User's Manual

32

RZ/N1D Group, RZ/N1S Group, RZ/N1L Group

User's Manual: Peripherals

RZ Family
RZ/N Series

Renesas Electronics
www.renesas.com

Rev.1.20 Dec, 2021

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

How to Use This Manual

1. Objective and Target Users

This manual was written to explain the hardware functions and electrical characteristics of this LSI to the target users, i.e. those who will be using this LSI in the design of application systems. Target users are expected to understand the fundamentals of electrical circuits, logic circuits, and microcomputers.

This manual is organized in the following items: an overview of the product, descriptions of the CPU, system control functions, and peripheral functions, electrical characteristics of the device, and usage notes.

When designing an application system that includes this LSI, take all points to note into account.
Points to note are given in their contexts and at the final part of each section, and in the section giving usage notes.

The list of revisions is a summary of major points of revision or addition for earlier versions. It does not cover all revised items. For details on the revised points, see the actual locations in the manual.

The following documents have been prepared for reference.

■ Documents related to RZ/N1

Document Name	Document Number
RZ/N1D Group, RZ/N1S Group, RZ/N1L Group DATASHEET	R01DS0323EJ****
RZ/N1D Group, RZ/N1S Group, RZ/N1L Group User's Manual: System Introduction, Multiplexing, Electrical and Mechanical Information	R01UH0750EJ****
RZ/N1D Group, RZ/N1S Group, RZ/N1L Group User's Manual: System Control and Peripheral	R01UH0751EJ****
RZ/N1D Group, RZ/N1S Group, RZ/N1L Group User's Manual: Peripherals	R01UH0752EJ**** (this manual)
RZ/N1D Group, RZ/N1S Group, RZ/N1L Group User's Manual: R-IN Engine and Ethernet Peripherals	R01UH0753EJ****
RZ/N1D Group, RZ/N1S Group, RZ/N1L Group User's Manual: PWMTimer	R01UH0913EJ****

2. Description of Registers

Each register description includes a bit chart, illustrating the arrangement of bits, and a table of bits, describing the meanings of the bit settings. The standard format and notation for bit charts and tables are described below.

X.X.X [Register Name]

(2)

Address: XXXX XXXXh

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	[Bit Field]					—	—	[Bit Field]		—	[Bit]	[Bit]	[Bit]
Value after reset	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0

Table X.X [Register Name] Register Contents

Bit Position	Bit Name	Function	R/W
b12 to b8	[Bit Field]	[Description]	R/W
b5 to b4	[Bit Field]	[Description] 2'b00: Hi-Z 2'b01: L Output Others: Prohibited (3)	R/W
b2	[Bit]	[Description] 1'b0: H-Z 1'b1: Output (defult)	R/W
b1	[Bit]	[Description] 1'b0: H-Z 1'b1: Output (defult)	R/W
b0	[Bit]	[Description] 1'b0: H-Z 1'b1: Output	R/W (1)

(1) R/W: The bit or field is readable and writable.

R/(W): The bit or field is readable and writable. However, writing to this bit or field has some limitations. For details on the limitations, see the description or notes of respective registers.

R: The bit or field is readable. Writing to this bit or field has no effect.

W: The bit or field is writable. Reading to this bit or field is not guaranteed.

(2) Reserved. Make sure to use the specified value when writing to this bit or field; otherwise, the correct operation is not guaranteed.

(3) Setting prohibited. The correct operation is not guaranteed if such a setting is performed.

3. List of Abbreviations and Acronyms

Abbreviation	Full Form
AHB	Arm Advanced High-performance Bus
APB	Arm Advanced Peripheral Bus
AXI	Arm Advanced eXtensible Interface
bps	bits per second
CA7	Arm Cortex-A7 module
CM3	Arm Cortex-M3 module
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
Hi-Z	High Impedance
HSR	High-availability Seamless Redundancy
HW-RTOS	Hard Ware Real Time OS
I/O	Input/Output
INTC	Interrupt Controller
LSB	Least Significant Bit
MSB	Most Significant Bit
NC	Non-Connect
NoC	Network-on-Chip
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
UART	Universal Asynchronous Receiver/Transmitter
OTP	One Time Programmable
PTP	Precision Time Protocol
PRP	Parallel Redundancy Protocol
SoC	System On Chip

4. Description of the Access Size

Access size:

8 bits = Byte

16 bits = Halfword

32 bits = Word

CAN (Controller Area Network): An automotive network specification developed by Robert Bosch GmbH of Germany

Table of Contents

Section 1 UART 20
1.1 Overview 20
1.2 Signal Interfaces 22
1.3 Register Map 23
1.3.1 Register Map UART 1 23
1.3.2 Register Map UART 2 24
1.3.3 Register Map UART 3 25
1.3.4 Register Map UART 4 26
1.3.5 Register Map UART 5 27
1.3.6 Register Map UART 6 28
1.3.7 Register Map UART 7 29
1.3.8 Register Map UART 8 30
1.4 Register Description 31
1.4.1 rUart_DLL — Divisor Latch (Low) 31
1.4.2 rUart_DLH — Divisor Latch (High) 32
1.4.3 rUart_IIR — Interrupt Identification Register 33
1.4.4 rUart_RBR_THR — Receive Buffer/Transmit Holding Register 34
1.4.5 rUart_IER — Interrupt Enable Register 35
1.4.6 rUart_FCR — FIFO Control Register 37
1.4.7 rUart_LCR — Line Control Register 39
1.4.8 rUart_MCR — Modem Control Register 41
1.4.9 rUart_LSR — Line Status Register 43
1.4.10 rUart_MSR — Modem Status Register 46
1.4.11 rUart_SCR — Scratchpad Register 48
1.4.12 rUart_SRBR_STHR — Shadow Receive Buffer/Transmit Holding Register 49
1.4.13 rUart_FAR — FIFO Access Register 50
1.4.14 rUart_TFR — Transmit FIFO Read 51
1.4.15 rUart_RFW — Receive FIFO Write 52
1.4.16 rUart_USR — UART Status Register 53
1.4.17 rUart_TFL — Transmit FIFO Level 55
1.4.18 rUart_RFL — Receive FIFO Level 56
1.4.19 rUart_SRR — Software Reset Register 57
1.4.20 rUart_SRTS — Shadow Request to Send 58
1.4.21 rUart_SBCR — Shadow Break Control Register 59
1.4.22 rUart_SFE — Shadow FIFO Enable 60
1.4.23 rUart_SRT — Shadow RCVR Trigger 61
1.4.24 rUart_STET — Shadow TX Empty Trigger 62
1.4.25 rUart_HTX — Halt TX 63
1.4.26 rUart_DMASA — DMA Software Acknowledge 64
1.4.27 rUart_TO — Time-Out Counter Configuration Register 65

1.4.28 rUart_CTRLTO — Time-Out Control Register 67
1.4.29 rUart_STATUSTO — Time-Out Counter Status Register 69
1.4.30 rUart_TDMACR — DMA Control Register in Transmit Mode 71
1.4.31 rUart_RDMACR — DMA Control Register in Receive Mode 73
1.5 Operation 75
1.5.1 Main Function Blocks Description 75
1.5.1.1 UART (RS232) Serial Protocol 75
1.5.1.2 Baud Rate Tolerance to 19200 baud 76
1.5.1.3 FIFO Management 77
1.5.1.4 Clock Management 77
1.5.1.5 Back to Back Character Stream Transmission 77
1.5.1.6 Interrupts 78
1.5.1.7 Auto Flow Control 80
1.5.1.8 Programmable THRE interrupt 82
1.5.1.9 DMA Management (Only UART4, 5, 6, 7, 8) 84
1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management 90
1.5.2 Usage Notes 96

Section 2 SPI 97
2.1 Overview 97
2.2 Signal Interfaces 99
2.3 Register Map 100
2.3.1 Register Map SPI1 (Master) 100
2.3.2 Register Map SPI2 (Master) 101
2.3.3 Register Map SPI3 (Master) 102
2.3.4 Register Map SPI4 (Master) 103
2.3.5 Register Map SPI5 (Slave) 104
2.3.6 Register Map SPI6 (Slave) 105
2.4 Register Description 106
2.4.1 rSpi_CTRLR0 — Control Register 0 106
2.4.2 rSpi_CTRLR1 — Control Register 1 108
2.4.3 rSpi_SSIENR — Enable Register 109
2.4.4 rSpi_MWCR — Microwire Control Register 110
2.4.5 rSpi_SER — Slave Enable Register 111
2.4.6 rSpi_BAUDR — Baud Rate Select 113
2.4.7 rSpi_TXFTLR — Transmit FIFO Threshold Level 114
2.4.8 rSpi_RXFTLR — Receive FIFO Threshold Level 115
2.4.9 rSpi_TXFLR — Transmit FIFO Level Register 116
2.4.10 rSpi_RXFLR — Receive FIFO Level Register 117
2.4.11 rSpi_SR — Status Register 118
2.4.12 rSpi_IMR — Interrupt Mask Register 120
2.4.13 rSpi_ISR — Interrupt Status Register 121
2.4.14 rSpi_RISR — Raw Interrupt Status Register 122
2.4.15 rSpi_TXOICR — Transmit FIFO Overflow Interrupt Clear Register 123

2.4.16 rSpi_RXOICR — Receive FIFO Overflow Interrupt Clear Register 124
2.4.17 rSpi_RXUICR — Receive FIFO Underflow Interrupt Clear Register 125
2.4.18 rSpi_ICR — Interrupt Clear Register 126
2.4.19 rSpi_DMACR — DMA Control Register 127
2.4.20 rSpi_DMATDLR — DMA Transmit Data Level 128
2.4.21 rSpi_DMARDLR — DMA Receive Data Level 129
2.4.22 rSpi_DR — Data Register 130
2.4.23 rSpi_RX_SAMPLE_DLY — RXD Sample Delay Register 131
2.4.24 rSpi_TDMACR — DMA Control Register in Transmit Mode 132
2.4.25 rSpi_RDMACR — DMA Control Register in Receive Mode 134
2.5 Operation 136
2.5.1 General description 136
2.5.2 Typical Connection between SPI Master & Slave 137
2.5.3 Control Slave Select Line by Hardware or Software Mode 138
2.5.4 Programmable Prescaler Clock 139
2.5.5 Data Input Sample Delay 140
2.5.6 Transmit & Receive FIFO & Control 141
2.5.7 Interruption Management 141
2.5.8 Transfer Mode 143
2.5.8.1 Transmit and Receive Mode 143
2.5.8.2 Transmit Only Mode 143
2.5.8.3 Receive Only Mode 143
2.5.8.4 EEPROM Read Mode 144
2.5.9 Motorola Serial Peripheral Interface 145
2.5.10 Texas Instruments Synchronous Serial Protocol 148
2.5.11 National Semiconductor Microwire 149
2.5.12 DMA Control 155
2.5.12.1 Overview on DMA Operation 156
2.5.12.2 Transmit Watermark Level and Transmit FIFO Underflow 156
2.5.12.3 Choosing the Transmit Watermark Level 156
2.5.12.4 Selecting DEST_MSIZE and Transmit FIFO Overflow 158
2.5.12.5 Receive Watermark Level and Receive FIFO Overflow 158
2.5.12.6 Choosing the Receive Watermark Level 159
2.5.12.7 Selecting SRC_MSIZE and Receive FIFO Underflow 159
2.6 Usage Notes 160
2.6.1 Programming Consideration 160
2.6.1.1 Programming Master SPI in Motorola & Texas Mode 160
2.6.1.2 Programming Master SPI in National Semiconductor Mode 163
2.6.1.3 Programming Slave SPI in Motorola & Texas Mode 165
2.6.1.4 Programming Slave SPI in National Semiconductor Mode 167

Section 3 I2C 168
3.1 Overview 168
3.2 Signal Interfaces 169
3.3 Register Map 170

3.3.1 I2C1 Register Map 170
3.3.2 I2C2 Register Map 171
3.4 Register Description 172
3.4.1 IC_CON — I2C Control Register 172
3.4.2 IC_TAR — I2C Target Address Register 174
3.4.3 IC_SAR — I2C Slave Address Register 175
3.4.4 IC_DATA_CMD — I2C Rx/Tx Data Buffer and Command Register 176
3.4.5 IC_SS_SCL_HCNT — Standard mode I2C Clock SCL High Count Register 178
3.4.6 IC_SS_SCL_LCNT — Standard mode I2C Clock SCL Low Count Register 179
3.4.7 IC_FS_SCL_HCNT — Fast mode I2C Clock SCL High Count Register 179
3.4.8 IC_FS_SCL_LCNT — Fast mode I2C Clock SCL Low Count Register 180
3.4.9 IC_INTR_STAT — I2C Interrupt Status Register 181
3.4.10 IC_INTR_MASK — I2C Interrupt Mask Register 182
3.4.11 IC_RAW_INTR_STAT — I2C Raw Interrupt Status Register 184
3.4.12 IC_RX_TL — I2C Receive FIFO Threshold Register 187
3.4.13 IC_TX_TL — I2C Transmit FIFO Threshold Register 188
3.4.14 IC_CLR_INTR — Clear Combined and Individual Interrupt Register 188
3.4.15 IC_CLR_RX_UNDER — Clear RX_UNDER Interrupt Register 189
3.4.16 IC_CLR_RX_OVER — Clear RX_OVER Interrupt Register 189
3.4.17 IC_CLR_TX_OVER — Clear TX_OVER Interrupt Register 190
3.4.18 IC_CLR_RD_REQ — Clear RD_REQ Interrupt Register 190
3.4.19 IC_CLR_TX_ABRT — Clear TX_ABRT Interrupt Register 191
3.4.20 IC_CLR_RX_DONE — Clear RX_DONE Interrupt Register 191
3.4.21 IC_CLR_ACTIVITY — Clear ACTIVITY Interrupt Register 192
3.4.22 IC_CLR_STOP_DET — Clear STOP_DET Interrupt Register 192
3.4.23 IC_CLR_START_DET — Clear START_DET Interrupt Register 193
3.4.24 IC_CLR_GEN_CALL — Clear GEN_CALL Interrupt Register 193
3.4.25 IC_ENABLE — I2C Enable Register 194
3.4.26 IC_STATUS — I2C Status Register 195
3.4.27 IC_TXFLR — I2C Transmit FIFO Level Register 197
3.4.28 IC_RXFLR — I2C Receive FIFO Level Register 198
3.4.29 IC_SDA_HOLD — I2C SDA Hold Time Length Register 199
3.4.30 IC_TX_ABRT_SOURCE — I2C Transmit Abort Source Register 200
3.4.31 IC_SLV_DATA_NACK_ONLY — Generate Slave Data NACK Register 202
3.4.32 IC_SDA_SETUP — I2C SDA Setup Register 203
3.4.33 IC_ACK_GENERAL_CALL — I2C ACK General Call Register 204
3.4.34 IC_ENABLE_STATUS — I2C Enable Status Register 205
3.4.35 IC_FS_SPKLEN — I2C Sm, Fm Spike Suppression Limit 207
3.4.36 IC_CLR_RESTART_DET — Clear RESTART_DET Interrupt Register 208
3.4.37 IC_COMP_PARAM_1 — Component Parameter Register 1 209
3.5 Operation Modes 210
3.5.1 Slave Mode Operation 210
3.5.1.1 Initial Configuration 210

