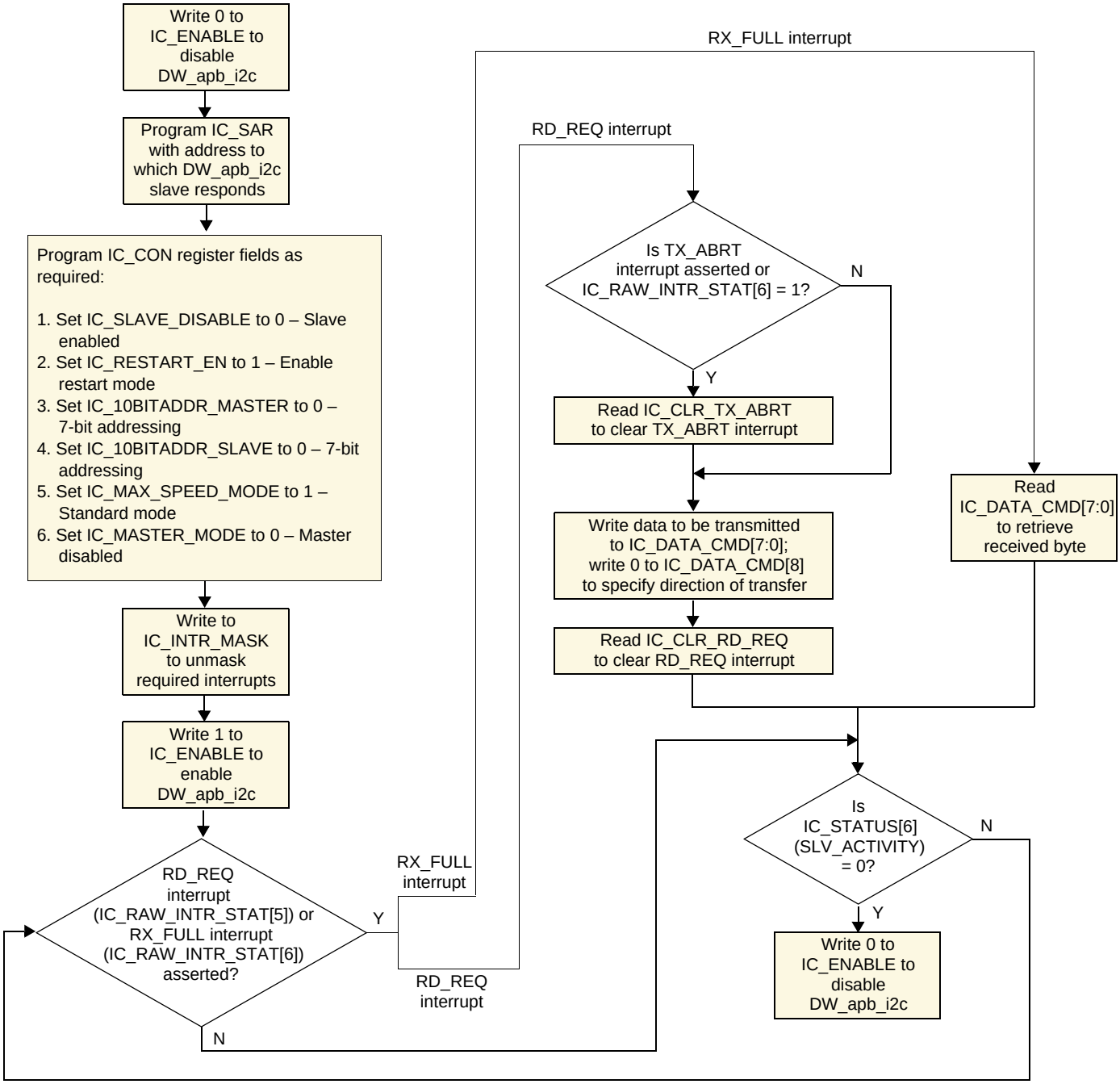
Note When the software has full knowledge of when it is safe to update the TAR address without requiring information from hardware, the MST_ON_HOLD interrupt is not required to update the TAR address.

Figure 6-4 Flowchart for DW_apb_i2c Master with TAR Address Update



The flow diagram in Figure 6-5 shows a programming example for the DW_apb_i2c Slave in standard mode, fast mode, or fast mode plus with 7-bit addressing.

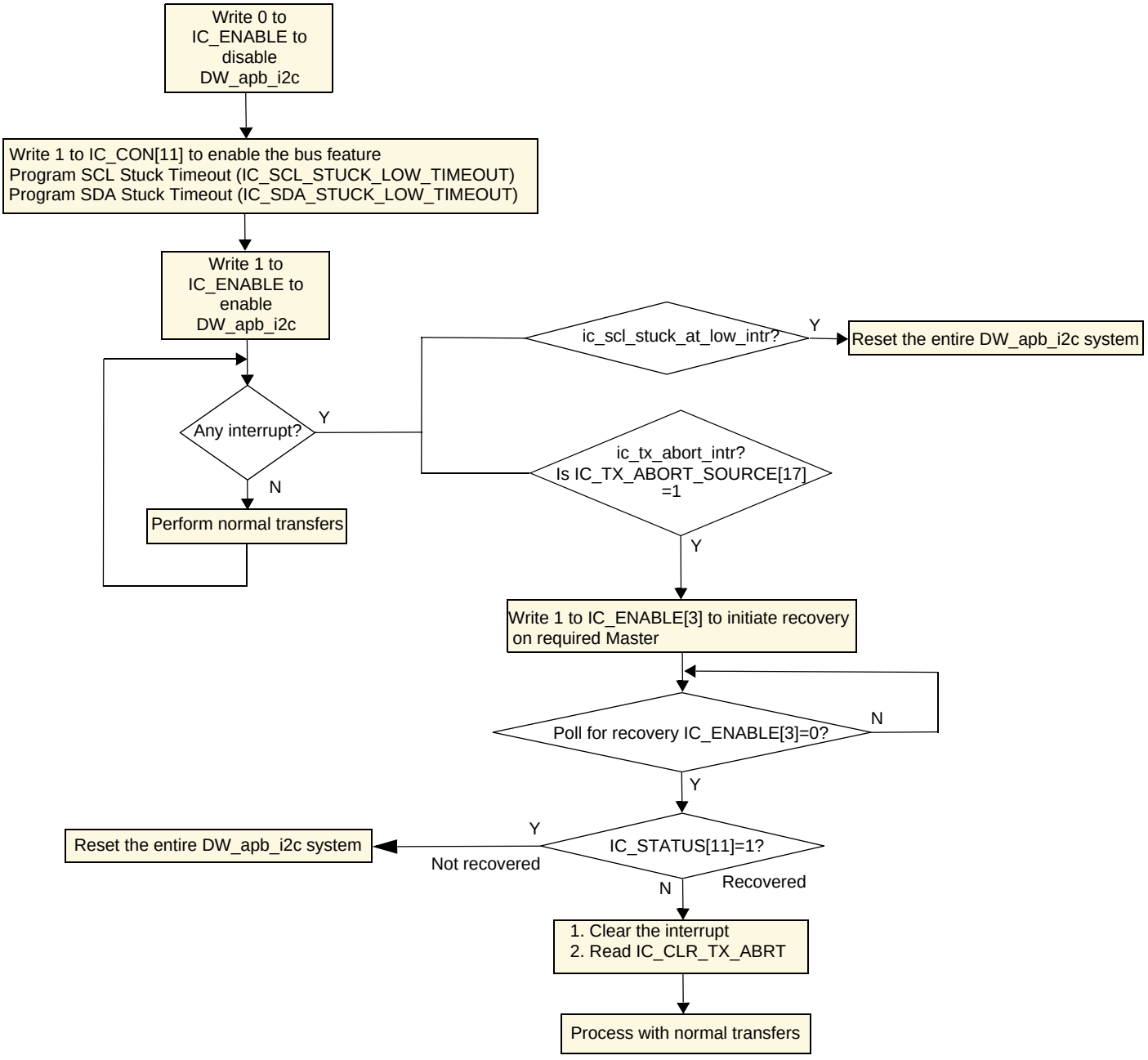
Figure 6-5 Flowchart for DW_apb_i2c Slave in Standard Mode, Fast Mode, or Fast Mode Plus with 7-bit Addressing



6.4 Programming Flow for SCL and SDA Bus Recovery

The flow diagram in Figure 6-6 shows a programming example for SCL and SDA bus recovery.

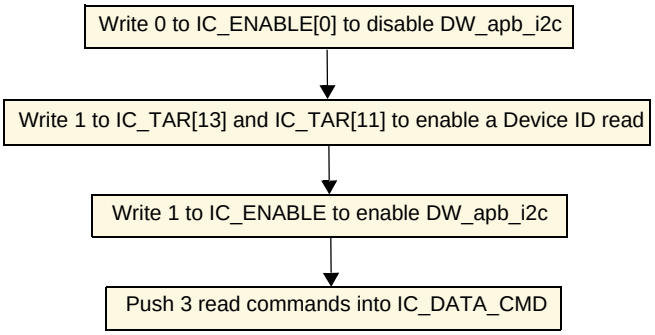
Figure 6-6 Flowchart for SCL and SDA Bus Recovery



6.5 Programming Flow for Reading the Device ID

Figure 6-7 shows a programming flow in the master to initiate a Device ID read.

Figure 6-7 Flowchart for Reading a Device ID

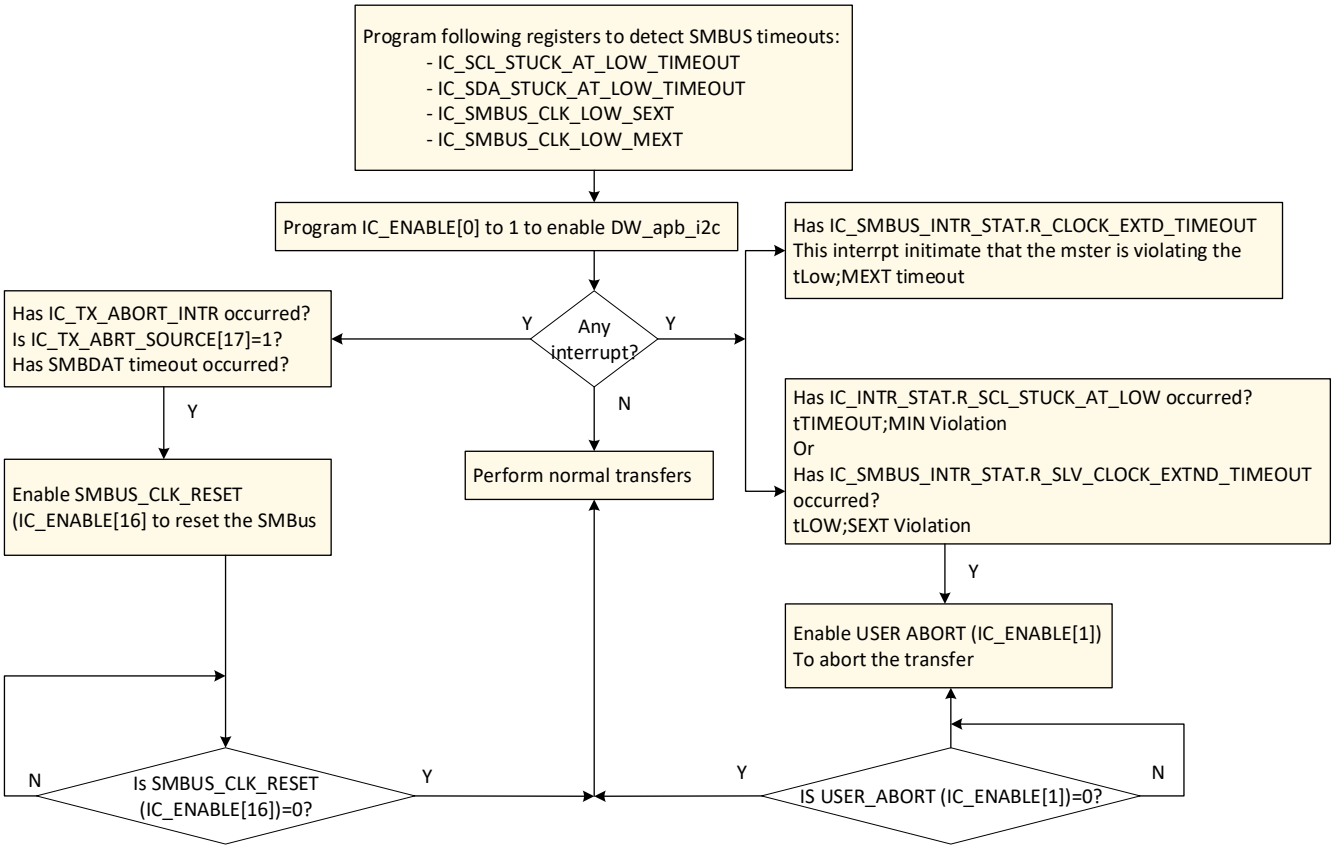


As the Device ID consists of 3 bytes, the user must issue 3 read commands in IC_DATA_CMD register. One read command populates one byte of Device ID in RX FIFO. If more than 3 commands are issued, the Device ID rolls back.

6.6 Programming Flow for SMBUS Timeout in Master Mode

Figure 6-8 shows a programming flow for SMBus timeout in master mode.

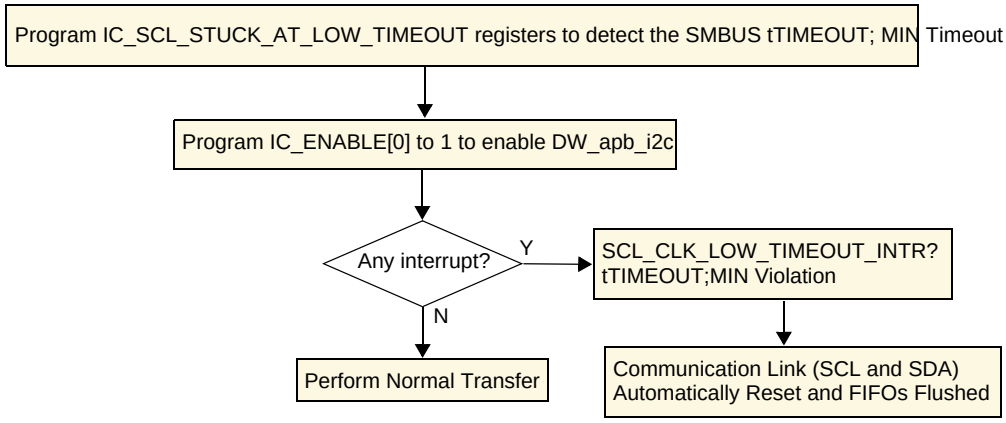
Figure 6-8 SMBUS Timeout Programming Flow in Master Mode



6.7 Programming Flow for SMBUS Timeout in Slave Mode

Figure 6-9 shows a programming flow for SMBus timeout in slave mode.

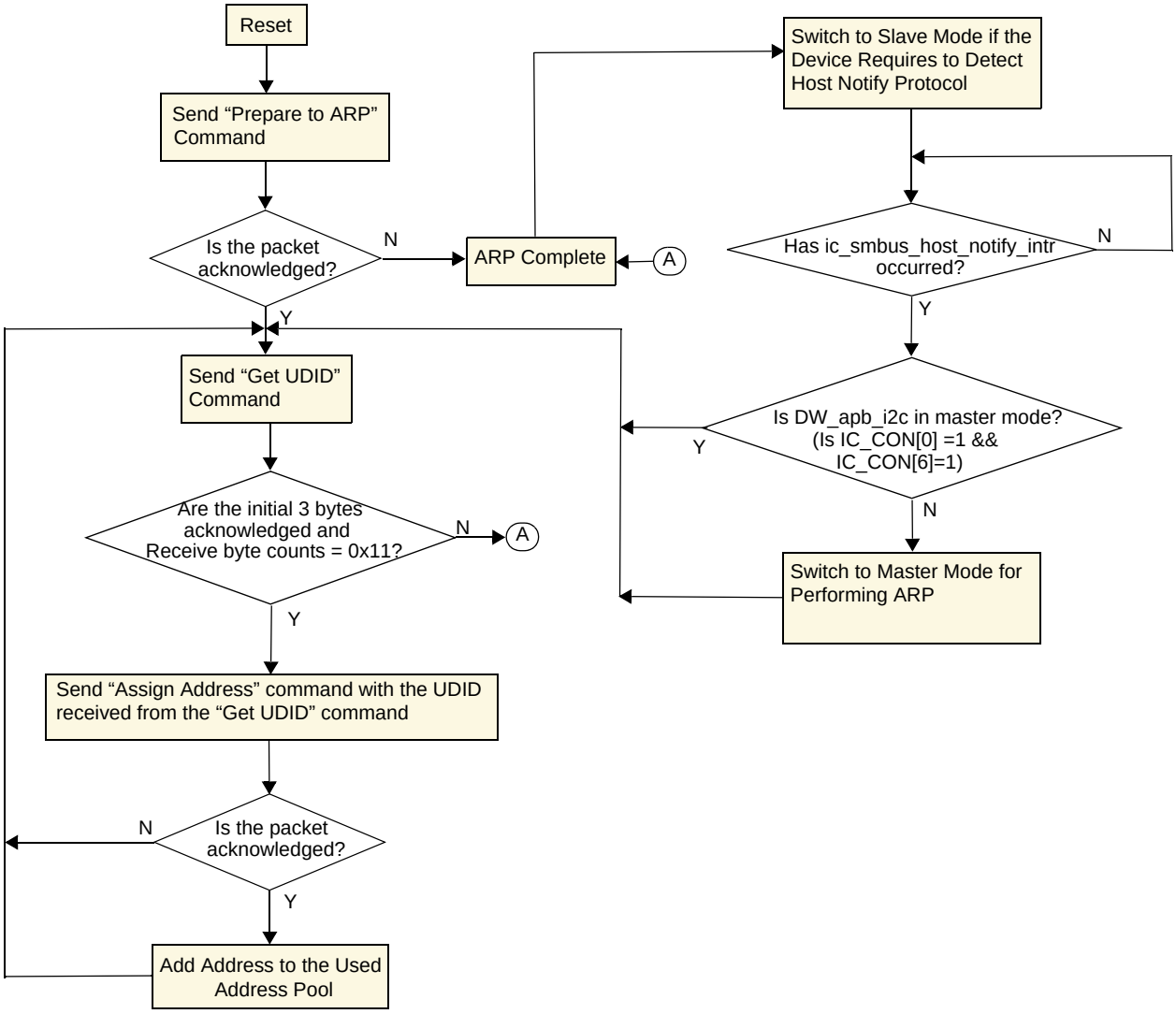
Figure 6-9 SMBUS Timeout Programming Flow in Slave Mode