3.5.1.2 Slave Transmitter Operation for a Single Byte 211
3.5.1.3 Slave Receiver Operation for a Single Byte 212
3.5.1.4 Slave Transfer Operation for Bulk Transfer 212
3.5.2 Master Mode Operation 214
3.5.2.1 Initial Configuration 214
3.5.2.2 Dynamic IC_TAR or IC_10BITADDR_MASTER Update 214
3.5.2.3 Master Transmit and Master Receive 215
3.5.3 Disabling the I2C controller 216
3.5.3.1 Procedure 216
3.5.4 Aborting the I2C Transfer 217
3.5.4.1 Procedure 217
3.6 Programming the I2C Controller 218
3.6.1 Spike Suppression 218
3.6.2 I2C_SCLK Frequency Configuration 219
3.6.2.1 Minimum High and Low Counts 219
3.6.3 SDA Hold Time 221
3.6.3.1 SDA Hold Timings in Receiver 221
3.6.3.2 SDA Hold Timings in Transmitter 222

Section 4 Basic GPIO 223
4.1 Overview 223
4.2 Signal Interfaces 225
4.3 Register Map 226
4.3.1 Register Map BGPIO1 226
4.3.2 Register Map BGPIO2 226
4.3.3 Register Map BGPIO3 227
4.4 Register Description 228
4.4.1 rGPIO_swporta_dr — GPIO Port A Data Output Register 228
4.4.2 rGPIO_swporta_ddr — GPIO Port A Data Direction Register 228
4.4.3 rGPIO_swportb_dr — GPIO Port B Data Output Register 229
4.4.4 rGPIO_swportb_ddr — GPIO Port B Data Direction Register 229
4.4.5 rGPIO_inten — GPIO Port A Interrupt Enable Register 230
4.4.6 rGPIO_intmask — GPIO Port A Interrupt Mask Register 230
4.4.7 rGPIO_inttype_level — GPIO Port A Interrupt Level Register 231
4.4.8 rGPIO_int_polarity — GPIO Port A Interrupt Polarity Register 231
4.4.9 rGPIO_intstatus — GPIO Port A Interrupt Status 232
4.4.10 rGPIO_raw_intstatus — GPIO Port A Raw Interrupt Status (Premasking) 232
4.4.11 rGPIO_porta_eoi — GPIO Port A Clear Interrupt Register 233
4.4.12 rGPIO_ext_porta — GPIO Port A Data Input Register 233
4.4.13 rGPIO_ext_portb — GPIO Port B Data Input Register 234
4.4.14 rGPIO_ls_sync — GPIO Port A Level-Sensitive Synchronization Enable Register 234
4.5 Operation 235
4.5.1 Main Functions Blocks Description 235
4.5.1.1 Data & Control Flow 235
4.5.1.2 Interruption (Only Port A) 235

4.5.1.3 Programmable Interrupts Routed on Cortex-A7 and M3 236
4.5.1.4 Trigger Synchronous Operation 237
4.6 Usage Notes 239
4.6.1 Programming Consideration 239

Section 5 Timer Block 240
5.1 Overview 240
5.2 Signal Interfaces 241
5.3 Register Map 242
5.3.1 TIMER1 Register Map 242
5.3.2 TIMER2 Register Map 242
5.4 Register Description 243
5.4.1 rTimerLoadCount_[n] — Preset Value of Sub-timer[n] (n = 0..5) 243
5.4.2 rTimerLoadCount_[n] — Preset Value of Sub-timer[n] (n = 6..7) 244
5.4.3 rTimerCurrentCount_[n] — Current Value of Sub-timer[n] (n = 0..5) 245
5.4.4 rTimerCurrentCount_[n] — Current Value of Sub-timer[n] (n = 6..7) 245
5.4.5 rTimerControl_[n] — Control Mode of Sub-timer[n] (n = 0..7) 246
5.4.6 rTimerClearInt_[n] — Clears the Interruption of Sub-timer[n] (n = 0..7) 247
5.4.7 rTimerStatusInt0_[n] — Interruption Status before Masking of Sub-timer[n] (n = 0..7) 248
5.4.8 rTimerStatusInt1_[n] — Interruption Status after Masking of Sub-timer[n] (n = 0..7) 249
5.4.9 rTimerAllClearInt — Clear All Interrupt 250
5.4.10 rTimerAllStatusInt0 — All Interrupts Status before Masking 251
5.4.11 rTimerAllStatusInt1 — All Interrupts Status after Masking 252
5.4.12 rTimer_DMA_Pending — TIMER DMA Requests Status 253
5.4.13 rTimer_DMA_PendingOvf — TIMER DMA Overflow Status 254
5.4.14 rTimer_DMA_PendingClrOvf — TIMER DMA Overflow Clear 255
5.5 Operation 256
5.5.1 Prescaler Counter 256
5.5.2 Counter 16 or 32 Bits 256
5.5.3 Interruption 259
5.5.4 DMA Control 260
5.6 Usage Notes 261

Section 6 CAN 262
6.1 Overview 262
6.2 Signal Interfaces 264
6.3 Register Map 265
6.3.1 Register Map (CAN1) 265
6.3.2 Register Map (CAN2) 266
6.4 Register Description 267
6.4.1 rCan_MOD — Configuration Mode Register 267
6.4.2 rCan_CMR — Command Register 269

6.4.3 rCan_SR — Controller Status Register 271
6.4.4 rCan_IR — Interrupt Register 273
6.4.5 rCan_IER — Interrupt Event Register....275
6.4.6 rCan_BTR0 — Bus Timing Register 0 276
6.4.7 rCan_BTR1 — Bus Timing Register 1 277
6.4.8 rCan_OCR — Output Control Register....278
6.4.9 rCan_ALC — Arbitration Lost Capture Register 279
6.4.10 rCan_ECC — Error Code Capture Register 280
6.4.11 rCan_EWLR — Error Warning Limit Register....282
6.4.12 rCan_RXERR — Receive Error Counter Register....283
6.4.13 rCan_TXERR — Transmit Error Counter Register 284
6.4.14 rCan_WrTransmitBuffer — Write Transmit Buffer Register....286
6.4.15 rCan_RdReceiveBuffer — Read Receive Buffer Register 287
6.4.16 rCan_ACR[n] — Acceptance Code Filter [n] Register (n = 0..3) 288
6.4.17 rCan_AMR[n] — Acceptance Mask Filter [n] Register (n = 0..3)....289
6.4.18 rCan_RMC — Receive Message Counter Register 290
6.4.19 rCan_RBSA — Receive Buffer Start Address Register....291
6.4.20 rCan_ReceiveFifo — Receive FIFO Register....292
6.4.21 rCan_RdTransmitBuffer — Read Transmit Buffer Register 293
6.4.22 rCan_SyncTransmitBuffer — Sync Frame Transmit Buffer Register 294
6.4.23 rCan_SyncPeriod — Time Window Sync Frame Transmission Register....295
6.4.24 rCan_SyncStatusInt — Sync Frame Interrupt Status Register....297
6.4.25 rCan_SyncMaskInt — Sync Frame Mask Interrupt Register....299
6.4.26 rCan_SyncClearInt — Sync Frame Clear Interrupt Register....300
6.4.27 rCan_SyncStatus — Sync Frame Status Configuration Register....301
6.4.28 rCan_SyncClearSetRunStop — Sync Frame Generation Register....303
6.4.29 rCan_SyncPassiveError — Sync Passive Error Detection Register 304
6.5 Operation 305
6.5.1 Main Features Description 305
6.5.2 Operation Mode 305
6.5.3 Transmission....306
6.5.4 Reception 307
6.5.5 Self Reception....308
6.5.6 Sleep Mode 309
6.5.7 Acceptance Filtering 309
6.5.8 Interrupts Generation 312
6.5.8.1 Receive Interrupts....312
6.5.8.2 Transmit Interrupts....313
6.5.8.3 Error Warning Interrupts 313
6.5.8.4 Data Overrun Interrupts 313
6.5.8.5 Wakeup Interrupts....314
6.5.8.6 Error Passive Interrupts 314
6.5.8.7 Arbitration Loss Interrupts....314
6.5.8.8 Bus Error Interrupts....315

6.5.8.9 Transmit “Sync frame” Interrupts 315
6.5.8.10 Transmit Overrun “Sync frame” Interrupts 316
6.5.9 Bus Arbitration 317
6.5.10 Error Handling 318
6.5.11 Transmit Buffer Layout 320
6.5.11.1 Descriptor Field of the Transmit Buffer 321
6.5.11.2 Frame Format (FF) 321
6.5.11.3 Remote Request (RTR) 321
6.5.11.4 Data Length Code (DLC) 321
6.5.11.5 Identifier (ID) 322
6.5.11.6 Data Field 322
6.5.12 Receive Buffer Layout 323
6.5.13 Bit Period and Bus Timing Parameters 324
6.5.14 Reset Mode 327
6.5.15 Synchronization Frame 328
6.5.15.1 CANopen Synchronous Frame Configuration 328
6.5.15.2 CANopen Emission of “Sync Frame” 333
6.5.16 Difference between CAN Controllers and Reference Philips SJA1000 Devices 336
6.6 Special Notice 337

Section 7 ADC Controller and 12bit A/D Converters 338
7.1 Overview 338
7.1.1 Analog Buffer 340
7.2 Signal Interfaces 341
7.3 Register Map 342
7.3.1 Register Map ADC1 342
7.3.2 Register Map ADC2 343
7.4 Register Description 344
7.4.1 Register Description ADC1 344
7.4.1.1 rADC_INTSTATUS0 — Interrupt Status Before Masking 344
7.4.1.2 rADC_INTSTATUS1 — Interrupt Status After Masking 345
7.4.1.3 rADC_INTCLR — Clear Interrupt 346
7.4.1.4 rADC_INTMASK — Mask Interrupt 347
7.4.1.5 rADC_INTOVFSTATUS0 — Interrupt Overflow Before Masking 348
7.4.1.6 rADC_INTOVFSTATUS1 — Interrupt Overflow After Masking 349
7.4.1.7 rADC_INTCLROVF — Clear Interrupt Overflow 350
7.4.1.8 rADC_INTOVFMASK — Mask Interrupt Overflow 351
7.4.1.9 rADC_PENDING — Start of Operation Pending 352
7.4.1.10 rADC_PENDINGOVF — Start of Operation Pending Overflow 354
7.4.1.11 rADC_PENDINGCLROVF — Clear Start of Operation Overflow 355
7.4.1.12 rADC_CONTROL — ADC Control 356
7.4.1.13 rADC_FORCE — ADC Request 357
7.4.1.14 rADC_SETFORCE — Set ADC Request 358
7.4.1.15 rADC_CLRFORCE — Clear ADC Request 359
7.4.1.16 rADC_PRIORITY — ADC Priority Mode 360
7.4.1.17 rADC_CONFIG — ADC Configuration 362
7.4.1.18 rADC_ACQS — ADC Control Sample and Hold 364
7.4.1.19 rADC_MASKLOCK[n] — Mask Data Locked [n] (n = 0..3) 365

7.4.1.20 rADC_VC[n] — ADC Control Register for Virtual Channel [n] (n = 0..15) 366
7.4.1.21 rADC1_DATA[n] — ADC1 Conversion Data of Virtual Channel [n] (n = 0..15) 371
7.4.1.22 rADC1_DATALOCK[n] — ADC1 DataLock[n] Register (n = 0..15) 372
7.4.2 Register Description ADC2 373
7.4.2.1 rADC2_DATA[n] — ADC2 Conversion Data of Virtual Channel [n] (n = 0..15) 373
7.4.2.2 rADC2_DATALOCK[n] — ADC2 DataLock[n] Register (n = 0..15) 374
7.5 Operation 375
7.5.1 Virtual Channel ADC_VC Principle Operation 376
7.5.2 Electric ADC Model and Acquisition Sample 381
7.5.3 Trigger Selection and Event Management 383
7.5.4 Physical Channel Selection 384
7.5.5 ADC Operation Priority 385
7.5.6 Simultaneous Sample and Hold 388
7.5.7 End of Command (EOC) and Interrupt Operation 390
7.5.8 Data Copy in Data Lock Register 391
7.5.9 Timing 393
7.5.9.1 Basic A/D Conversion on 3 Channels 393
7.5.9.2 Sample & Hold following by A/D Conversion on One Channel 394
7.5.9.3 Sample & Hold following by A/D Conversion on 3 Channels 396
7.5.9.4 Power Down 398
7.5.9.5 A/D Conversion Rate 399
7.5.10 DMA control 400
7.5.10.1 Overview on DMA Operation 401
7.6 Usage Notes 402
7.6.1 Restriction 402

Section 8 LCD Controller 403
8.1 Overview 403
8.2 Signal Interfaces 405
8.3 Register Map 406
8.3.1 Coding Palette (Palette Registers) Map 406
8.4 Register Description 407
8.4.1 rLcd_CR1 — Control Register 1 407
8.4.2 rLcd_HTR — Horizontal Timing Register 410
8.4.3 rLcd_VTR1 — Vertical1 Timing Register 411
8.4.4 rLcd_VTR2 — Vertical2 Timing Register 412
8.4.5 rLcd_PCTR — Pixel Clock Timing Register 413
8.4.6 rLcd_ISR — Interrupt Status Register Before Masking 414
8.4.7 rLcd_IMR — Interrupt Mask Register 416
8.4.8 rLcd_IVR — Interrupt Status Register After Masking 417
8.4.9 rLcd_ISCR — Interrupt Scan Compare Register 419
8.4.10 rLcd_DBAR — DMA Start Base Address of Frame Buffer Memory 420
8.4.11 rLcd_DCAR — DMA Current Base Address on Going 421
8.4.12 rLcd_DEAR — DMA End Address 422

8.4.13 rLcd_PWMFR_0 — PWM0 Frequency Register 423
8.4.14 rLcd_PWMDCR_0 — PWM0 Duty Cycle Register 424
8.4.15 rLcd_HVTER — Horizontal and Vertical Timing Extension Register....425
8.4.16 rLcd_HPPLOR — Horizontal Pixels-Per-Line Override Control 426
8.4.17 rLcd_PWMFR_1 — PWM1 Frequency Register 427
8.4.18 rLcd_PWMDCR_1 — PWM1 Duty Cycle Register 428
8.4.19 rLcd_GPIOR — Blink Control 429
8.4.20 rLcd_CIR — Core Identification Register....430
8.4.21 Coding Palette (Palette registers) Description....431
8.4.21.1 rLcd_PAL_RGB_555 — Coding Palette when RGB 5:5:5 Mode 431
8.4.21.2 rLcd_PAL_RGB_565 — Coding Palette when RGB 5:6:5 Mode 432
8.4.21.3 rLcd_PAL_BGR_555 — Coding Palette when BGR 5:5:5 Mode 433
8.4.21.4 rLcd_PAL_BGR_565 — Coding Palette when BGR 5:6:5 Mode 434
8.5 Operation 435
8.5.1 Main Features Description 435
8.5.2 Bandwidth Limitation 436
8.5.3 Timing and Control....437
8.5.4 DMA Controller and Memory Interface 439
8.5.5 Frame Buffer Organization....439
8.5.6 Input FIFO 439
8.5.7 Pixel Unpack 439
8.5.8 Palette Lookup Table 442
8.5.9 Output FIFO and Formatter 443
8.5.10 Initializing Configuration Registers....446
8.5.11 Interrupts 446
8.5.12 Power Sequencing....447
8.5.13 Frame Buffer 24 bpp Packed Word 448
8.5.14 Pulse Width Modulation 449
8.5.15 Blink Function....449
8.5.16 Limitation....451