6.8 ARP Master Programming Flow

Figure 6-10 shows the programming flow for an ARP master.

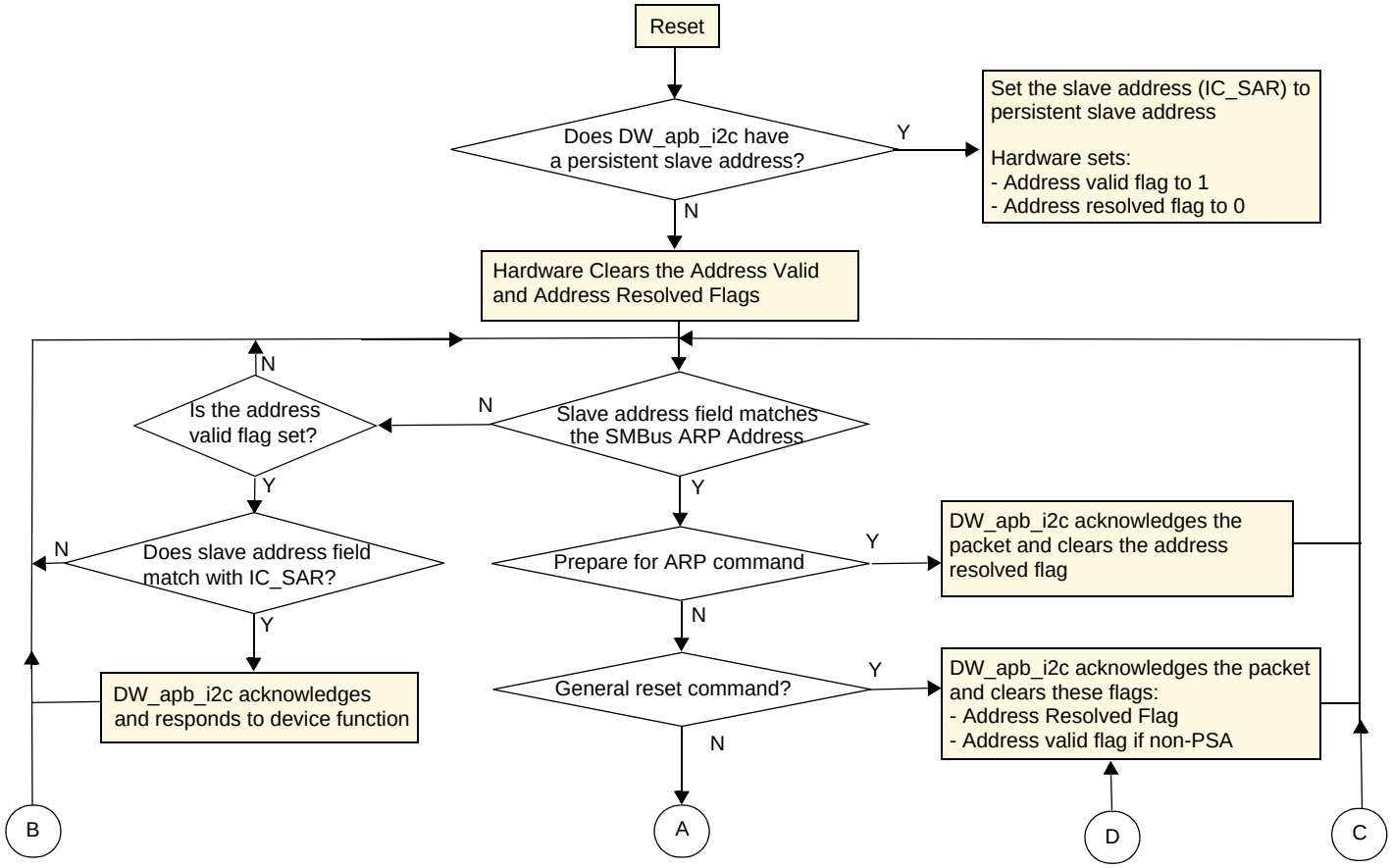
Figure 6-10 ARP Master Programming Flow

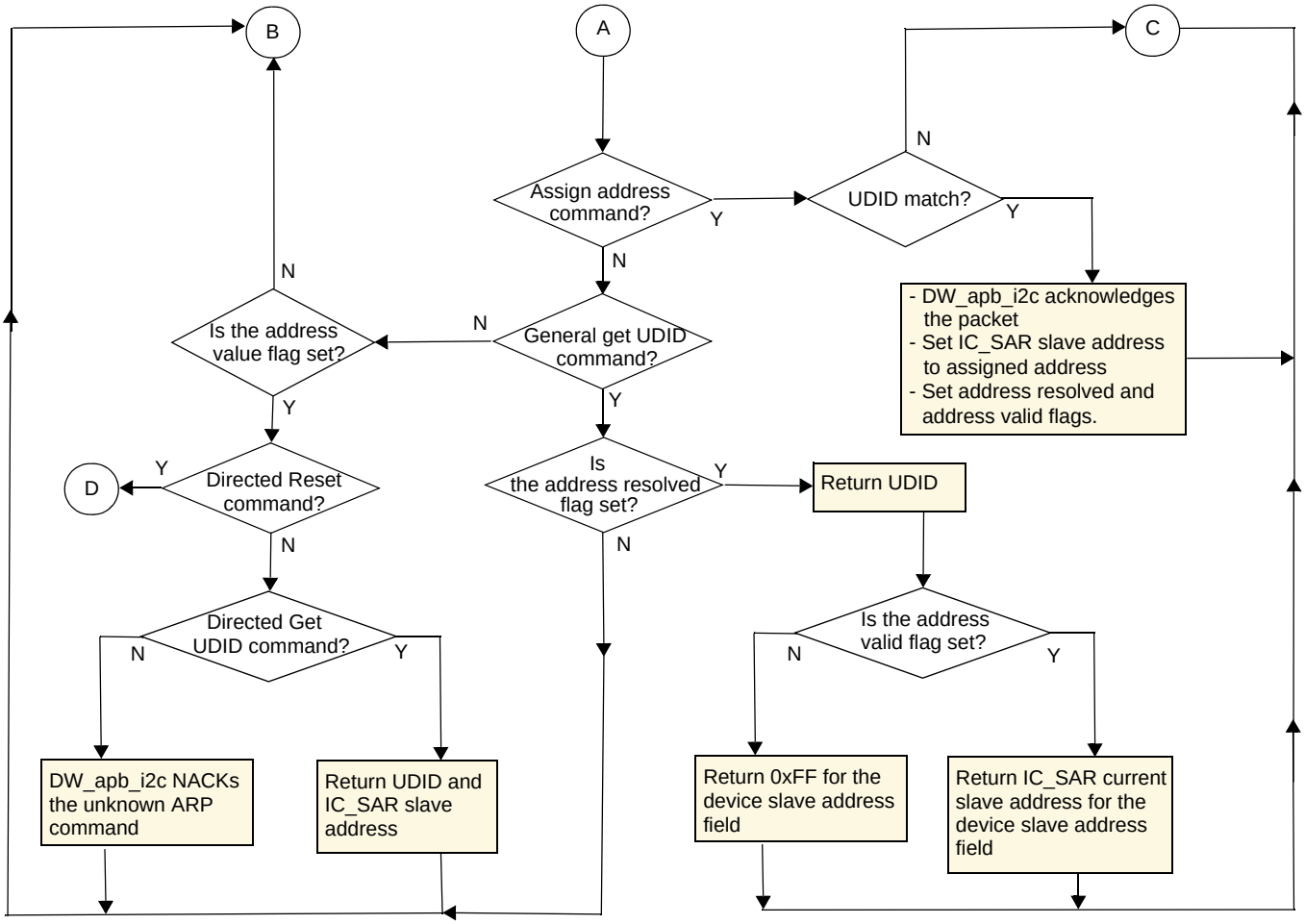


6.9 ARP Slave Programming Flow

Figure 6-11 shows the programming flow for an ARP slave.