Section 9 Semaphore 452
9.1 Overview....452
9.2 Signal Interfaces....452
9.3 Register Map 453
9.4 Register Description 454
9.4.1 rSemaphoreLockCPU[m]_[n] — Semaphore Lock CPU[m] Register [n]....454
9.4.2 rSemaphoreStatusCPU[m]_[n] — Semaphore Status CPU[m] Register [n]....455
9.5 Operation 456
9.5.1 Semaphore [n] (n = 0..63)....456
9.5.2 CPU Identify and Address Decoding 457
9.6 Usage Notes 458

Section 10 Medium Speed External Bus Interface (MSEBI) 459
10.1 Overview 459
10.1.1 Signal Interfaces 462
10.1.2 MSEBI Master Address Mapping of CS[n] from CPU 462
10.1.3 Multiplexed Signal Interface 463
10.1.3.1 Mode32 Multiplexer 465
10.1.3.2 Mode16 Multiplexer 467
10.1.3.3 Mode8 Multiplexer 469
10.2 Register Map 471
10.2.1 Register Map MSEBI Master from CPU 471
10.2.2 Register Map MSEBI Master from DMA 471
10.2.3 Register Map MSEBI Slave from CPU 471
10.2.4 Register Map MSEBI Slave from MSEBI 472
10.3 Register Description 473
10.3.1 Register Description MSEBI Master from CPU 473
10.3.1.1 rMSEBIM_CYCLESIZE_CS[n]_N — Chip Select CycleSize Register (n = 0..3) 473
10.3.1.2 rMSEBIM_SETUPHOLD_CS[n]_N — Chip Select SetupHold Register (n = 0..3) 475
10.3.1.3 rMSEBIM_TDMACR_CS[n]_N — DMA Transmit Control and Status Register (n = 0..1) 477
10.3.1.4 rMSEBIM_RDMACR_CS[n]_N — DMA Receive Control and Status Register (n = 0..1) 479
10.3.1.5 rMSEBIM_ADDRDMA_READ_CS[n]_N — DMA Read Address Register (n = 0..1) 481
10.3.1.6 rMSEBIM_ADDRDMA_CURRENTREAD_CS[n]_N — DMA Current Read Address Register (n = 0..1) 482
10.3.1.7 rMSEBIM_ADDRDMA_WRITE_CS[n]_N — DMA Write Address Register (n = 0..1) 483
10.3.1.8 rMSEBIM_ADDRDMA_CURRENTWRITE_CS[n]_N — DMA Current Write Address Register (n = 0..1) 484
10.3.1.9 rMSEBIM_DMATDLR_CS[n]_N — DMA Transmit Data Level Register (n = 0..1) 485
10.3.1.10 rMSEBIM_DMARDLR_CS[n]_N — DMA Receive Data Level Register (n = 0..1) 487
10.3.1.11 rMSEBIM_CONFIG_CS[n]_N — Chip Select Config Register (n = 0..3) 489
10.3.1.12 rMSEBIM_CONFIG — Common Config Register 493
10.3.1.13 rMSEBIM_CPU_FIFOREAD_FLUSH — Flush Receive FIFO Register 495
10.3.2 Register Description MSEBI Master from DMA 496
10.3.2.1 rMSEBIM_DMA_FIFOREAD_CS[n]_N — DMA Receive FIFO (64 KB) (n = 0..1) 496
10.3.2.2 rMSEBIM_DMA_FIFOWRITE_CS[n]_N — DMA Transmit FIFO (64 KB) (n = 0..1) 497
10.3.3 Register Description MSEBI Slave from CPU 498
10.3.3.1 rMSEBIS_CYCLESIZE_CS[n]_N — Chip Select CycleSize Register (n = 0..3) 498
10.3.3.2 rMSEBIS_SETUPHOLD_CS[n]_N — Chip Select SetupHold Register (n = 0..3) 500
10.3.3.3 rMSEBIS_MMU_ADDR_CS[n]_N — MMU Base Address Register (n = 0..3) 501
10.3.3.4 rMSEBIS_MMU_ADDR_MASK_CS[n]_N — MMU Address Mask Register (n = 0..3) 502

10.3.3.5 rMSEBIS_DMATX_REQ_CS[n]_N — DMA Transmit Request Register (n = 0..1) 503
10.3.3.6 rMSEBIS_DMARX_REQ_CS[n]_N — DMA Receive Request Register (n = 0..1) 504
10.3.3.7 rMSEBIS_DMATDLR_CS[n]_N — DMA Transmit Data Level Register (n = 0..1) 505
10.3.3.8 rMSEBIS_DMARDLR_CS[n]_N — DMA Receive Data Level Register (n = 0..1) 507
10.3.3.9 rMSEBIS_CONFIG_CS[n]_N — Chip Select Config Register (n = 0..3) 508
10.3.3.10 rMSEBIS_CONFIG — Common Config Register 512
10.3.3.11 rMSEBIS_STATUS_INT0 — Interrupt Status Register 515
10.3.3.12 rMSEBIS_STATUS_INT1 — Masked Interrupt Status Register 516
10.3.3.13 rMSEBIS_MASK_INT — Interrupt Mask Register 517
10.3.3.14 rMSEBIS_CLR_INT — Interrupt Clear Register 518
10.3.3.15 rMSEBIS_EOB_ADDR — End Of Block Address Register 519
10.3.4 Register Description MSEBI Slave from MSEBI 520
10.3.4.1 rMSEBIS_INT — Slave Interrupt Register 520
10.3.4.2 rMSEBIS_STATUS — Slave Status Register 522
10.3.4.3 rMSEBIS_ID_CS[n]_N — Slave ID Register (n = 0..3) 524
10.4 Operation 525
10.4.1 AHB Interface 525
10.4.1.1 AHB Slave Interface 525
10.4.1.2 AHB Master Interface (MSEBI Slave only) 525
10.4.2 Use Case Device Connection 526
10.4.2.1 One Device, Mode32, Synchronous 527
10.4.2.2 One Device, Mode16, Synchronous 528
10.4.2.3 One Device, Mode8, Synchronous 529
10.4.2.4 Three Devices, Mode8/16/32, Synchronous 530
10.4.2.5 Three Devices, Mode8/16/32, Asynchronous 531
10.4.2.6 Three Devices, Mode8/16/32, Mixed Synchronous and Asynchronous 532
10.4.2.7 One Device, Mode8, Asynchronous, ALE in Parallel Mode 533
10.4.3 Main Principle of Phase ADDRESS CONTROL and DATA 534
10.4.3.1 Address Latch Phase ALE (ADDRESS) 534
10.4.3.2 Control Latch Phase CLE (CONTROL) 540
10.4.3.3 Data Phase SETUP + VALID + HOLD (DATA) 540
10.4.4 MSEBI Timing 543
10.4.4.1 Asynchronous Mode, One ALE 543
10.4.4.2 Asynchronous Mode, No ALE MSEBI Master Only 548
10.4.4.3 Asynchronous Mode, Two ALE 550
10.4.4.4 Synchronous Mode, No Burst, One ALE 555
10.4.4.5 Synchronous Mode, No Burst, No ALE 562
10.4.4.6 Synchronous Mode, No Burst, Multiple ALE 564
10.4.4.7 Synchronous Mode, Burst, One ALE 568
10.4.4.8 Synchronous Mode, Burst, No ALE 573
10.4.5 MSEBI Interrupt 575
10.4.5.1 MSEBI Interrupt: Overview 575
10.4.5.2 MSEBI Interrupt: End of Block Detection by the Master 576
10.4.5.3 MSEBI Interrupt: End of Block Detection by the Slave 580
10.4.6 MSEBI Master Mode 583
10.4.6.1 Master Mode Overview 583
10.4.6.2 MSEBI Master: Burst Mode 584
10.4.6.3 MSEBI Master: DMA Control 595

10.4.7 MSEBI Slave Mode 605
10.4.7.1 Slave Mode Overview 605
10.4.7.2 MSEBI Slave: Burst Mode 607
10.4.7.3 MSEBI Slave: Detection of Request Initiator 619
10.4.7.4 MSEBI Slave: Register Access by Master 619
10.4.7.5 MSEBI Slave: Chip select Configuration Status 620
10.4.7.6 MSEBI Slave: Addressing Mode 621
10.4.7.7 MSEBI Slave: Write Protect 622
10.4.7.8 MSEBI Slave: Configuration Registers & Synchronization 622
10.5 Usage Notes 623

Section 1 UART

1.1 Overview

The PG0 (Peripheral Group 0) Subsystem and PG1 (Peripheral Group 1) Subsystem of RZ/N1 provide a total of 8 blocks of UART.

Each UART has the same features except DMA capability:

- UART f (Full) in PG1 : UART4, UART5, UART6, UART7, UART8
- UART r (Reduce) in PG0 : UART1, UART2, UART3

Each UART is configured with following features:

- Functionality based on the 16550 UART, as follows:
  - Programmable character properties, such as number of data bits per character (5 to 8), optional parity bit (with odd or even select) and number of stop bits (1, 1.5 or 2)
  - Generation and detection of line breaks
  - Prioritized interrupt identification
- Separate 16×8 (16 × 8-bit width) transmit and 16×8 receive FIFOs
- RS485 & MODBUS® enhanced features
- Programmable FIFO enable/disable
- There are 2 possible sources for the UART clock (UART_SCLK)
  1. Programmable frequency, 7.81 MHz to 83.33 MHz from MAIN PLL via a programmable integer divider
  2. Fixed 48 MHz from USBPLL

  The programmable integer divider which belongs to PG0 is shared by UART1..3, and the one which belongs to PG1 is shared by UART4..8.
- Programmable baud rate generator, up to UART_SCLK/16
- False start bit detection
- Programmable hardware flow control
- Shadow registers to reduce Software overhead and also include a Software programmable reset
- Auto Flow Control mode as specified in the 16750 standard
- Transmit Holding Register Empty (THRE) interrupt mode
- Busy functionality
- Loopback mode that enables greater testing of Modem Control and Auto Flow Control features
- Additional FIFO status registers
- Modem and status lines are independently controlled
- Two Receiver Time-Out provide support in handling the interframe time in MODBUS link
- Two Transceiver Time-Out provide support in handling the interframe time in MODBUS link

- Half-Duplex Management on “Two-Wire” interface by signal “DE (Data Enable)”
- Supports TXD, RXD, CTS_N, RTS_N, DTR_N, DSR_N, DCD_N, RI_N (multiplexed on GPIO pins)

UART Full (UART4..8) has the additional features:

- DMA Coupling
  - Peripheral flow controller mode
  - 2 DMA channels available (one for transmit, one for receive)

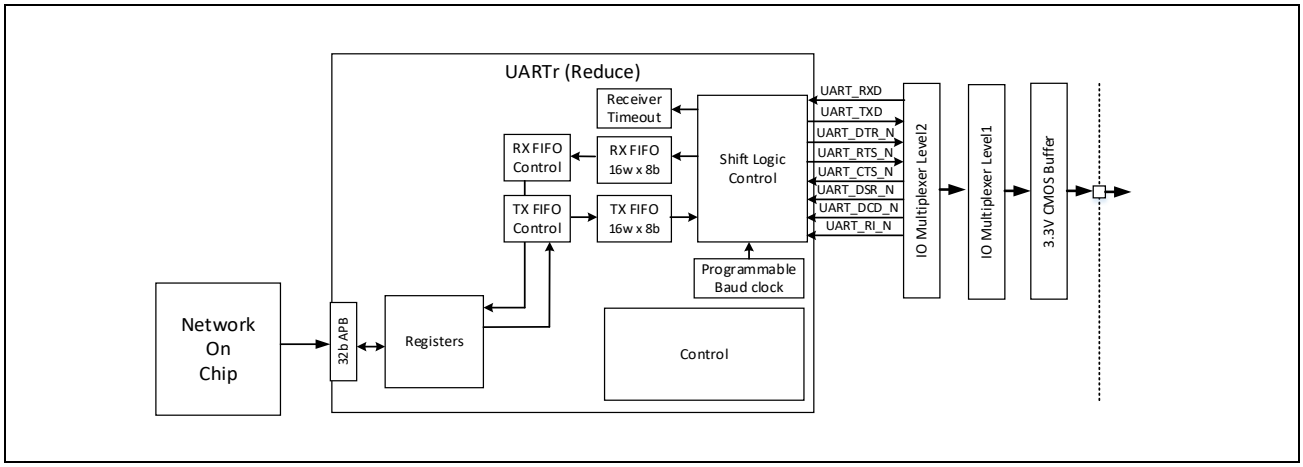

Figure 1.1 UART Reduce Synoptic (UART1..3)

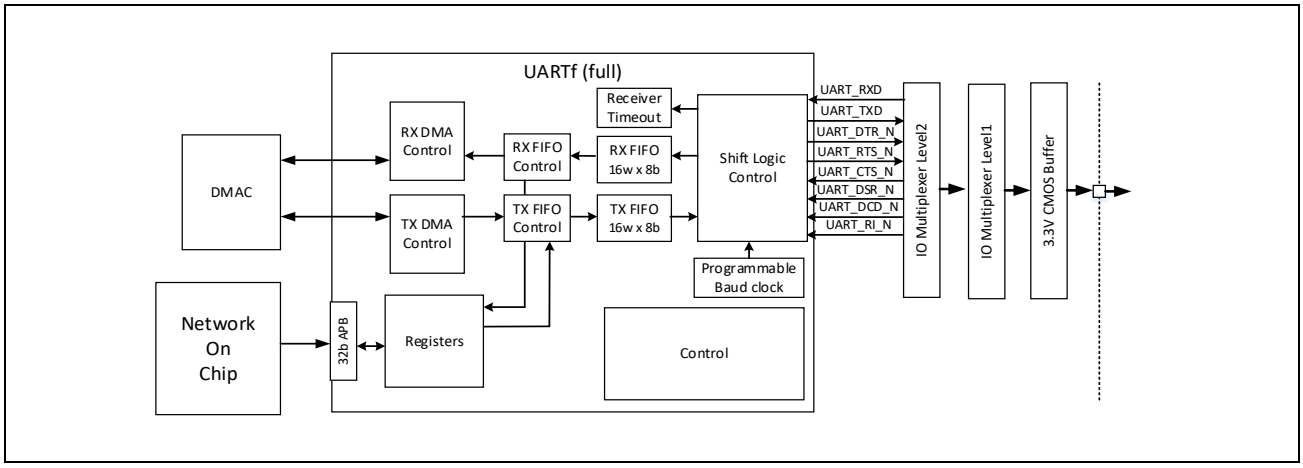

Figure 1.2 UART Full Synoptic (UART4..8)