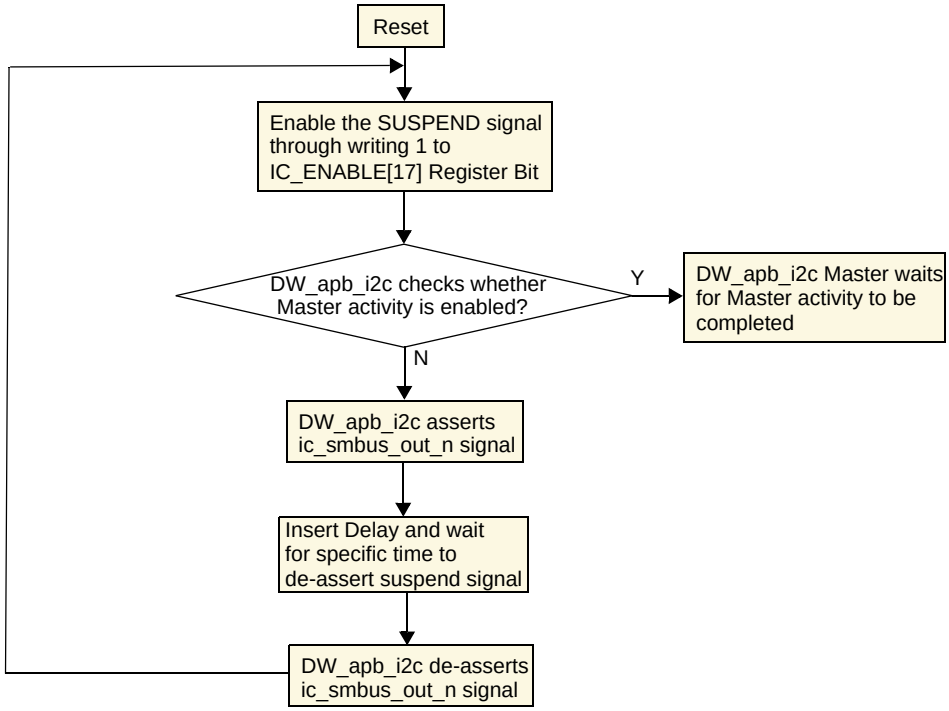
Figure 6-11 ARP Slave Programming Flow




A
B
C
D
Assign address command?
Y
N
UDID match?
N
Y
- DW_apb_i2c acknowledges the packet
- Set IC_SAR slave address to assigned address
- Set address resolved and address valid flags.
General get UDID command?
N
Y
Is the address value flag set?
N
Y
Is the address resolved flag set?
Y
N
Return UDID
Directed Reset command?
Y
N
Directed Get UDID command?
N
Y
Is the address valid flag set?
N
Y
DW_apb_i2c NACKs the unknown ARP command
Return UDID and IC_SAR slave address
Return 0xFF for the device slave address field
Return IC_SAR current slave address for the device slave address field

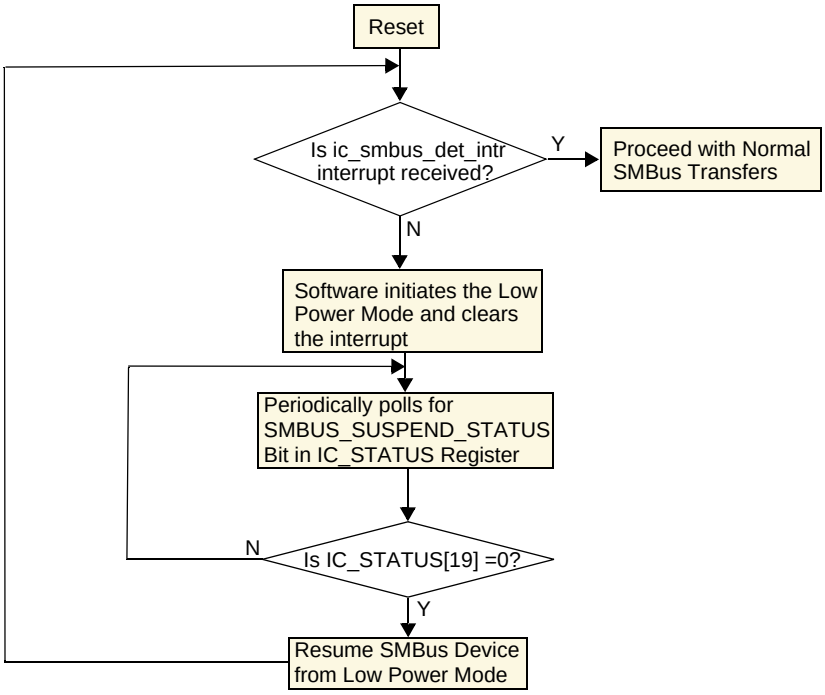
6.10 SMBus SUSPEND Programming Flow in Host Mode

Figure 6-12 Suspend Programming Flow in Host Mode



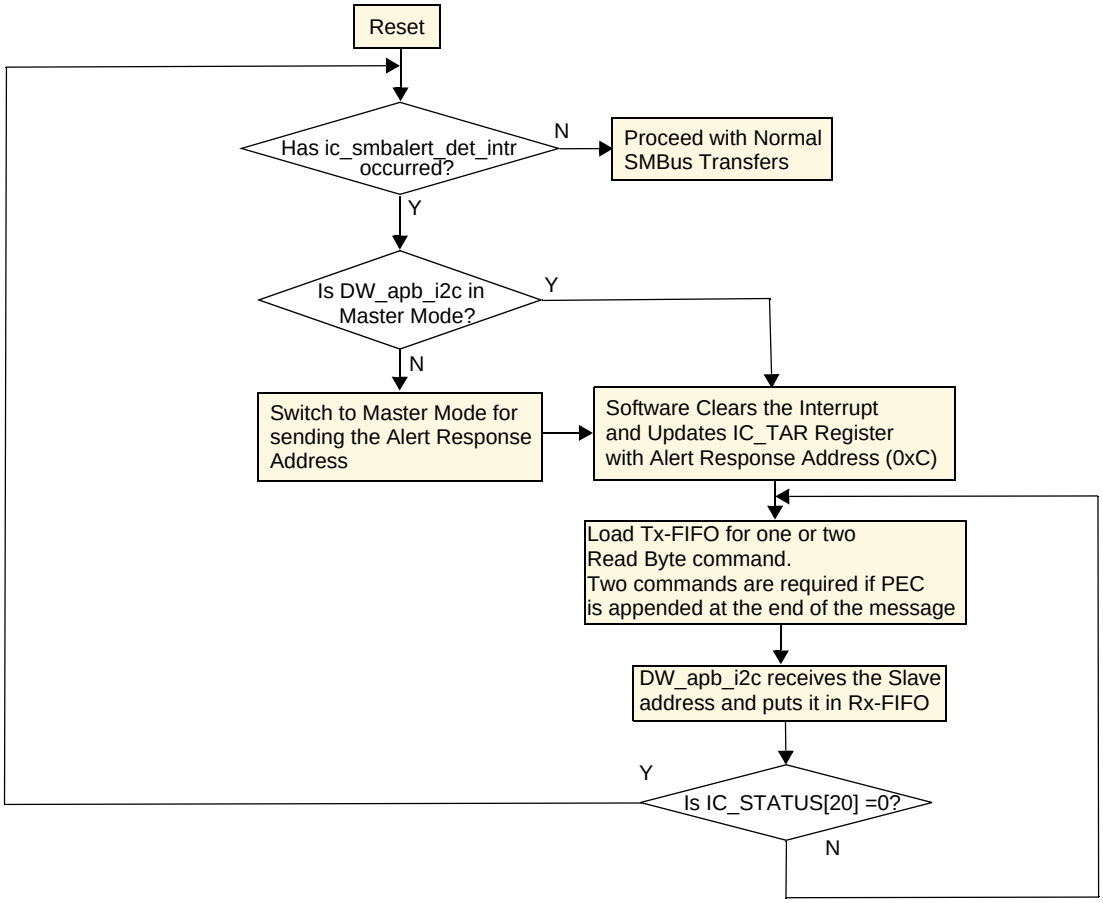
6.11 SMBus SUSPEND Programming Flow in Device Mode