1.2 Signal Interfaces

Signal Name	Input Output	Description
Clock		
UART[m]_PCLK	Input	Internal bus clock (APB)
UART[m]_SCLK	Input	Serial reference clock
Interrupt		
UART[m]_Int	Output	Level sensitive interrupt output, Active High
External Signal		
UART[m]_RXD	Input	Receive data
UART[m]_TXD	Output	Transmit data
UART[m]_CTS_N	Input	Clear to Send Modem Status
UART[m]_DSR_N	Input	Data Set Ready Modem Status
UART[m]_DCD_N	Input	Data Carrier Detect Modem Status
UART[m]_RI_N	Input	Ring Indicator Modem Status
UART[m]_DTR_N	Output	Modem Control Data Terminal Ready
UART[m]_RTS_N	Output	Modem Control Request to Send in Full-Duplex Mode Transmit Data Enable in Half-Duplex Mode

Note: m = 1..8.
Index removed style is mainly used in this chapter.
Ex) UART_PCLK

1.3 Register Map

1.3.1 Register Map UART 1

Table 1.1 Register Map UART 1

Address	Register Symbol	Register Name
4006 0000h	(bUart_DLAB = 0) rUart_RBR_THR	Receive Buffer/Transmit Holding Register
	(bUart_DLAB = 1) rUart_DLL	Divisor Latch (Low)
4006 0004h	(bUart_DLAB = 0) rUart_IER	Interrupt Enable Register
	(bUart_DLAB = 1) rUart_DLH	Divisor Latch (High)
4006 0008h	(when written) rUart_FCR	FIFO Control Register
	(when read) rUart_IIR	Interrupt Identification Register
4006 000Ch	rUart_LCR	Line Control Register
4006 0010h	rUart_MCR	Modem Control Register
4006 0014h	rUart_LSR	Line Status Register
4006 0018h	rUart_MSR	Modem Status Register
4006 001Ch	rUart_SCR	Scratchpad Register
4006 0030h	rUart_SRBR_STHR	Shadow Receive Buffer/Transmit Holding Register*1
4006 0070h	rUart_FAR	FIFO Access Register
4006 0074h	rUart_TFR	Transmit FIFO Read
4006 0078h	rUart_RFW	Receive FIFO Write
4006 007Ch	rUart_USR	UART Status Register
4006 0080h	rUart_TFL	Transmit FIFO Level
4006 0084h	rUart_RFL	Receive FIFO Level
4006 0088h	rUart_SRR	Software Reset Register
4006 008Ch	rUart_SRTS	Shadow Request to Send
4006 0090h	rUart_SBCR	Shadow Break Control Register
4006 0098h	rUart_SFE	Shadow FIFO Enable
4006 009Ch	rUart_SRT	Shadow RCVR Trigger
4006 00A0h	rUart_STET	Shadow TX Empty Trigger
4006 00A4h	rUart_HTX	Halt TX
4006 0100h	rUart_TO	Time-Out Counter Configuration Register
4006 0104h	rUart_CTRLTO	Time-Out Control Register
4006 0108h	rUart_STATUSTO	Time-Out Counter Status Register

Note 1. This address is assigned from 4006 0030h to 4006 006Ch

1.3.2 Register Map UART 2

Table 1.2 Register Map UART 2

Address	Register Symbol	Register Name
4006 1000h	(bUart_DLAB = 0) rUart_RBR_THR	Receive Buffer/Transmit Holding Register
	(bUart_DLAB = 1) rUart_DLL	Divisor Latch (Low)
4006 1004h	(bUart_DLAB = 0) rUart_IER	Interrupt Enable Register
	(bUart_DLAB = 1) rUart_DLH	Divisor Latch (High)
4006 1008h	(when written) rUart_FCR	FIFO Control Register
	(when read) rUart_IIR	Interrupt Identification Register
4006 100Ch	rUart_LCR	Line Control Register
4006 1010h	rUart_MCR	Modem Control Register
4006 1014h	rUart_LSR	Line Status Register
4006 1018h	rUart_MSR	Modem Status Register
4006 101Ch	rUart_SCR	Scratchpad Register
4006 1030h	rUart_SRBR_STHR	Shadow Receive Buffer/Transmit Holding Register*1
4006 1070h	rUart_FAR	FIFO Access Register
4006 1074h	rUart_TFR	Transmit FIFO Read
4006 1078h	rUart_RFW	Receive FIFO Write
4006 107Ch	rUart_USR	UART Status Register
4006 1080h	rUart_TFL	Transmit FIFO Level
4006 1084h	rUart_RFL	Receive FIFO Level
4006 1088h	rUart_SRR	Software Reset Register
4006 108Ch	rUart_SRTS	Shadow Request to Send
4006 1090h	rUart_SBCR	Shadow Break Control Register
4006 1098h	rUart_SFE	Shadow FIFO Enable
4006 109Ch	rUart_SRT	Shadow RCVR Trigger
4006 10A0h	rUart_STET	Shadow TX Empty Trigger
4006 10A4h	rUart_HTX	Halt TX
4006 1100h	rUart_TO	Time-Out Counter Configuration Register
4006 1104h	rUart_CTRLTO	Time-Out Control Register
4006 1108h	rUart_STATUSTO	Time-Out Counter Status Register

Note 1. This address is assigned from 4006 1030h to 4006 106Ch

1.3.3 Register Map UART 3

Table 1.3 Register Map UART 3

Address	Register Symbol	Register Name
4006 2000h	(bUart_DLAB = 0) rUart_RBR_THR	Receive Buffer/Transmit Holding Register
	(bUart_DLAB = 1) rUart_DLL	Divisor Latch (Low)
4006 2004h	(bUart_DLAB = 0) rUart_IER	Interrupt Enable Register
	(bUart_DLAB = 1) rUart_DLH	Divisor Latch (High)
4006 2008h	(when written) rUart_FCR	FIFO Control Register
	(when read) rUart_IIR	Interrupt Identification Register
4006 200Ch	rUart_LCR	Line Control Register
4006 2010h	rUart_MCR	Modem Control Register
4006 2014h	rUart_LSR	Line Status Register
4006 2018h	rUart_MSR	Modem Status Register
4006 201Ch	rUart_SCR	Scratchpad Register
4006 2030h	rUart_SRBR_STHR	Shadow Receive Buffer/Transmit Holding Register*1
4006 2070h	rUart_FAR	FIFO Access Register
4006 2074h	rUart_TFR	Transmit FIFO Read
4006 2078h	rUart_RFW	Receive FIFO Write
4006 207Ch	rUart_USR	UART Status Register
4006 2080h	rUart_TFL	Transmit FIFO Level
4006 2084h	rUart_RFL	Receive FIFO Level
4006 2088h	rUart_SRR	Software Reset Register
4006 208Ch	rUart_SRTS	Shadow Request to Send
4006 2090h	rUart_SBCR	Shadow Break Control Register
4006 2098h	rUart_SFE	Shadow FIFO Enable
4006 209Ch	rUart_SRT	Shadow RCVR Trigger
4006 20A0h	rUart_STET	Shadow TX Empty Trigger
4006 20A4h	rUart_HTX	Halt TX
4006 2100h	rUart_TO	Time-Out Counter Configuration Register
4006 2104h	rUart_CTRLTO	Time-Out Control Register
4006 2108h	rUart_STATUSTO	Time-Out Counter Status Register

Note 1. This address is assigned from 4006 2030h to 4006 206Ch

1.3.4 Register Map UART 4

Table 1.4 Register Map UART 4

Address	Register Symbol	Register Name
5000 0000h	(bUart_DLAB = 0) rUart_RBR_THR	Receive Buffer/Transmit Holding Register
	(bUart_DLAB = 1) rUart_DLL	Divisor Latch (Low)
5000 0004h	(bUart_DLAB = 0) rUart_IER	Interrupt Enable Register
	(bUart_DLAB = 1) rUart_DLH	Divisor Latch (High)
5000 0008h	(when written) rUart_FCR	FIFO Control Register
	(when read) rUart_IIR	Interrupt Identification Register
5000 000Ch	rUart_LCR	Line Control Register
5000 0010h	rUart_MCR	Modem Control Register
5000 0014h	rUart_LSR	Line Status Register
5000 0018h	rUart_MSR	Modem Status Register
5000 001Ch	rUart_SCR	Scratchpad Register
5000 0030h	rUart_SRBR_STHR	Shadow Receive Buffer/Transmit Holding Register*1
5000 0070h	rUart_FAR	FIFO Access Register
5000 0074h	rUart_TFR	Transmit FIFO Read
5000 0078h	rUart_RFW	Receive FIFO Write
5000 007Ch	rUart_USR	UART Status Register
5000 0080h	rUart_TFL	Transmit FIFO Level
5000 0084h	rUart_RFL	Receive FIFO Level
5000 0088h	rUart_SRR	Software Reset Register
5000 008Ch	rUart_SRTS	Shadow Request to Send
5000 0090h	rUart_SBCR	Shadow Break Control Register
5000 0098h	rUart_SFE	Shadow FIFO Enable
5000 009Ch	rUart_SRT	Shadow RCVR Trigger
5000 00A0h	rUart_STET	Shadow TX Empty Trigger
5000 00A4h	rUart_HTX	Halt TX
5000 00A8h	rUart_DMASA	DMA Software Acknowledge
5000 0100h	rUart_TO	Time-Out Counter Configuration Register
5000 0104h	rUart_CTRLTO	Time-Out Control Register
5000 0108h	rUart_STATUSTO	Time-Out Counter Status Register
5000 010Ch	rUart_TDMACR	DMA Control Register in Transmit Mode
5000 0110h	rUart_RDMACR	DMA Control Register in Receive Mode

Note 1. This address is assigned from 5000 0030h to 5000 006Ch

1.3.5 Register Map UART 5

Table 1.5 Register Map UART 5

Address	Register Symbol	Register Name
5000 1000h	(bUart_DLAB = 0) rUart_RBR_THR	Receive Buffer/Transmit Holding Register
	(bUart_DLAB = 1) rUart_DLL	Divisor Latch (Low)
5000 1004h	(bUart_DLAB = 0) rUart_IER	Interrupt Enable Register
	(bUart_DLAB = 1) rUart_DLH	Divisor Latch (High)
5000 1008h	(when written) rUart_FCR	FIFO Control Register
	(when read) rUart_IIR	Interrupt Identification Register
5000 100Ch	rUart_LCR	Line Control Register
5000 1010h	rUart_MCR	Modem Control Register
5000 1014h	rUart_LSR	Line Status Register
5000 1018h	rUart_MSR	Modem Status Register
5000 101Ch	rUart_SCR	Scratchpad Register
5000 1030h	rUart_SRBR_STHR	Shadow Receive Buffer/Transmit Holding Register*1
5000 1070h	rUart_FAR	FIFO Access Register
5000 1074h	rUart_TFR	Transmit FIFO Read
5000 1078h	rUart_RFW	Receive FIFO Write
5000 107Ch	rUart_USR	UART Status Register
5000 1080h	rUart_TFL	Transmit FIFO Level
5000 1084h	rUart_RFL	Receive FIFO Level
5000 1088h	rUart_SRR	Software Reset Register
5000 108Ch	rUart_SRTS	Shadow Request to Send
5000 1090h	rUart_SBCR	Shadow Break Control Register
5000 1098h	rUart_SFE	Shadow FIFO Enable
5000 109Ch	rUart_SRT	Shadow RCVR Trigger
5000 10A0h	rUart_STET	Shadow TX Empty Trigger
5000 10A4h	rUart_HTX	Halt TX
5000 10A8h	rUart_DMASA	DMA Software Acknowledge
5000 1100h	rUart_TO	Time-Out Counter Configuration Register
5000 1104h	rUart_CTRLTO	Time-Out Control Register
5000 1108h	rUart_STATUSTO	Time-Out Counter Status Register
5000 110Ch	rUart_TDMACR	DMA Control Register in Transmit Mode
5000 1110h	rUart_RDMACR	DMA Control Register in Receive Mode

Note 1. This address is assigned from 5000 1030h to 5000 106Ch

1.3.6 Register Map UART 6

Table 1.6 Register Map UART 6

Address	Register Symbol	Register Name
5000 2000h	(bUart_DLAB = 0) rUart_RBR_THR	Receive Buffer/Transmit Holding Register
	(bUart_DLAB = 1) rUart_DLL	Divisor Latch (Low)
5000 2004h	(bUart_DLAB = 0) rUart_IER	Interrupt Enable Register
	(bUart_DLAB = 1) rUart_DLH	Divisor Latch (High)
5000 2008h	(when written) rUart_FCR	FIFO Control Register
	(when read) rUart_IIR	Interrupt Identification Register
5000 200Ch	rUart_LCR	Line Control Register
5000 2010h	rUart_MCR	Modem Control Register
5000 2014h	rUart_LSR	Line Status Register
5000 2018h	rUart_MSR	Modem Status Register
5000 201Ch	rUart_SCR	Scratchpad Register
5000 2030h	rUart_SRBR_STHR	Shadow Receive Buffer/Transmit Holding Register*1
5000 2070h	rUart_FAR	FIFO Access Register
5000 2074h	rUart_TFR	Transmit FIFO Read
5000 2078h	rUart_RFW	Receive FIFO Write
5000 207Ch	rUart_USR	UART Status Register
5000 2080h	rUart_TFL	Transmit FIFO Level
5000 2084h	rUart_RFL	Receive FIFO Level
5000 2088h	rUart_SRR	Software Reset Register
5000 208Ch	rUart_SRTS	Shadow Request to Send
5000 2090h	rUart_SBCR	Shadow Break Control Register
5000 2098h	rUart_SFE	Shadow FIFO Enable
5000 209Ch	rUart_SRT	Shadow RCVR Trigger
5000 20A0h	rUart_STET	Shadow TX Empty Trigger
5000 20A4h	rUart_HTX	Halt TX
5000 20A8h	rUart_DMASA	DMA Software Acknowledge
5000 2100h	rUart_TO	Time-Out Counter Configuration Register
5000 2104h	rUart_CTRLTO	Time-Out Control Register
5000 2108h	rUart_STATUSTO	Time-Out Counter Status Register
5000 210Ch	rUart_TDMACR	DMA Control Register in Transmit Mode
5000 2110h	rUart_RDMACR	DMA Control Register in Receive Mode