Figure 6-13 SMBus SUSPEND Programming flow in Device Mode



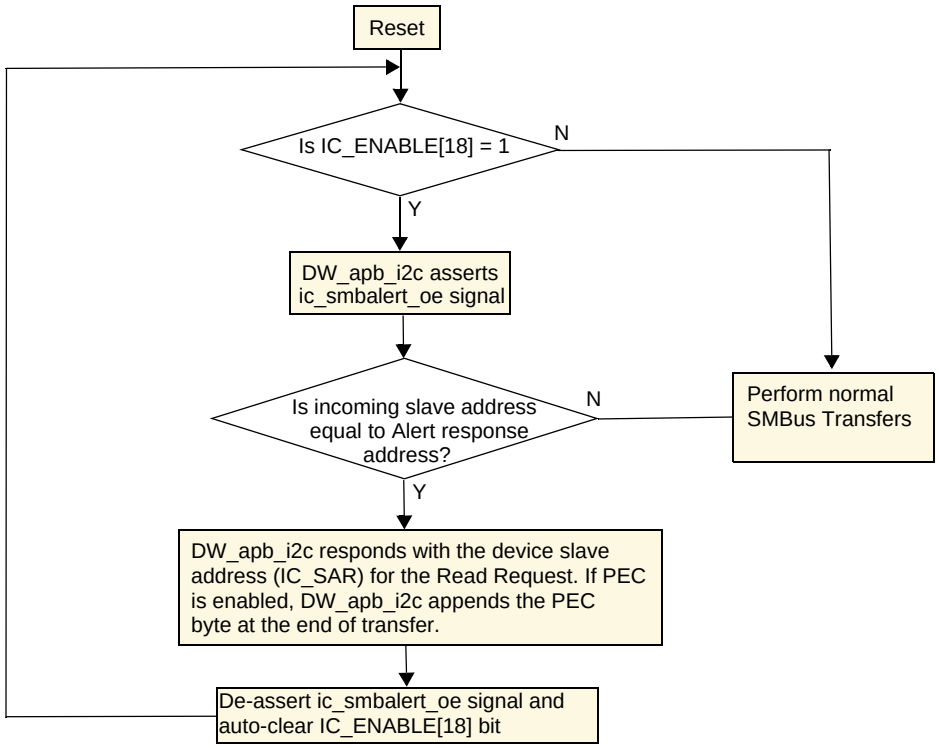
6.12 SMBus ALERT Programming Flow in Host Mode

Figure 6-14 SMBus Alert Programming Flow in Host Mode



6.13 SMBus ALERT Programming Flow in Device Mode

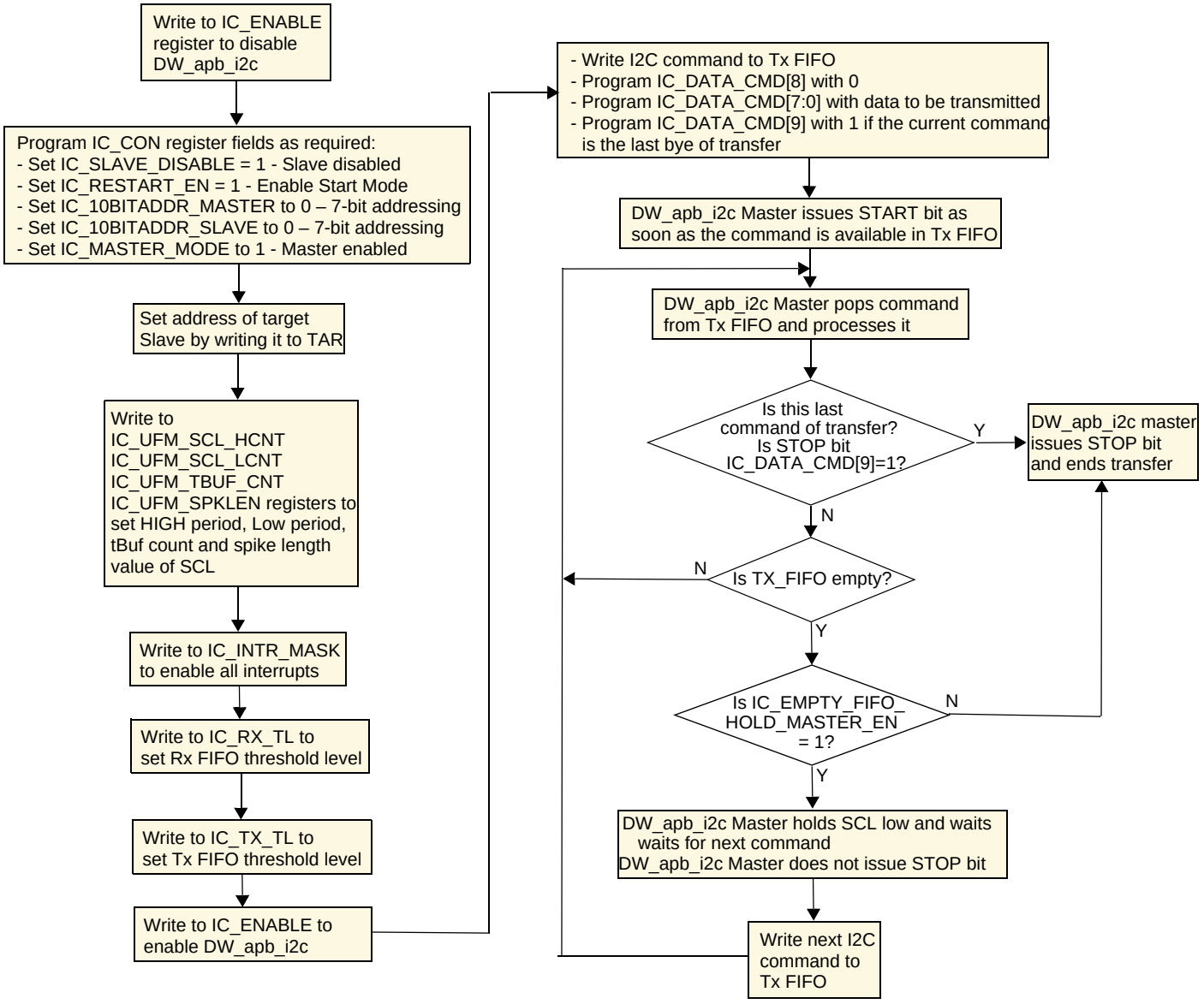
Figure 6-15 SMBus Alert Programming Flow in Device Mode



6.14 Programming Flow Of DW_apb_i2c in Ultra-Fast Mode

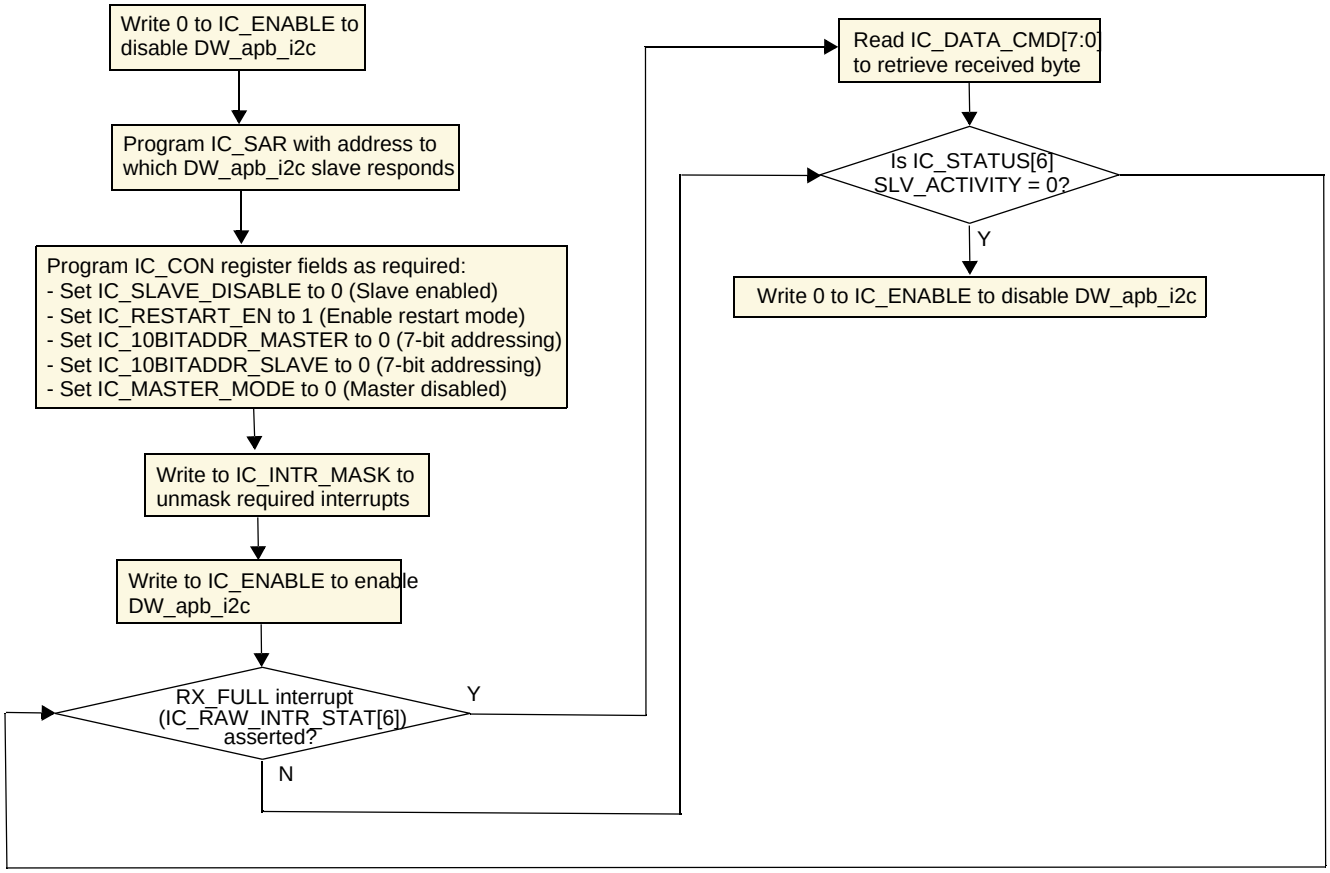
6.14.1 DW_apb_i2c Master Mode

Figure 6-16 DW_apb_i2c Ultra-Fast Master Mode



6.14.2 DW_apb_i2c Slave Mode

Figure 6-17 DW_apb_i2c Ultra-Fast Slave Mode



7 Verification

This chapter provides an overview of the testbench available for DW_apb_i2c verification. Once you have configured the DW_apb_i2c in coreConsultant and have set up the verification environment, you can run simulations automatically.

DW_apb_i2c consists of the following types of environments:

- "Vera Testbench Environment" – Uses the AMBA VMT VIPs and I2C BFM models.

7.1 Vera Testbench Environment

7.1.1 Overview of Vera Tests

The DW_apb_i2c verification testbench performs the following set of tests that have been written to exhaustively verify the functionality and have also achieved maximum RTL code coverage.

Note

The DW_apb_i2c verification testbench is built with DesignWare Verification IP (VIP). Make sure you have the supported version of the VIP components for this release, otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, see the following web page:

www.synopsys.com/products/designware/docs/doc/amba/latest/dw_amba_install.pdf

- All tests use the APB Interface to program memory mapped registers dynamically during tests.

7.1.2 APB Slave Interface

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

- All non-configuration parameter register reset values are verified.
- All read-only registers are written to with opposite values to verify that they are read only.
- All writable registers are written to with opposite values to verify that they can be written.
- Some registers can be written only when the DW_apb_i2c is disabled.Confirm that those registers are non-writable in that mode. Attempt to write the opposite values to those registers while the DW_apb_i2c is disabled and verify that the writes are ignored.

- The *CNT registers can be written to only if the configuration parameter IC_HC_COUNT_VALUES = 0. Verify that the registers are read-only when IC_HC_COUNT_VALUES = 0 and writable when IC_HC_COUNT_VALUES = 1.
- Confirm that it is not possible to write the transmit buffer threshold level (IC_TX_TL) higher than the size of the transmit buffer. Verify that if a larger value is written that the value becomes set to the size of the transmit buffer (max).
- Confirm that it is not possible to write the receive buffer threshold level (IC_RX_TL) higher than the size of the transmit buffer. Verify that if a larger value is written that the value becomes set to the size of the transmit buffer (max).
- Write illegal value 0 to SPEED bits in IC_CON and verify that the new value is parameter IC_MAX_SPEED_MODE.
- Verify that the SPEED bits in IC_CON cannot be written to higher speeds than configuration parameter IC_MAX_SPEED_MODE.

7.1.3 DW_apb_i2c Master Operation

This suite of tests is run only when the DW_apb_i2c is configured as a master. For instance, these tests go through all combinations of speed, addressing, read / write, and multi-byte transfers. Commands are issued to the DW_apb_i2c, and the I^2C Slave is the target and used to verify the transfers. The tests also verify the following:

- SCL low and SCL high times are with I^2C specification
- Operation of all registers
- Master arbitration
- Debug outputs
- Disabling of DW_apb_i2c shown correctly on ic_en output
- Programmed count values for all the *CNT registers
- The current source enable output operates correctly
- Combined format operation (7- and 10-bit addressing modes)
- Restart enable and disable
- Clock synchronization by stretching SCL
- Loop-back operation by performing simultaneous master-transmitter, slave-receiver sending multiple bytes. A single-byte transfer with master-receiver, slave-transmitter is also performed

7.1.4 DW_apb_i2c Slave Operation

This suite of tests is run only when the DW_apb_i2c is configured as a slave. Similar to the tests developed for the master, the driving force is the Serial Master BFM. For instance, these tests go through all combinations of speed, addressing, read / write, and multi-byte transfers. The I^2C master is used to generate

transfers and the DW_apb_i2c is the target; the AHB Master is used to verify the transfers. The tests also verify the following:

- Operation of all registers
- Debug outputs
- Disabling of DW_apb_i2c shown correctly on ic_en output
- Combined format operation (7- and 10-bit addressing modes)

7.1.5 DW_apb_i2c Interrupts

These tests verify that the DW_apb_i2c generates and handles the servicing of interrupts correctly. They also verify operation of the debug ports.

7.1.6 DMA Handshaking Interface

These tests verify that DW_apb_i2c generates and responds through the handshaking interface. Transfers are generated within the DMA BFM and transmitted through the I^2C protocol from the DUT to the ALT_DUT and vice versa. Different watermark levels are selected to control the clearing on the dma_tx_req / dma_rx_req lines once an acknowledgement is received. A random number of bytes are transferred using only the handshaking interface.

7.1.7 DW_apb_i2c Dynamic IC_TAR and IC_10BITADDR_MASTER Update

This test is run only if the DW_apb_i2c is configured as a master and the parameter I2C_DYNAMIC_TAR_UPDATE = 1. This test verifies that DW_apb_i2c Master Target address (IC_TAR) and the parameter IC_10BITADDR_MASTER can be updated dynamically while the DW_apb_i2c Slave is involved in an I2C transfer on the I2C bus.

7.1.8 Generate NACK as a Slave-Receiver

This test is always run and tests the functionality of DW_apb_i2c, depending on whether the parameter IC_SLV_DATA_NACK_ONLY is set to 0 or 1. This test verifies that ACK / NACKs are generated correctly when DW_apb_i2c is acting as a slave-receiver, depending on whether IC_SLV_DATA_NACK_ONLY register exists (set by having parameter IC_SLV_DATA_NACK_ONLY=1). If the register exists, its value is set to 1 for the duration of the test. If the register exists (and therefore its value is 1), a NACK is generated by the slave when data is sent to it, the transfer is aborted, and data is not written to the receive buffer. Otherwise, ACKs are generated for the duration of the transfer, the transfer completes successfully, and the data is written to the receive buffer successfully.

7.1.9 SCL Held Low for Duration Specified in IC_SDA_SETUP

This test verifies that during a Slave-Receive I^2C transfer, DW_apb_i2c asserts the output port ic_data_oe, holding SCL low for the minimum period specified in the IC_SDA_SETUP register. This only happens every time the I^2C master ACKs a data byte, and the transmit FIFO in DW_apb_i2c is not filled to satisfy this read request.

7.1.10 Generate ACK/NACK for General Call

This test verifies that the IC_ACK_GENERAL_CALL bit controls whether DW_apb_i2c ACK or NACKs an I^2C general call address.

8 Integration Considerations

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

8.1 Accessing Top-level Constraints

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

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

```
set_activity_parameter Synthesize ScriptsOnly 1
```

2. This cC command autocompletes the activity:

```
autocomplete_activity Synthesize
```

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

```
write_sdc <filename>
```

8.2 Performance

This section discusses performance and the hardware configuration parameters that affect the performance of the DW_apb_i2c.

8.2.1 Power Consumption, Frequency, and Area Results

Table 8-1 provides information about the synthesis results (power consumption, frequency, and area) of the DW_apb_i2c using the industry standard 28nm technology library and how it affects performance.