Note 1. This address is assigned from 5000 2030h to 5000 206Ch

1.3.7 Register Map UART 7

Table 1.7 Register Map UART 7

Address	Register Symbol	Register Name
5000 3000h	(bUart_DLAB = 0) rUart_RBR_THR	Receive Buffer/Transmit Holding Register
	(bUart_DLAB = 1) rUart_DLL	Divisor Latch (Low)
5000 3004h	(bUart_DLAB = 0) rUart_IER	Interrupt Enable Register
	(bUart_DLAB = 1) rUart_DLH	Divisor Latch (High)
5000 3008h	(when written) rUart_FCR	FIFO Control Register
	(when read) rUart_IIR	Interrupt Identification Register
5000 300Ch	rUart_LCR	Line Control Register
5000 3010h	rUart_MCR	Modem Control Register
5000 3014h	rUart_LSR	Line Status Register
5000 3018h	rUart_MSR	Modem Status Register
5000 301Ch	rUart_SCR	Scratchpad Register
5000 3030h	rUart_SRBR_STHR	Shadow Receive Buffer/Transmit Holding Register*1
5000 3070h	rUart_FAR	FIFO Access Register
5000 3074h	rUart_TFR	Transmit FIFO Read
5000 3078h	rUart_RFW	Receive FIFO Write
5000 307Ch	rUart_USR	UART Status Register
5000 3080h	rUart_TFL	Transmit FIFO Level
5000 3084h	rUart_RFL	Receive FIFO Level
5000 3088h	rUart_SRR	Software Reset Register
5000 308Ch	rUart_SRTS	Shadow Request to Send
5000 3090h	rUart_SBCR	Shadow Break Control Register
5000 3098h	rUart_SFE	Shadow FIFO Enable
5000 309Ch	rUart_SRT	Shadow RCVR Trigger
5000 30A0h	rUart_STET	Shadow TX Empty Trigger
5000 30A4h	rUart_HTX	Halt TX
5000 30A8h	rUart_DMASA	DMA Software Acknowledge
5000 3100h	rUart_TO	Time-Out Counter Configuration Register
5000 3104h	rUart_CTRLTO	Time-Out Control Register
5000 3108h	rUart_STATUSTO	Time-Out Counter Status Register
5000 310Ch	rUart_TDMACR	DMA Control Register in Transmit Mode
5000 3110h	rUart_RDMACR	DMA Control Register in Receive Mode

Note 1. This address is assigned from 5000 3030h to 5000 306Ch

1.3.8 Register Map UART 8

Table 1.8 Register Map UART 8

Address	Register Symbol	Register Name
5000 4000h	(bUart_DLAB = 0) rUart_RBR_THR	Receive Buffer/Transmit Holding Register
	(bUart_DLAB = 1) rUart_DLL	Divisor Latch (Low)
5000 4004h	(bUart_DLAB = 0) rUart_IER	Interrupt Enable Register
	(bUart_DLAB = 1) rUart_DLH	Divisor Latch (High)
5000 4008h	(when written) rUart_FCR	FIFO Control Register
	(when read) rUart_IIR	Interrupt Identification Register
5000 400Ch	rUart_LCR	Line Control Register
5000 4010h	rUart_MCR	Modem Control Register
5000 4014h	rUart_LSR	Line Status Register
5000 4018h	rUart_MSR	Modem Status Register
5000 401Ch	rUart_SCR	Scratchpad Register
5000 4030h	rUart_SRBR_STHR	Shadow Receive Buffer/Transmit Holding Register*1
5000 4070h	rUart_FAR	FIFO Access Register
5000 4074h	rUart_TFR	Transmit FIFO Read
5000 4078h	rUart_RFW	Receive FIFO Write
5000 407Ch	rUart_USR	UART Status Register
5000 4080h	rUart_TFL	Transmit FIFO Level
5000 4084h	rUart_RFL	Receive FIFO Level
5000 4088h	rUart_SRR	Software Reset Register
5000 408Ch	rUart_SRTS	Shadow Request to Send
5000 4090h	rUart_SBCR	Shadow Break Control Register
5000 4098h	rUart_SFE	Shadow FIFO Enable
5000 409Ch	rUart_SRT	Shadow RCVR Trigger
5000 40A0h	rUart_STET	Shadow TX Empty Trigger
5000 40A4h	rUart_HTX	Halt TX
5000 40A8h	rUart_DMASA	DMA Software Acknowledge
5000 4100h	rUart_TO	Time-Out Counter Configuration Register
5000 4104h	rUart_CTRLTO	Time-Out Control Register
5000 4108h	rUart_STATUSTO	Time-Out Counter Status Register
5000 410Ch	rUart_TDMACR	DMA Control Register in Transmit Mode
5000 4110h	rUart_RDMACR	DMA Control Register in Receive Mode

Note 1. This address is assigned from 5000 4030h to 5000 406Ch

1.4 Register Description

1.4.1 rUart_DLL — Divisor Latch (Low)

- Dependencies: bUart_DLAB bit = 1

Address: 4006 0000h (UART1)
4006 1000h (UART2)
4006 2000h (UART3)
5000 0000h (UART4)
5000 1000h (UART5)
5000 2000h (UART6)
5000 3000h (UART7)
5000 4000h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_DLL							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.9 rUart_DLL Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	bUart_DLL	Lower 8 bits of a 16-bit. Divisor Latch register that contains the baud rate divisor for the UART. This register may only be accessed when the bUart_DLAB bit is set (rUart_LCR register) and the UART is not busy, bUart_BUSY bit is zero (rUart_USR register). The baud clock is equal to UART_SCLK frequency divided by sixteen times the value of the baud rate divisor, as follows: baud clock = UART_SCLK / (16 × baud rate divisor). ***Note*)** If the baud rate divisor (bUart_DLL and bUart_DLH) is set to zero, the baud clock is disabled and no serial communications occur. ***Caution*)** Also, once the bUart_DLL or bUart_DLH is set, at least 8 clock cycles of the slowest clock should be allowed to pass before transmitting or receiving data.	R/W

1.4.2 rUart_DLH — Divisor Latch (High)

- Dependencies: bUart_DLAB bit = 1

Address: 4006 0004h (UART1)
4006 1004h (UART2)
4006 2004h (UART3)
5000 0004h (UART4)
5000 1004h (UART5)
5000 2004h (UART6)
5000 3004h (UART7)
5000 4004h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_DLH							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.10 rUart_DLH Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	bUart_DLH	Upper 8 bits of a 16-bit. Divisor Latch register that contains the baud rate divisor for the UART. This register may only be accessed when the bUart_DLAB bit is set (rUart_LCR register) and the UART is not busy, bUart_BUSY bit is zero (rUart_USR register). Refer to **Section 1.4.1, rUart_DLL — Divisor Latch (Low)** for a detailed description of the baud clock.	R/W

1.4.3 rUart_IIR — Interrupt Identification Register

- When these addresses read

Address: 4006 0008h (UART1)
4006 1008h (UART2)
4006 2008h (UART3)
5000 0008h (UART4)
5000 1008h (UART5)
5000 2008h (UART6)
5000 3008h (UART7)
5000 4008h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_FIFOSE		—	—	bUart_IID			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 1.11 rUart_IIR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7, b6	bUart_FIFOSE	FIFOs Enabled. This is used to indicate whether the FIFOs are enabled or disabled. 2'b00 = disabled 2'b11 = enabled	R
b5, b4	Reserved		R
b3 to b0	bUart_IID	Interrupt ID. This indicates the highest priority pending interrupt which can be one of the following types: 4'b0000 = modem status 4'b0001 = no interrupt pending 4'b0010 = THR empty 4'b0100 = received data available 4'b0101 = receiver time out 4'b0110 = receiver line status 4'b0111 = busy detect 4'b1100 = character timeout The interrupt priorities are split into six levels that are detailed in See **Table 1.41, Interrupt Control Functions**. ***Note*)** Bit 3 of bUart_IID indicates an interrupt can only occur when the FIFOs are enabled and used to distinguish a Character Timeout condition interrupt.	R

1.4.4 rUart_RBR_THR — Receive Buffer/Transmit Holding Register

- Dependencies: bUart_DLAB bit = 0

Address: 4006 0000h (UART1)
4006 1000h (UART2)
4006 2000h (UART3)
5000 0000h (UART4)
5000 1000h (UART5)
5000 2000h (UART6)
5000 3000h (UART7)
5000 4000h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_RBR_THR							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.12 rUart_RBR_THR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	bUart_RBR_THR	When reading this register — Receive Buffer Register (rUart_RBR) • Data byte received on the serial input port UART_RXD. The data in this register is valid only if the Data Ready (bUart_DR) bit in the Line Status Register (rUart_LSR) is set. • If FIFOs are disabled (bUart_FIFOE = 0), the data in the rUart_RBR must be read before the next data arrives, otherwise it is overwritten, resulting in an over-run error. • If FIFOs are enabled (bUart_FIFOE = 1), this register accesses the head of the receive FIFO. If the receive FIFO is full and this register is not read before the next data character arrives, then the data already in the FIFO is preserved, but any incoming data are lost and an overrun error occurs. When writing to this register — Transmit Holding Register (rUart_THR) • Data to be transmitted on the serial output port UART_TXD. • If FIFOs are disabled (bUart_FIFOE = 0) and bUart_THRE is set, writing a single character to the rUart_THR clears the bUart_THRE. Any additional writes to the rUart_THR before the bUart_THRE is set again causes the rUart_THR data to be overwritten. • If FIFOs are enabled (bUart_FIFOE = 1) and bUart_THRE is set, 16 number of characters of data may be written to the rUart_THR before the FIFO is full. Any attempt to write data when the FIFO is full results in the write data being lost. See **Section 1.5.1.8, Programmable THRE interrupt**.	R/W

1.4.5 rUart_IER — Interrupt Enable Register

- Dependencies: bUart_DLAB bit = 0

Address: 4006 0004h (UART1)
4006 1004h (UART2)
4006 2004h (UART3)
5000 0004h (UART4)
5000 1004h (UART5)
5000 2004h (UART6)
5000 3004h (UART7)
5000 4004h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bUart_ETIMEOUT3	bUart_ETIMEOUT2	bUart_ETIMEOUT1	bUart_ETIMEOUT0	bUart_PTIME	—	—	—	bUart_EDSSI	bUart_ELSI	bUart_ETBEI	bUart_ERBFI
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.13 rUart_IER Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b12	Reserved		R
b11	bUart_ETIMEOUT3	Enable Receiver or Transceiver Time-Out n (with n = 0..3) interrupt. Enable Transceiver Time-Out n with n = 3 This is used to enable/disable the generation of Receiver (with n = 0..1) or Transceiver Time-Out n (with n = 2..3) interrupt. This is the sixth highest priority interrupt. For each Time-Out n with n = 0..3, we have: 1'b0 = disabled 1'b1 = enabled See **Section 1.5.1.10(1), Receiver Time-Out**. See **Section 1.5.1.10(2), Transceiver Time-Out**.	R/W
b10	bUart_ETIMEOUT2	Enable Transceiver Time-Out n with n = 2 See description detailed above	R/W
b9	bUart_ETIMEOUT1	Enable Receiver Time-Out n with n = 1 See description detailed above	R/W
b8	bUart_ETIMEOUT0	Enable Receiver Time-Out n with n = 0 See description detailed above	R/W
b7	bUart_PTIME	Programmable THRE Interrupt Mode Enable. This is used to enable/disable the generation of THRE Interrupt. 1'b0 = disabled 1'b1 = enabled See **Section 1.5.1.8, Programmable THRE interrupt**.	R/W
b6 to b4	Reserved		R
b3	bUart_EDSSI	Enable Modem Status Interrupt. This is used to enable/disable the generation of Modem Status Interrupt. This is the fourth highest priority interrupt. 1'b0 = disabled 1'b1 = enabled	R/W

Table 1.13 rUart_IER Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b2	bUart_ELSI	Enable Receiver Line Status Interrupt. This is used to enable/disable the generation of Receiver Line Status Interrupt. This is the highest priority interrupt. 1'b0 = disabled 1'b1 = enabled	R/W
b1	bUart_ETBEI	Enable Transmit Holding Register Empty Interrupt. This is used to enable/disable the generation of Transmit Holding Register Empty Interrupt. This is the third highest priority interrupt. 1'b0 = disabled 1'b1 = enabled	R/W
b0	bUart_ERBFI	Enable Received Data Available Interrupt. This is used to enable/disable the generation of Received Data Available Interrupt and the Character Timeout Interrupt (if FIFOs enabled). This is the second highest priority interrupt. 1'b0 = disabled 1'b1 = enabled	R/W

1.4.6 rUart_FCR — FIFO Control Register

- When these addresses written

Address: 4006 0008h (UART1)
4006 1008h (UART2)
4006 2008h (UART3)
5000 0008h (UART4)
5000 1008h (UART5)
5000 2008h (UART6)
5000 3008h (UART7)
5000 4008h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_RCVR		bUart_TET		—	bUart_XFIFOR	bUart_RFIFOR	bUart_FIFOE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.14 rUart_FCR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7, b6	bUart_RCVR	Receive FIFOs trigger. This is used to select the trigger level in the receive FIFO at which the Received Data Available Interrupt is generated. The following trigger levels are supported: 2'b00 = 1 character in the FIFO 2'b01 = FIFO 1/4 full 2'b10 = FIFO 1/2 full 2'b11 = FIFO 2 less than full	W
b5, b4	bUart_TET	Transmit FIFOs Empty trigger. This is used to select the empty threshold level at which the THRE Interrupts are generated when the mode is active. The following trigger levels are supported: 2'b00 = FIFO empty 2'b01 = 2 characters in the FIFO 2'b10 = FIFO 1/4 full 2'b11 = FIFO 1/2 full	W
b3	Reserved		R
b2	bUart_XFIFOR	Transmit FIFO Reset. This resets the control portion of the transmit FIFO and treats the FIFO as empty by writing 1b to this bit. ***Note*)** This bit is "self-clearing". It is not necessary to clear this bit.	W
b1	bUart_RFIFOR	Receive FIFO Reset. This resets the control portion of the receive FIFO and treats the FIFO as empty by writing 1b to this bit. ***Note*)** This bit is "self-clearing". It is not necessary to clear this bit.	W

Table 1.14 rUart_FCR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b0	bUart_FIFOE	FIFO Enable. This enables/disables the transmit and receive FIFOs. Whenever the value of this bit is changed both the transmit and receive controller portion of FIFOs is reset. 1'b0 = Disable 1'b1 = Enable	W

1.4.7 rUart_LCR — Line Control Register

Address: 4006 000Ch (UART1)
4006 100Ch (UART2)
4006 200Ch (UART3)
5000 000Ch (UART4)
5000 100Ch (UART5)
5000 200Ch (UART6)
5000 300Ch (UART7)
5000 400Ch (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_DLAB	bUart_BC	bUart_StickParity	bUart_EPS	bUart_PEN	bUart_STOP	bUart_DLS	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.15 rUart_LCR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7	bUart_DLAB	Divisor Latch Access Bit. Writeable only when UART is not busy (bUart_BUSY = 0). This bit is used to enable reading and writing of the baud rate divisor (bUart_DLL and bUart_DLH) to set the baud rate of the UART. This bit must be cleared after initial baud rate setup in order to access other registers. 1'b0 = Divisor Latch Access disable 1'b1 = Divisor Latch Access enable	R/W
b6	bUart_BC	Break Control Bit. This is used to cause a break condition to be transmitted to the receiving device. If set to one the serial output is forced to the spacing (logic 0) state. When not in Loopback Mode, as determined by bUart_LB, the UART_TXD line is forced low until the bUart_BC is cleared. When in Loopback Mode, the break condition is internally looped back to the receiver.	R/W
b5	bUart_StickParity	Stick Parity bit. Writeable only when UART is not busy (bUart_BUSY = 0). This bit is used to force parity value. • When bUart_PEN, bUart_EPS, and bUart_StickParity are set to 1, the parity bit is transmitted and checked as logic 0. • When bUart_PEN and bUart_StickParity are set to 1 and bUart_EPS is a logic 0, then parity bit is transmitted and checked as a logic 1. • When set to 0, Stick Parity is disabled.	R/W
b4	bUart_EPS	Even Parity Select. Writeable only when UART is not busy (bUart_BUSY = 0). This is used to select between even and odd parity, when parity is enabled (bUart_PEN = 1). • If set to one, an even number of logic 1s is transmitted or checked. • If set to zero, an odd number of logic 1s is transmitted or checked.	R/W
b3	bUart_PEN	Parity Enable. Writeable only when UART is not busy (bUart_BUSY = 0) This bit is used to enable and disable parity generation and detection in transmitted and received serial character respectively. 1'b0 = parity disabled 1'b1 = parity enabled	R/W