Table 8-1 Power Consumption, Frequency, and Area Results for DW_apb_i2c Using 28nm Technology Library

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption
Default Configuration	pclk: 200 MHz	11297 gates	0.179uW	167.305uW

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption
Maximum SFIFO Configuration: IC_CLK_TYPE=1 IC_HAS_ASYNC_FIFO=0 APB_DATA_WIDTH=32 IC_TX_BUFFER_DEPTH=16 IC_RX_BUFFER_DEPTH=16 SLAVE_INTERFACE_TYPE=2 SLVERR_RESP_EN=1 REG_TIMEOUT_WIDTH=8 HC_REG_TIMEOUT_VALUE=0 REG_TIMEOUT_VALUE=8	pclk: 200 MHz ic_clk: 200 MHz	12869 gates	0.205uW	174.351uW
Maximum AFIFO Configuration: IC_CLK_TYPE=1 IC_HAS_ASYNC_FIFO=1 APB_DATA_WIDTH=32 IC_TX_BUFFER_DEPTH=16 IC_RX_BUFFER_DEPTH=16 SLAVE_INTERFACE_TYPE=2 SLVERR_RESP_EN=1 REG_TIMEOUT_WIDTH=8 HC_REG_TIMEOUT_VALUE=0 REG_TIMEOUT_VALUE=8	pclk: 200 MHz ic_clk: 200 MHz	13375 gates	0.213uW	226.352uW
Maximum smbus SFIFO Configuration: IC_CLK_TYPE=1 IC_HAS_ASYNC_FIFO=0 APB_DATA_WIDTH=32 IC_TX_BUFFER_DEPTH=16 IC_RX_BUFFER_DEPTH=16 SLAVE_INTERFACE_TYPE=2 SLVERR_RESP_EN=1 REG_TIMEOUT_WIDTH=8 HC_REG_TIMEOUT_VALUE=0 REG_TIMEOUT_VALUE=8 IC_SMBUS=1 IC_SMBUS_UDID_HC=0 IC_SMBUS_ARP=1	pclk: 200 MHz ic_clk: 200 MHz	20027 gates	0.319uW	60.7887uW

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption
Maximum smbus AFIFO Configuration: IC_CLK_TYPE=1 IC_HAS_ASYNC_FIFO=1 APB_DATA_WIDTH=32 IC_TX_BUFFER_DEPTH=16 IC_RX_BUFFER_DEPTH=16 SLAVE_INTERFACE_TYPE=2 SLVERR_RESP_EN=1 REG_TIMEOUT_WIDTH=8 HC_REG_TIMEOUT_VALUE=0 REG_TIMEOUT_VALUE=8 IC_SMBUS=1 IC_SMBUS_UDID_HC=0 IC_SMBUS_ARP=1	pclk: 200 MHz ic_clk: 200 MHz	20560 gates	0.329uW	63.1965uW
Minimum Configuration: IC_CLK_TYPE=0 IC_MAX_SPEED_MODE=1 IC_10BITADDR_MASTER=0 IC_10BITADDR_SLAVE=0 IC_MASTER_MODE=0 IC_TX_BUFFER_DEPTH=2 IC_RX_BUFFER_DEPTH=2 IC_HC_COUNT_VALUES=1	pclk: 200 MHz ic_clk: 200 MHz	5777 gates	0.0884 uW	16.2522uW

A
Synchronizer Methods

This appendix describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_apb_i2c to cross clock boundaries.

This appendix contains the following sections:

- "Synchronizers Used in DW_apb_i2c" on page 326
- "Synchronizer 1: Simple Double Register Synchronizer" on page 327
- "Synchronizer 2: Simple Double Register Synchronizer with Configurable Polarity Reset" on page 327

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

A.1 Synchronizers Used in DW_apb_i2c

Each of the synchronizers and synchronizer sub-modules are comprised of verified DesignWare Basic Core (BCM) RTL designs. The BCM synchronizer designs are identified by the synchronizer type. The corresponding RTL files comprising the BCM synchronizers used in the DW_apb_i2c are listed and cross referenced to the synchronizer type in Table A-1. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

Table A-1 Synchronizers used in DW_apb_i2c

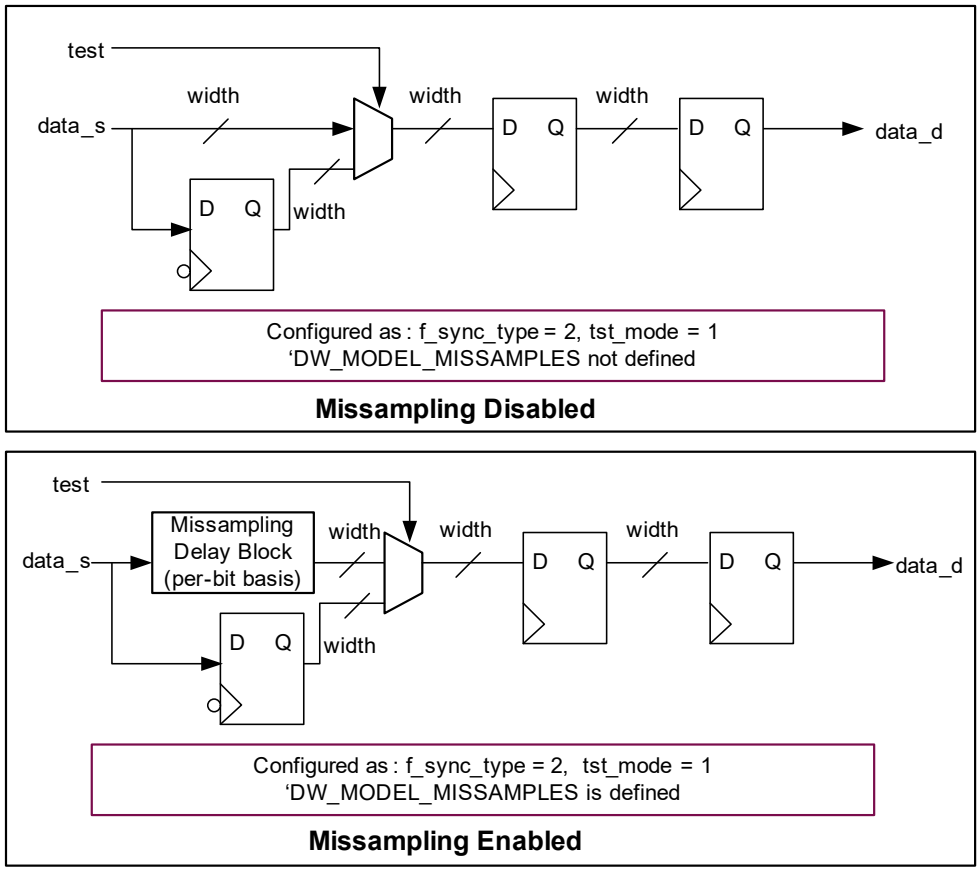
Synchronizer module file	Sub module file	Synchronizer Type and Number
DW_apb_i2c_bcm21.v		Synchronizer 1: Simple Multiple Register Synchronizer
DW_apb_i2c_bcm41.v	DW_apb_i2c_bcm21.v	Synchronizer 2: Simple Multiple Register Synchronizer with Configurable Polarity Reset

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

A.2 Synchronizer 1: Simple Double Register Synchronizer

This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. The synchronization scheme depends on core configuration. If pclk and ic_clk are asynchronous (IC_CLK_TYPE =ASYNC) then DW_apb_i2c_bcm21 is instantiated inside the core for synchronization. This uses two stage synchronization process () both using positive edge of clock.

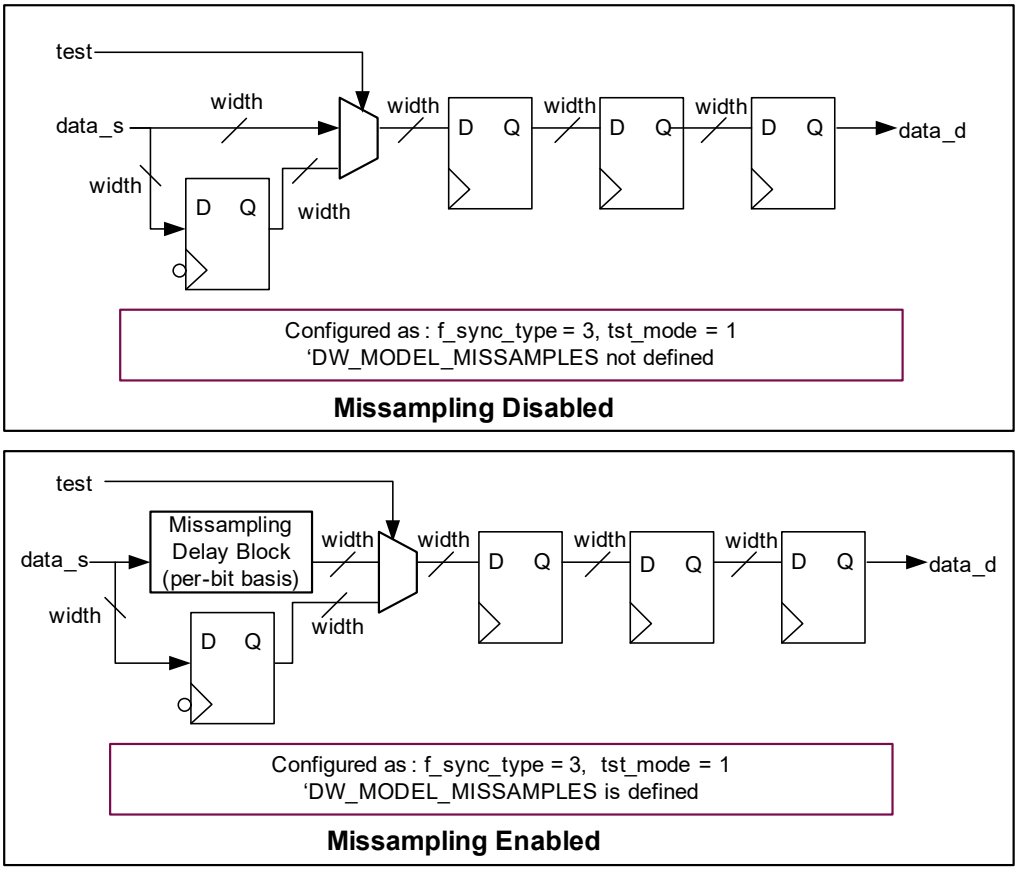
Figure A-1 Block Diagram of Synchronizer 1 With Two Stage Synchronization (Both Positive Edges)



A.3 Synchronizer 2: Simple Double Register Synchronizer with Configurable Polarity Reset

This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries with configurable polarity reset. The synchronization scheme depends on core configuration. If pclk and ic_clk are asynchronous (IC_CLK_TYPE =ASYNC) then DW_apb_i2c_bcm41 is instantiated inside the core for synchronization of ic_clk_in_a and ic_data_in_a input signals. This DW_apb_i2c_bcm41 synchronizer is similar to the DW_apb_i2c_bcm21 synchronizer and the polarity of the output of this synchronizer can be configured. Figure A-2 shows the block diagram of Synchronizer 2.

Figure A-2 Block Diagram of Synchronizer 2 With Two Stage Synchronization (Both Positive Edges)



B
Internal Parameter Descriptions

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

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

Table B-1 Internal Parameters

Parameter Name	Equals To
ASYNC	2'b01
ENCODED_APB_DATA_WIDTH	{[function_of: APB_DATA_WIDTH]}
ENCODED_IC_RX_BUFFER_DEPTH	{[function_of: IC_RX_BUFFER_DEPTH]}
ENCODED_IC_TX_BUFFER_DEPTH	{[function_of: IC_TX_BUFFER_DEPTH]}
IC_ADDR_SLICE_LHS	3'b111
IC_BUS_CLEAR_FEATURE_EN	1
IC_DEFAULT_SDA_RX_HOLD	{[function_of: IC_DEFAULT_SDA_HOLD]}
IC_DEFAULT_SDA_TX_HOLD	{[function_of: IC_DEFAULT_SDA_HOLD]}
IC_FS_MAX_SPKLEN	50
IC_HCNT_LO_LIMIT	=((IC_ULTRA_FAST_MODE ==1) ? 3 : ((IC_CLK_FREQ_OPTIMIZATION == 1) ? 1 : 6))
IC_HIGHSPEED_MODE_EN	=(IC_MAX_SPEED_MODE == 3 ? 1 : 0)
IC_HS_MAX_SPKLEN	10

Table B-1 Internal Parameters (Continued)

Parameter Name	Equals To
IC_LCNT_LO_LIMIT	=((IC_ULTRA_FAST_MODE ==1) ? 5 : ((IC_CLK_FREQ_OPTIMIZATION == 1) ? 6 : 8))
IC_SMBUS_UDID_WORD1_DEFAULT	IC_SMBUS_UDID_MSB & 96'h000000000000000ffffffff
IC_SMBUS_UDID_WORD2_DEFAULT	(IC_SMBUS_UDID_MSB & 96'h00000000ffffffff00000000) >> 32
IC_SMBUS_UDID_WORD3_DEFAULT	(IC_SMBUS_UDID_MSB & 96'hffffffff0000000000000000) >> 64
IC_ULTRA_FAST_MODE_EN	=(IC_ULTRA_FAST_MODE == 1 ? 1 : 0)
IC_VERSION_ID	32'h3230322a
POW_2_REG_TIMEOUT_WIDTH	{[function_of: REG_TIMEOUT_WIDTH]}
RX_ABW	{[function_of: IC_RX_BUFFER_DEPTH]}
RX_ABW_P1	RX_ABW + 1
TX_ABW	{[function_of: IC_TX_BUFFER_DEPTH]}
TX_ABW_P1	TX_ABW + 1

C
Glossary

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

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

Design View	A simulation model for a core generated by coreConsultant.
DesignWare Synthesizable Components	The Synopsys name for the collection of AMBA-compliant coreKits and verification models delivered with DesignWare and used with coreConsultant or coreAssembler to quickly build DesignWare Synthesizable Component designs.
DesignWare cores	A specific collection of synthesizable cores that are licensed individually. For more information, refer to www.synopsys.com/designware.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the DesignWare Synthesizable Components.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multi-byte word; see also little-endian and big-endian.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
GPIO	General Purpose Input Output.
GTECH	A generic technology view used for RTL simulation of encrypted source code by non-Synopsys simulators.
hard IP	Non-synthesizable implementation IP.
HDL	Hardware Description Language – examples include Verilog and VHDL.
IIP	Implementation Intellectual Property — A generic term for synthesizable HDL and non-synthesizable “hard” IP in all of its forms (coreKit, component, core, MacroCell, and so on).
implementation view	The RTL for a core. You can simulate, synthesize, and implement this view of a core in a real chip.
instantiate	The act of placing a core or model into a design.
interface	Set of ports and parameters that defines a connection point to a component.
IP	Intellectual property — A term that encompasses simulation models and synthesizable blocks of HDL code.
little-endian	Data format in which the least-significant byte comes first.
MacroCell	Bigger IP blocks (6811, 8051, memory controller) available in the DesignWare Library and delivered with coreConsultant.
master	Device or model that initiates and controls another device or peripheral.
model	A Verification IP component or a Design View of a core.
monitor	A device or model that gathers performance statistics of a system.

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

Index

A

active command queue
definition 331
activity
definition 331
AHB
definition 331
AMBA
definition 331
APB
definition 331
APB bridge
definition 331
application design
definition 331
arbiter
definition 331
Arbitration, of master 41

B

BFM
definition 331
big-endian
definition 331
Block diagram, of DW_apb_i2c 21
blocked command stream
definition 331
blocking command
definition 332
bus bridge
definition 332

C

Clock synchronization 43
command channel
definition 332
command stream
definition 332
component
definition 332
Configuration
of IC_CLK frequency 66
configuration
definition 332
configuration intent
definition 332
core
definition 332
core developer
definition 332
core integrator
definition 332
coreAssembler
definition 332
coreConsultant
definition 332
coreKit
definition 332
Customer Support 16
cycle command
definition 332

D

decoder
definition 332
design context
definition 332
design creation
definition 332
Design View
definition 333
DesignWare cores
definition 333
DesignWare Library
definition 333

DesignWare Synthesizable Components
definition 333
Disabling DW_apb_i2c
version 1.06a 50
DMA Controller
and DW_apb_i2c 80
dual role device
definition 333
DW_apb_i2c
block diagram of 21
functional behavior 25
functional overview 21
operation modes 43
overview of 26
protocols 31
testbench
overview of tests 317
Dynamic update of IC_TAR
initial configuration of master mode 47
or 10-bit addressing for master mode 49

E

endian
definition 333
Environment, licenses 23

F

Full-Functional Mode
definition 333
Functional behavior, of DW_apb_i2c 25
Functional overview, of DW_apb_i2c 21

G

GPIO
definition 333
GTECH
definition 333

H

hard IP
definition 333
HDL
definition 333

I

IC_CLK frequency, configuration of 66
IIP
definition 333
implementation view
definition 333
instantiate
definition 333
interface
definition 333
Interfaces
DMA Controller 80
IP
definition 333

L

Licenses 23
little-endian
definition 333

M

MacroCell
definition 333
master
definition 333
Master arbitration 41
Master mode 47
model
definition 333
monitor
definition 333

N

non-blocking command
definition 334

O

Operation modes 43

P

peripheral
definition 334
Protocols, of I^2C 31

R

RTL
definition 334

S

SDRAM
definition 334
SDRAM controller
definition 334
Simple double register synchronizer 327
slave
definition 334

Slave mode 44
SoC
  definition 334
SoC Platform
  AHB contained in 19
  APB, contained in 19
  defined 19
soft IP
  definition 334
static controller
  definition 334
subsystem
  definition 334
Synchronizer
  simple double register 327
synthesis intent
  definition 334
synthesizable IP
  definition 334

T

technology-independent
  definition 334
Testsuite Regression Environment (TRE)
  definition 334
TRE
  definition 334

V

Vera, overview of tests 317
Verification
  and Vera tests 317
VIP
  definition 334

W

workspace
  definition 334
wrap
  definition 334
wrapper
  definition 334

Z

zero-cycle command
  definition 334