Table 1.15 rUart_LCR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b2	bUart_STOP	Number of stop bits. Writeable only when UART is not busy (bUart_BUSY = 0). This is used to select the number of stop bits per character that the peripheral transmits and receives. • If set to zero, one stop bit is transmitted in the serial data. • If set to one and the data bits are set to 5 (bUart_DLS = 0) one and a half stop bits is transmitted. Otherwise, two stop bits are transmitted. Note that regardless of the number of stop bits selected, the receiver checks only the first stop bit. 1'b0 = 1 stop bit 1'b1 = 1.5 stop bits when bUart_DLS is zero, else 2 stop bit ***Note*)** The STOP bit duration implemented by UART may appear longer due to the idle time inserted between characters for some configurations and baud rate divisor values in the transmit direction. For details on idle time between transmitted transfers, refer to **Section 1.5.1.5, Back to Back Character Stream Transmission**.	R/W
b1, b0	bUart_DLS	Data Length Select. Writeable only when UART is not busy (bUart_BUSY = 0). This is used to select the number of data bits per character that the peripheral transmits and receives. The number of bit that may be selected areas follows: 2'b00 = 5 bits 2'b01 = 6 bits 2'b10 = 7 bits 2'b11 = 8 bits	R/W

1.4.8 rUart_MCR — Modem Control Register

Address: 4006 0010h (UART1)
4006 1010h (UART2)
4006 2010h (UART3)
5000 0010h (UART4)
5000 1010h (UART5)
5000 2010h (UART6)
5000 3010h (UART7)
5000 4010h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	bUart_AFCE	bUart_LB	bUart_OUT2	bUart_OUT1	bUart_RTS	bUart_DTR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.16 rUart_MCR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b6	Reserved		R
b5	bUart_AFCE	Auto Flow Control Enable. When FIFOs are enabled (bUart_FIFOE bit is set) and the Auto Flow Control Enable (bUart_AFCE bit is set), Auto Flow Control features are enabled as described in **Section 1.5.1.7, Auto Flow Control**. 1'b0 = Auto Flow Control Mode disabled 1'b1 = Auto Flow Control Mode enabled	R/W
b4	bUart_LB	LoopBack Bit. This is used to put the UART into a diagnostic mode for test purposes. Data on the UART_TXD line is held high, while serial data output is looped back to the UART_RXD line, internally. In this mode, all the interrupts are fully functional. Also, in loopback mode, the modem control inputs (UART_DSR_N, UART_CTS_N, UART_RI_N, UART_DCD_N) are disconnected and the modem control outputs (UART_DTR_N, UART_RTS_N, UART_OUT1_N, UART_OUT2_N) are looped back to the inputs, internally. 1'b0 = Loop back mode disable 1'b1 = Loop back mode enable	R/W
b3	bUart_OUT2	This is used to directly control the user-designated Output2 (UART_OUT2_N) output. The value written to this location is inverted and driven out on UART_OUT2_N, that is: 1'b0 = UART_OUT2_N de-asserted (logic 1) 1'b1 = UART_OUT2_N asserted (logic 0) ***Caution***) This output pin is not connected on pinout. It uses in Loopback mode. ***Note***) In Loopback mode (bUart_LB = 1), the UART_OUT2_N output is held inactive high while the value of this location is internally looped back to an input.	R/W
b2	bUart_OUT1	This is used to directly control the user-designated Output1 (UART_OUT1_N) output. The value written to this location is inverted and driven out on UART_OUT1_N, that is: 1'b0 = UART_OUT1_N de-asserted (logic 1) 1'b1 = UART_OUT1_N asserted (logic 0) ***Caution***) This output pin is not connected on pinout. It uses in Loopback mode. ***Note***) In Loopback mode (bUart_LB = 1), the UART_OUT1_N output is held inactive high while the value of this location is internally looped back to an input.	R/W

Table 1.16 rUart_MCR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b1	bUart_RTS	Request to Send. This is used to directly control the Request to Send (UART_RTS_N) output. The Request to Send (UART_RTS_N) output is used to inform the modem or data set that the UART is ready to exchange data. When Auto RTS Flow Control is not enabled (bUart_AFCE = 0), the UART_RTS_N signal is set low by programming bUart_RTS to a high. In Auto Flow Control, (bUart_AFCE = 1) and FIFOs enable (bUart_FIFOE = 1), the UART_RTS_N output is controlled in the same way, but it is gated by the receive FIFO almost-full trigger, where “almost full” refers to two available slots in the FIFO (UART_RTS_N is inactive high when above the threshold). The UART_RTS_N signal is de-asserted when bUart_RTS is set low. See **Section 1.5.1.7, Auto Flow Control**. ***Note*)** In Loopback mode (bUart_LB = 1), the UART_RTS_N output is held inactive high while the value of this location is internally looped back to an input.	R/W
b0	bUart_DTR	Data Terminal Ready. This is used to directly control the Data Terminal Ready (UART_DTR_N) output. The value written to this location is inverted and driven out on UART_DTR_N, that is: 1’b0 = UART_DTR_N de-asserted (logic 1) 1’b1 = UART_DTR_N asserted (logic 0) The Data Terminal Ready output is used to inform the modem or data set that the UART is ready to establish communications. ***Note*)** In Loopback mode (bUart_LB = 1), the UART_DTR_N output is held inactive high while the value of this location is internally looped back to an input.	R/W

1.4.9 rUart_LSR — Line Status Register

Address: 4006 0014h (UART1)
4006 1014h (UART2)
4006 2014h (UART3)
5000 0014h (UART4)
5000 1014h (UART5)
5000 2014h (UART6)
5000 3014h (UART7)
5000 4014h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_RFE	bUart_TEMT	bUart_THRE	bUart_BI	bUart_FE	bUart_PE	bUart_OE	bUart_DR
Value after reset	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Table 1.17 rUart_LSR Register Contents (1/3)

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7	bUart_RFE	Receiver FIFO Error bit. This bit is only relevant if FIFOs are enabled (bUart_FIFOE = 1). This is used to indicate if there is at least one parity error, framing error, or break indication in the receive FIFO. 1'b0 = no error in Receive FIFO 1'b1 = error in Receive FIFO This bit is cleared when the rUart_LSR is read and the character with the error is at the top of the receive FIFO and there are no subsequent errors in the FIFO.	R
b6	bUart_TEMT	Transmitter Empty bit. If FIFOs enabled (bUart_FIFOE = 1), this bit is set whenever the Transmitter Shift Register and the Transmit FIFO are both empty. If FIFOs are disabled, this bit is set whenever the Transmit Holding Register (rUart_THR) and the Transmitter Shift Register are both empty.	R
b5	bUart_THRE	Transmit Holding Register Empty bit. If THRE mode is disabled (bUart_PTIME = 0) and regardless of FIFO's being enabled or not, this bit indicates that the rUart_THR or Transmit FIFO is empty. This bit is set whenever data is transferred from the rUart_THR or Transmit FIFO to the Transmitter Shift Register and no new data has been written to the rUart_THR or Transmit FIFO. This also causes a THRE Interrupt to occur, if the THRE Interrupt is Enabled (bUart_ETBEI). If THRE mode and FIFO are enabled (bUart_PTIME = 1 and bUart_FIFOE = 1 respectively), the functionality is switched to indicate the transmit FIFO is full, and no longer controls THRE interrupts, which are then controlled by the bUart_TET threshold setting. For more details, see **Section 1.5.1.8, Programmable THRE interrupt**.	R

Table 1.17 rUart_LSR Register Contents (2/3)

Bit Position	Bit Name	Function	R/W
b4	bUart_BI	Break Interrupt bit. This is used to indicate the detection of a break sequence on the serial input data. It is set whenever the serial input, UART_RXD, is held in a logic “0” state for longer than the sum of start time + data bits + parity + stop bits. A break condition on serial input causes one and only one character, consisting of all zeros, to be received by the UART. In the FIFO mode (bUart_FIFOE = 1), the character associated with the break condition is carried through the FIFO and is revealed when the character is at the top of the FIFO. Reading the rUart_LSR clears the bUart_BI bit. In the non-FIFO mode, the bUart_BI indication occurs immediately and persists until the rUart_LSR is read. ***Note*)** If a FIFO is full when a break condition is received, a FIFO overrun occurs. The break condition and all the information associated with it-parity and framing errors-is discarded any information that a break character was received is lost.	R
b3	bUart_FE	Framing Error bit. This is used to indicate the occurrence of a framing error in the receiver. A framing error occurs when the receiver does not detect a valid STOP bit in the received data. In the FIFO mode (bUart_FIFOE = 1), since the framing error is associated with a character received, it is revealed when the character with the framing error is at the top of the FIFO. When a framing error occurs, the UART tries to resynchronize. It does this by assuming that the error was due to the start bit of the next character and then continues receiving the other bit i.e. data, and/or parity and stop. It should be noted that the Framing Error bUart_FE bit is set if a break interrupt has occurred, as indicated by Break Interrupt bUart_BI bit. This happens because the break character implicitly generates a framing error by holding the UART_RXD input to logic 0 for longer than the duration of a character. 1’b0 = no framing error 1’b1 = framing error Reading the rUart_LSR clears the rUart_FE bit.	R
b2	bUart_PE	Parity Error bit. This is used to indicate the occurrence of a parity error in the receiver if the Parity Enable bUart_PEN bit is set. In the FIFO mode (bUart_FIFOE = 1), since the parity error is associated with a character received, it is revealed when the character with the parity error arrives at the top of the FIFO. It should be noted that the Parity Error bUart_PE bit is set if a break interrupt has occurred, as indicated by Break Interrupt bUart_BI bit and parity generation and detection are enabled (bUart_PEN=1) and the parity is set to odd (bUart_EPS = 0). 1’b0 = no parity error 1’b1 = parity error Reading the rUart_LSR clears the bUart_PE bit.	R
b1	bUart_OE	Overrun error bit. This is used to indicate the occurrence of an overrun error. This occurs if a new data character was received before the previous data was read. In the non-FIFO mode (bUart_FIFOE = 0), the bUart_OE bit is set when a new character arrives in the receiver before the previous character was read from the rUart_RBR. When this happens, the data in the rUart_RBR is overwritten. In the FIFO mode, an overrun error occurs when the FIFO is full and a new character arrives at the receiver. The data in the FIFO is retained and the data in the receive shift register is lost. 1’b0 = no overrun error 1’b1 = overrun error Reading the rUart_LSR clears the bUart_OE bit.	R

Table 1.17 rUart_LSR Register Contents (3/3)

Bit Position	Bit Name	Function	R/W
b0	bUart_DR	Data Ready bit. This is used to indicate that the receiver contains at least one character in the rUart_RBR or the receive FIFO. 1'b0 = no data ready 1'b1 = data ready This bit is cleared when the rUart_RBR is read in non-FIFO mode, or when the receive FIFO is empty, in FIFO mode.	R

1.4.10 rUart_MSR — Modem Status Register

Address: 4006 0018h (UART1)
4006 1018h (UART2)
4006 2018h (UART3)
5000 0018h (UART4)
5000 1018h (UART5)
5000 2018h (UART6)
5000 3018h (UART7)
5000 4018h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_DCD	bUart_RI	bUart_DSR	bUart_CTS	bUart_DDCD	bUart_TERI	bUart_DDSR	bUart_DCTS
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.18 rUart_MSR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7	bUart_DCD	Data Carrier Detect. This is used to indicate the current state of the modem control line UART_DCD_N. This bit is the complement of UART_DCD_N. When the Data Carrier Detect input (UART_DCD_N) is asserted, it is an indication that the carrier has been detected by the modem or data set. 1'b0 = UART_DCD_N input is de-asserted (logic 1) 1'b1 = UART_DCD_N input is asserted (logic 0) In Loopback Mode (bUart_LB = 1), bUart_DCD is the same as bUart_OUT2.	R
b6	bUart_RI	Ring Indicator. This is used to indicate the current state of the modem control line UART_RI_N. This bit is the complement of UART_RI_N. When the Ring Indicator input (UART_RI_N) is asserted, it is an indication that a telephone ringing signal has been received by the modem or data set. 1'b0 = UART_RI_N input is de-asserted (logic 1) 1'b1 = UART_RI_N input is asserted (logic 0) In Loopback Mode (bUart_LB = 1), bUart_RI is the same as bUart_OUT1.	R
b5	bUart_DSR	Data Set Ready. This is used to indicate the current state of the modem control line UART_DSR_N. This bit is the complement of UART_DSR_N. When the Data Set Ready input (UART_DSR_N) is asserted, it is an indication that the modem or data set is ready to establish communications with the UART. 1'b0 = UART_DSR_N input is de-asserted (logic 1) 1'b1 = UART_DSR_N input is asserted (logic 0) In Loopback Mode (bUart_LB = 1), bUart_DSR is the same as bUart_DTR.	R

Table 1.18 rUart_MSR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b4	bUart_CTS	Clear to Send. This is used to indicate the current state of the modem control line UART_CTS_N. This bit is the complement of UART_CTS_N. When the Clear to Send input (UART_CTS_N) is asserted, it is an indication that the modem or data set is ready to exchange data with the UART. 1'b0 = UART_CTS_N input is de-asserted (logic 1) 1'b1 = UART_CTS_N input is asserted (logic 0) In Loopback Mode (bUart_LB = 1), bUart_CTS is the same as bUart_RTS.	R
b3	bUart_DDCD	Delta Data Carrier Detect. This is used to indicate that the modem control line UART_DCD_N has changed since the last time the rUart_MSR was read. 1'b0 = no change on UART_DCD_N since last read of rUart_MSR 1'b1 = change on UART_DCD_N since last read of rUart_MSR Reading the rUart_MSR clears the bUart_DDCD bit. In Loopback Mode (bUart_LB = 1), bUart_DDCD reflects changes on bUart_OUT2. ***Note*)** If the bUart_DDCD bit is not set and the UART_DCD_N signal is asserted (low) and a reset occurs (Software or otherwise), then the bUart_DDCD bit is set when the reset is removed.	R
b2	bUart_TERI	Trailing Edge of Ring Indicator. This is used to indicate that a change on the input UART_RI_N (from an active-low to an inactive-high state) has occurred since the last time the rUart_MSR was read. 1'b0 = no change on UART_RI_N since last read of rUart_MSR 1'b1 = change on UART_RI_N since last read of rUart_MSR Reading the rUart_MSR clears the bUart_TERI bit. In Loopback Mode (bUart_LB = 1), bUart_TERI reflects when bUart_OUT1 has changed state from a high to a low.	R
b1	bUart_DDSR	Delta Data Set Ready. This is used to indicate that the modem control line UART_DSR_N has changed since the last time the rUart_MSR was read. 1'b0 = no change on UART_DSR_N since last read of rUart_MSR 1'b1 = change on UART_DSR_N since last read of rUart_MSR Reading the rUart_MSR clears the bUart_DDSR bit. In Loopback Mode (bUart_LB = 1), bUart_DDSR reflects changes on bUart_DTR. ***Note*)** If the bUart_DDSR bit is not set and the UART_DSR_N signal is asserted (low) and a reset occurs (Software or otherwise), then the bUart_DDSR bit is set when the reset is removed.	R
b0	bUart_DCTS	Delta Clear to Send. This is used to indicate that the modem control line UART_CTS_N has changed since the last time the rUart_MSR was read. 1'b0 = no change on UART_CTS_N since last read of rUart_MSR 1'b1 = change on UART_CTS_N since last read of rUart_MSR Reading the rUart_MSR clears the bUart_DCTS bit. In Loopback Mode (bUart_LB = 1), bUart_DCTS reflects changes on bUart_RTS. ***Note*)** If the bUart_DCTS bit is not set and the UART_CTS_N signal is asserted (low) and a reset occurs (Software or otherwise), then the bUart_DCTS bit is set when the reset is removed.	R

1.4.11 rUart_SCR — Scratchpad Register

Address: 4006 001Ch (UART1)
4006 101Ch (UART2)
4006 201Ch (UART3)
5000 001Ch (UART4)
5000 101Ch (UART5)
5000 201Ch (UART6)
5000 301Ch (UART7)
5000 401Ch (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_SCR							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.19 rUart_SCR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	bUart_SCR	This register is for programmers to use as a temporary storage space. It has no defined purpose in the UART.	R/W

1.4.12 rUart_SRBR_STHR — Shadow Receive Buffer/Transmit Holding Register

- Dependencies: bUart_DLAB bit =0

Address: 4006 0030h (UART1)
4006 1030h (UART2)
4006 2030h (UART3)
5000 0030h (UART4)
5000 1030h (UART5)
5000 2030h (UART6)
5000 3030h (UART7)
5000 4030h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_SRBR_STHR							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.20 rUart_SRBR_STHR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	bUart_SRBR_STHR	When reading this register - Shadow Receive Buffer Register (rUart_SRBR) • This is a shadow register for the rUart_RBR and has been allocated sixteen 32-bit locations so as to accommodate burst accesses from the master. This register contains the data byte received on the serial input port UART_RXD. The data in this register is valid only if the Data Ready (bUart_DR bit) in the Line Status Register (rUart_LSR) is set. • If FIFOs are disabled (bUart_FIFOE = 0), the data in the rUart_RBR must be read before the next data arrives, otherwise it is overwritten, resulting in an overrun error. • If FIFOs are enabled (bUart_FIFOE = 1), this register accesses the head of the receive FIFO. If the receive FIFO is full and this register is not read before the next data character arrives, then the data already in the FIFO are preserved, but any incoming data is lost. An overrun error also occurs. When writing to this register - Shadow Transmit Holding Register (rUart_STHR) • This is a shadow register for the rUart_THR and has been allocated sixteen 32-bit locations so as to accommodate burst accesses from the master. This register contains data to be transmitted on the serial output port UART_TXD. Data should only be written to the rUart_THR when the THR Empty (bUart_THRE) bit in rUart_LSR register is set. • If FIFOs are disabled (bUart_FIFOE = 0) and bUart_THRE is set, writing a single character to the rUart_THR clears the bUart_THRE. Any additional writes to the rUart_THR before the bUart_THRE is set again causes the rUart_THR data to be overwritten. • If FIFOs are enabled (bUart_FIFOE = 1) and bUart_THRE is set, 16 number of characters of data may be written to the rUart_THR before the FIFO is full. Any attempt to write data when the FIFO is full results in the write data being lost.	R/W

1.4.13 rUart_FAR — FIFO Access Register

Address: 4006 0070h (UART1)
4006 1070h (UART2)
4006 2070h (UART3)
5000 0070h (UART4)
5000 1070h (UART5)
5000 2070h (UART6)
5000 3070h (UART7)
5000 4070h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_FAR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.21 rUart_FAR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	bUart_FAR	FIFO access registers. This register is use to enable a FIFO access mode for testing, so that the receive FIFO can be written by the master and the transmit FIFO can be read by the master when FIFOs are enabled. When FIFOs are not enabled it allows the rUart_RBR to be written by the master and the rUart_THR to be read by the master. 1'b0 = FIFO access mode disabled 1'b1 = FIFO access mode enabled ***Note*)** When the FIFO access mode is enabled/disabled, the control portion of the receive FIFO and transmit FIFO is reset and the FIFOs are treated as empty.	R/W

1.4.14 rUart_TFR — Transmit FIFO Read

Address: 4006 0074h (UART1)
4006 1074h (UART2)
4006 2074h (UART3)
5000 0074h (UART4)
5000 1074h (UART5)
5000 2074h (UART6)
5000 3074h (UART7)
5000 4074h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bUart_TFR							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.22 rUart_TFR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	bUart_TFR	Transmit FIFO Read. These bits are only valid when FIFO access mode is enabled (bUart_FAR = 1). When FIFOs are enabled (bUart_FIFOE), reading this register gives the data at the top of the transmit FIFO. Each consecutive read pops the transmit FIFO and gives the next data value that is currently at the top of the FIFO. When FIFOs are not enabled, reading this register gives the data in the rUart_THR register.	R

1.4.15 rUart_RFW — Receive FIFO Write

Address: 4006 0078h (UART1)
4006 1078h (UART2)
4006 2078h (UART3)
5000 0078h (UART4)
5000 1078h (UART5)
5000 2078h (UART6)
5000 3078h (UART7)
5000 4078h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	bUart_RFFE	bUart_RFPE	bUart_RFWD							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.23 rUart_RFW Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b10	Reserved		R
b9	bUart_RFFE	Receive FIFO Framing Error. This bit is only valid when FIFO access mode is enabled (bUart_FAR = 1). When FIFOs are enabled, this bit is used to write framing error detection information to the receive FIFO. When FIFOs are not enabled, this bit is used to write framing error detection information to the rUart_RBR register. ***Note*)** This bit also active a Break Condition in the Receive FIFO.	W
b8	bUart_RFPE	Receive FIFO Parity Error. This bit is only valid when FIFO access mode is enabled (bUart_FAR = 1). When FIFOs are enabled, this bit is used to write parity error detection information to the receive FIFO. When FIFOs are not enabled, this bit is used to write parity error detection information to the rUart_RBR register.	W
b7 to b0	bUart_RFWD	Receive FIFO Write Data. These bits are only valid when FIFO access mode is enabled (bUart_FAR = 1). When FIFOs are enabled, the data that is written to the bUart_RFWD is pushed into the receive FIFO. Each consecutive write pushes the new data to the next write location in the receive FIFO. When FIFOs are not enabled, the data that is written to the bUart_RFWD is pushed into the rUart_RBR register.	W

1.4.16 rUart_USR — UART Status Register

Address: 4006 007Ch (UART1)
4006 107Ch (UART2)
4006 207Ch (UART3)
5000 007Ch (UART4)
5000 107Ch (UART5)
5000 207Ch (UART6)
5000 307Ch (UART7)
5000 407Ch (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bUart_RFF	bUart_RFNE	bUart_TFE	bUart_TFNF	bUart_BUSY
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Table 1.24 rUart_USR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved		R
b4	bUart_RFF	Receive FIFO Full. This is used to indicate that the receive FIFO is completely full. 1'b0 = Receive FIFO not full 1'b1 = Receive FIFO Full This bit is cleared when the Receive FIFO is no longer full.	R
b3	bUart_RFNE	Receive FIFO Not Empty. This is used to indicate that the receive FIFO contains one or more entries. 1'b0 = Receive FIFO is empty 1'b1 = Receive FIFO is not empty This bit is cleared when the Receive FIFO is empty.	R
b2	bUart_TFE	Transmit FIFO Empty. This is used to indicate that the transmit FIFO is completely empty. 1'b0 = Transmit FIFO is not empty 1'b1 = Transmit FIFO is empty This bit is cleared when the Transmit FIFO is no longer empty.	R
b1	bUart_TFNF	Transmit FIFO Not Full. This is used to indicate that the transmit FIFO in not full. 1'b0 = Transmit FIFO is full 1'b1 = Transmit FIFO is not full This bit is cleared when the Transmit FIFO is full.	R

Table 1.24 rUart_USR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b0	bUart_BUSY	UART Busy. This bit indicates that a serial transfer is in progress, when cleared, indicates that the UART is idle or inactive. 1'b0 = UART is idle or inactive 1'b1 = UART is busy (actively transferring data) This bit will be set under any of the following conditions: • Transmission in progress on serial interface • Transmit data present in rUart_THR, when FIFO access mode is not being used (bUart_FAR = 0) and the baud rate divisor is non-zero ({rUart_DLH, rUart_DLL} != 0) when the divisor latch access bit is 0 (bUart_DLAB = 0) • Reception in progress on the interface • Receive data present in rUart_RBR, when FIFO access mode is not being used (bUart_FAR = 0) ***Note*)** It is possible for the bUart_BUSY bit to be cleared even though a new character may have been sent from another device. That is, if the UART has no data in rUart_RBR and bUart_THR and there is no transmission in progress and a start bit of a new character has just reached the UART. This is due to the fact that a valid start is not seen until the middle of the bit period and this duration is dependent on the baud rate divisor that has been programmed. The assertion of this bit is also delayed by several cycles of the slower clock UART_SCLK and UART_PCLK.	R

1.4.17 rUart_TFL — Transmit FIFO Level

Address: 4006 0080h (UART1)
4006 1080h (UART2)
4006 2080h (UART3)
5000 0080h (UART4)
5000 1080h (UART5)
5000 2080h (UART6)
5000 3080h (UART7)
5000 4080h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bUart_TFL				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.25 rUart_TFL Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved		R
b4 to b0	bUart_TFL	Transmit FIFO Level. This indicates the number of data entries in the transmit FIFO.	R

1.4.18 rUart_RFL — Receive FIFO Level

Address: 4006 0084h (UART1)
4006 1084h (UART2)
4006 2084h (UART3)
5000 0084h (UART4)
5000 1084h (UART5)
5000 2084h (UART6)
5000 3084h (UART7)
5000 4084h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bUart_RFL				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.26 rUart_RFL Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved		R
b4 to b0	bUart_RFL	Receive FIFO Level. This indicates the number of data entries in the receive FIFO.	R

1.4.19 rUart_SRR — Software Reset Register

Address: 4006 0088h (UART1)
4006 1088h (UART2)
4006 2088h (UART3)
5000 0088h (UART4)
5000 1088h (UART5)
5000 2088h (UART6)
5000 3088h (UART7)
5000 4088h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_XFR	bUart_RFR	bUart_UR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.27 rUart_SRR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b3	Reserved		R
b2	bUart_XFR	Transmit FIFO Reset. This is a shadow register for the Transmit FIFO Reset bit (bUart_XFIFOR). This can be used just to reset the transmit FIFO by writing 1b to this bit. This resets the control portion of the transmit FIFO and treats the FIFO as empty. ***Note*)** This bit is 'self-clearing'. It is not necessary to clear this bit.	W
b1	bUart_RFR	Receive FIFO Reset. This is a shadow register for the receive FIFO Reset bit (bUart_RFIFOR). This can be used just to reset the receive FIFO by writing 1b to this bit. This resets the control portion of the receive FIFO and treats the FIFO as empty. ***Note*)** This bit is 'self-clearing'. It is not necessary to clear this bit.	W
b0	bUart_UR	UART Reset. This asynchronously resets the UART and synchronously removes the reset assertion by writing 1b to this bit. Both UART_SCLK and UART_PCLK domains are reset.	W

1.4.20 rUart_SRTS — Shadow Request to Send

Address: 4006 008Ch (UART1)
4006 108Ch (UART2)
4006 208Ch (UART3)
5000 008Ch (UART4)
5000 108Ch (UART5)
5000 208Ch (UART6)
5000 308Ch (UART7)
5000 408Ch (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_SRTS
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.28 rUart_SRTS Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	bUart_SRTS	Shadow Request to Send. This is a shadow register for the bUart_RTS bit. This can be used to remove the burden of having to performing a read-modify-write on the rUart_MCR. This is used to directly control the Request to Send (UART_RTS_N) output. The Request to Send (UART_RTS_N) output is used to inform the modem or data set that the UART is ready to exchange data. When Auto RTS Flow Control is not enabled (bUart_AFCE = 0), the UART_RTS_N signal is set low by programming bUart_RTS to a high. In Auto Flow Control, (bUart_AFCE = 1) and FIFOs enable (bUart_FIFOE = 1), the UART_RTS_N output is controlled in the same way, but it is gated by the receive FIFO almost-full trigger, where “almost full” refers to two available slots in the FIFO (UART_RTS_N is inactive high when above the threshold). ***Note*)** In Loopback mode (bUart_LB = 1), the UART_RTS_N output is held inactive-high while the value of this location is internally looped back to an input.	R/W

1.4.21 rUart_SBCR — Shadow Break Control Register

Address: 4006 0090h (UART1)
4006 1090h (UART2)
4006 2090h (UART3)
5000 0090h (UART4)
5000 1090h (UART5)
5000 2090h (UART6)
5000 3090h (UART7)
5000 4090h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_SBCR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.29 rUart_SBCR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	bUart_SBCR	Shadow Break Control Bit. This is a shadow register for the Break bit (bUart_BC), this can be used to remove the burden of having to performing a read modify write on the rUart_LCR. This is used to cause a break condition to be transmitted to the receiving device. If set to one the serial output is forced to the spacing (logic 0) state. When not in Loopback Mode, as determined by bUart_LB, the UART_TXD line is forced low until the Break bit is cleared. When in Loopback Mode, the break condition is internally looped back to the receiver.	R/W

1.4.22 rUart_SFE — Shadow FIFO Enable

Address: 4006 0098h (UART1)
4006 1098h (UART2)
4006 2098h (UART3)
5000 0098h (UART4)
5000 1098h (UART5)
5000 2098h (UART6)
5000 3098h (UART7)
5000 4098h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_SFE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.30 rUart_SFE Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	bUart_SFE	Shadow FIFO Enable. This is a shadow register for the FIFO enable bit (bUart_FIFOE). This can be used to remove the burden of having to store the previously written value to the rUart_FCR in memory and having to mask this value so that only the FIFO enable bit gets updated. This enables/disables the transmit and receive FIFOs. If this bit is set to zero (disabled) after being enabled then both the transmit and receive controller portion of FIFOs are reset. 1'b0 = Disable 1'b1 = Enable	R/W

1.4.23 rUart_SRT — Shadow RCVR Trigger

Address: 4006 009Ch (UART1)
4006 109Ch (UART2)
4006 209Ch (UART3)
5000 009Ch (UART4)
5000 109Ch (UART5)
5000 209Ch (UART6)
5000 309Ch (UART7)
5000 409Ch (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_SRCVR	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.31 rUart_SRT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved		R
b1, b0	bUart_SRCVR	Shadow Receive FIFO Trigger. This is a shadow register for the receive FIFO trigger bits (bUart_RCVR). This can be used to remove the burden of having to store the previously written value to the rUart_FCR in memory and having to mask this value so that only the Receive FIFO trigger bit gets updated. This is used to select the trigger level in the receive FIFO at which the Received Data Available Interrupt is generated. The following trigger levels are supported: 2’b00 = 1 character in the FIFO 2’b01 = FIFO 1/4 full 2’b10 = FIFO 1/2 full 2’b11 = FIFO 2 less than full	R/W

1.4.24 rUart_STET — Shadow TX Empty Trigger

Address: 4006 00A0h (UART1)
4006 10A0h (UART2)
4006 20A0h (UART3)
5000 00A0h (UART4)
5000 10A0h (UART5)
5000 20A0h (UART6)
5000 30A0h (UART7)
5000 40A0h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_STET	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.32 rUart_STET Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved		R
b1, b0	bUart_STET	Shadow Transmit Empty Trigger. This is a shadow register for the Transmit empty trigger bits (bUart_TET). This can be used to remove the burden of having to store the previously written value to the rUart_FCR in memory and having to mask this value so that only the Transmit empty trigger bit gets updated. This is used to select the empty threshold level at which the THRE Interrupts are generated when the mode is active. The following trigger levels are supported: 2’b00 = FIFO empty 2’b01 = 2 characters in the FIFO 2’b10 = FIFO 1/4 full 2’b11 = FIFO 1/2 full	R/W

1.4.25 rUart_HTX — Halt TX

Address: 4006 00A4h (UART1)
4006 10A4h (UART2)
4006 20A4h (UART3)
5000 00A4h (UART4)
5000 10A4h (UART5)
5000 20A4h (UART6)
5000 30A4h (UART7)
5000 40A4h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_HTX
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.33 rUart_HTX Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	bUart_HTX	Halt Transmission This register is used to halt transmissions for testing, so that the transmit FIFO can be filled by the master when FIFOs are enabled. 1'b0 = Halt Transmission disabled 1'b1 = Halt Transmission enabled ***Note*)** If FIFOs are not enabled, the setting of the bUart_HTX register has no effect on operation.	R/W

1.4.26 rUart_DMASA — DMA Software Acknowledge

- Only for UART4..8

Address: 5000 00A8h (UART4)
5000 10A8h (UART5)
5000 20A8h (UART6)
5000 30A8h (UART7)
5000 40A8h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bUart_DMASA
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.34 rUart_DMASA Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	bUart_DMASA	This register is used to perform a DMA Software acknowledge if a transfer needs to be terminated due to an error condition. An ACK response is executed by writing 1b to this bit. For example, if the DMA disables the channel, then the UART should clear its request. This causes the transmit and receive request signals to de-assert. ***Note*)** This bit is “self-clearing”. It is not necessary to clear this bit.	W

1.4.27 rUart_TO — Time-Out Counter Configuration Register

The Time-Out delay period during which the receiver or transceiver waits for a new character.

Address: 4006 0100h (UART1)
4006 1100h (UART2)
4006 2100h (UART3)
5000 0100h (UART4)
5000 1100h (UART5)
5000 2100h (UART6)
5000 3100h (UART7)
5000 4100h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bUart_TO3								bUart_TO2							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bUart_TO1								bUart_TO0							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.35 rUart_TO Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b24	bUart_TO3	bUart_TO[n] with n = 0..3 Time-Out n value with n = 3 There are 4 Time-Out • Two for reception bUart_TO0..1, dedicated for idle condition (Silent Interval Detection) on UART_RXD • Two for transmission bUart_TO2..3, dedicated for idle condition (Silent Interval Detection) on UART_TXD For each Time-Out (n = 0..3), we have: 8'h0: - The Time-Out is disabled. 8'h1..8'hff: - The Time-Out is enabled and the Time-out delay is bUart_TO[n] × "Baud clock period". See **Section 1.5.1.1, UART (RS232) Serial Protocol**. See **Section 1.5.1.10(2), Transceiver Time-Out**. See **Section 1.5.1.10(1), Receiver Time-Out**. The baud clock is set by the baud rate divisor (bUart_DLL and bUart_DLH). The time-out delay period is the time during which the receiver or transceiver waits for a new character on UART_RXD (Time-Out n with n = 0..1) or UART_TXD (Time-Out n with n = 2..3 output). If the bUart_TO[n] field is programmed at 0, the clocking of Time-Out Counter is stopped, the counter keeps current value. The bUart_TIMEOUTInt[n] bit in rUart_STATUSTO keeps current value. Otherwise, the receiver counter [n] with n = 0..1 or the transceiver counter [n] with n = 2..3 loads an 8-bit counter with the value programmed in bUart_TO[n].	R/W

Table 1.35 rUart_TO Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
		This counter [n] is decremented at each bit period and reloaded each time a new character is received (n = 0..1) or transmitted (n = 2..3). If the counter [n] reaches 0, the bUart_TIMEOUTInt[n] bit in the Time-Out Counter Status Register (rUart_STATUSTO) rises and activates an interruption (if not masked). When the counter reaches 0, it remains locked until reception of a LOAD command. See **Figure 1.14, Receiver Time-Out Synoptic**. See **Figure 1.15, Transceiver Time-Out Synoptic**. See **Figure 1.16, Receiver & Transceiver Time-Out0..3, Timing Description**. See **Section 1.5.1.10(1), Receiver Time-Out**. See **Section 1.5.1.10(2), Transceiver Time-Out**.	
b23 to b16	bUart_TO2	Time-Out n value with n = 2 See description detailed above	R/W
b15 to b8	bUart_TO1	Time-Out n value with n = 1 See description detailed above	R/W
b7 to b0	bUart_TO0	Time-Out n value with n = 0 See description detailed above	R/W

1.4.28 rUart_CTRLTO — Time-Out Control Register

Address: 4006 0104h (UART1)
4006 1104h (UART2)
4006 2104h (UART3)
5000 0104h (UART4)
5000 1104h (UART5)
5000 2104h (UART6)
5000 3104h (UART7)
5000 4104h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	bUart_TG							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	bUart_EnableFilteringRXD	bUart_EnableDE	bUart_REARMTO3	bUart_REARMTO2	bUart_REARMTO1	bUart_REARMTO0	bUart_STARTTO3	bUart_STARTTO2	bUart_STARTTO1	bUart_STARTTO0
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.36 rUart_CTRLTO Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b24	Reserved		R
b23 to b16	bUart_TG	Time-Guard value Not available in this LSI. Keep the initial value.	R/W
b15 to b10	Reserved		R
b9	bUart_EnableFilteringRXD	Allows the filtering of UART_RXD in Half-Duplex mode Not available in this LSI. Keep the initial value.	R/W
b8	bUart_EnableDE	Allow the multiplexing of external pin UART_RTS_N. Not available in this LSI. Keep the initial value.	R/W
b7	bUart_REARMTO3	bUart_REARMTO[n] with n = 0..3, Rearm Time-out Rearm Time-Out n value with n = 3 For each Time-Out (n = 0..3), we have: 0 = No effect 1 = Restart the Time-Out counter After activation of bUart_REARMTO[n] (n = 0..3) (Rising edge detection), the counter starts counting down immediately from the bUart_TO[n] (n = 0..3). See **Figure 1.14, Receiver Time-Out Synoptic**. See **Figure 1.15, Transceiver Time-Out Synoptic**. See **Section 1.5.1.10(1), Receiver Time-Out**. See **Section 1.5.1.10(2), Transceiver Time-Out**.	R/W
b6	bUart_REARMTO2	Rearm Time-Out n value with n = 2 See description detailed above	R/W
b5	bUart_REARMTO1	Rearm Time-Out n value with n = 1 See description detailed above	R/W
b4	bUart_REARMTO0	Rearm Time-Out n value with n = 0 See description detailed above	R/W

Table 1.36 rUart_CTRLTO Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3	bUart_STARTTO3	bUart_STARTTO[n] with n = 0..3, Start Time-out Start Time-Out n value with n=3 For each Time-Out (n = 0..3), we have: 0 = No effect 1 = Starts an initialization of the Time-Out Counter After activation of bUart_STARTTO[n] (n = 0..3) (Rising edge detection), initializes the Time-Out counter and turns it in the locked state. See **Figure 1.14, Receiver Time-Out Synoptic**. See **Figure 1.15, Transceiver Time-Out Synoptic**. See **Section 1.5.1.10(1), Receiver Time-Out**. See **Section 1.5.1.10(2), Transceiver Time-Out**.	R/W
b2	bUart_STARTTO2	Start Time-Out n value with n = 2 See description detailed above	R/W
b1	bUart_STARTTO1	Start Time-Out n value with n = 1 See description detailed above	R/W
b0	bUart_STARTTO0	Start Time-Out n value with n = 0 See description detailed above	R/W

1.4.29 rUart_STATUSTO — Time-Out Counter Status Register

The Software driver (application) reads this register during interrupt service routine or polling to determine the status of each Time-Out.

Address: 4006 0108h (UART1)
4006 1108h (UART2)
4006 2108h (UART3)
5000 0108h (UART4)
5000 1108h (UART5)
5000 2108h (UART6)
5000 3108h (UART7)
5000 4108h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	bUart_DE	bUart_TIMEOUTStatus3	bUart_TIMEOUTStatus2	bUart_TIMEOUTStatus1	bUart_TIMEOUTStatus0	bUart_TIMEOUTInt3	bUart_TIMEOUTInt2	bUart_TIMEOUTInt1	bUart_TIMEOUTInt0
Value after reset	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0

Table 1.37 rUart_STATUSTO Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b9	Reserved		R
b8	bUart_DE	This register gives the internal value of the Transmit Data Enable. Not available in this LSI. Ignore the read value.	R
b7	bUart_TIMEOUTStatus3	bUart_TIMEOUTStatus[n] with n = 0..3, Time-Out Detection Status Time-Out n Detection Status with n = 3 This register is usually read by the Software driver during an interrupt service routine or polling. There are 4 Time-Out (bUart_TO[n] with n = 0.3) • Two for reception bUart_TO0..1, dedicated for idle condition (Silent Interval Detection) on UART_RXD • Two for transmission bUart_TO2..3, dedicated for idle condition (Silent Interval Detection) on UART_TXD For each Time-Out n with n = 0..3, we have: 1'b0 = The Time-Out Counter[n] value is different of "0". 1'b1 = The Time-Out Counter[n] value is equal of "0". See **Section 1.5.1.10, Transceiver & Receiver Time-Out for MODBUS Management**. See **Figure 1.14, Receiver Time-Out Synoptic**. See **Figure 1.15, Transceiver Time-Out Synoptic**. See **Figure 1.16, Receiver & Transceiver Time-Out0..3, Timing Description**. See **Section 1.5.1.10(1), Receiver Time-Out**. See **Section 1.5.1.10(2), Transceiver Time-Out**.	R
b6	bUart_TIMEOUTStatus2	Time-Out n Detection Status with n = 2 See description detailed above	R
b5	bUart_TIMEOUTStatus1	Time-Out n Detection Status with n = 1 See description detailed above	R
b4	bUart_TIMEOUTStatus0	Time-Out n Detection Status with n = 0 See description detailed above	R

Table 1.37 rUart_STATUSTO Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3	bUart_TIMEOUTInt3	bUart_TIMEOUTInt[n] with n = 0..3, Time-Out Detection Interruption Time-Out n Detection Interruption with n = 3 This register is usually read by the Software driver during an interrupt service routine or polling. Most of the fields in this register cause the host to be interrupted. There are 4 Time-Out (bUart_TO[n] with n=0.3) • Two for reception bUart_TO0..1, dedicated for idle condition (Silent Interval Detection) on UART_RXD • Two for transmission bUart_TO2..3, dedicated for idle condition (Silent Interval Detection) on UART_TXD For each Time-Out n with n = 0..3, we have: 1'b0 = There has not been a Time-Out since the last Start Time-out command. There is not active interruption from Time-Out. 1'b1 = There has been a Time-Out since the last Start Time-Out command. When this bit is high, the interrupt signal, UART_Int, is high. Each field (bits[3:0]) can be masked by masking the appropriate bit in rUart_IER register. See **Section 1.5.1.6, Interrupts**. In this register, the bits are not cleared when read. Writing 1'b1 to (unreserved) bits in this register (bits[3:0]) clears them and writing 1'b0 has no effect. Each Time-Out n with n = 0..3 can be started or reset by appropriate bit in rUart_CTRLTO register See **Section 1.5.1.10, Transceiver & Receiver Time-Out for MODBUS Management**. See **Figure 1.14, Receiver Time-Out Synoptic**. See **Figure 1.15, Transceiver Time-Out Synoptic**. See **Figure 1.16, Receiver & Transceiver Time-Out0..3, Timing Description**. See **Section 1.5.1.10(1), Receiver Time-Out**. See **Section 1.5.1.10(2), Transceiver Time-Out**.	R/W
b2	bUart_TIMEOUTInt2	Time-Out n Detection Interruption with n = 2 See description detailed above	R/W
b1	bUart_TIMEOUTInt1	Time-Out n Detection Interruption with n = 1 See description detailed above	R/W
b0	bUart_TIMEOUTInt0	Time-Out n Detection Interruption with n = 0 See description detailed above	R/W

1.4.30 rUart_TDMACR — DMA Control Register in Transmit Mode

- Only for UART4..8

CAUTION

Only these UARTs can manage DMA request with DMA controller.

Address: 5000 010Ch (UART4)
5000 110Ch (UART5)
5000 210Ch (UART6)
5000 310Ch (UART7)
5000 410Ch (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	bUart_CURRENT_DEST_BLOCK_SIZE												
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bUart_DEST_BLOCK_SIZE													bUart_DEST_BURST_SIZE		bUart_TDMAE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.38 rUart_TDMACR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b29	Reserved		R
b28 to b16	bUart_CURRENT_DEST_BLOCK_SIZE	Current remaining of DEST_BLOCK_SIZE. This field is decremented each time the block transfer ends. bUart_CURRENT_DEST_BLOCK_SIZE is reloaded with bUart_DEST_BLOCK_SIZE value, when the firmware: Set “1” on bUart_TDMAE (rising edge)	R
b15 to b3	bUart_DEST_BLOCK_SIZE	DEST_BLOCK_SIZE Destination Block Transfer Size in Transmit FIFO. UART is the flow controller. Thus, the user must write this field before or at the same time the DMA mode is enabled. The number programmed into DEST_BLOCK_SIZE indicates the total number of single transactions to perform for each block transfer. The size of single transaction is one byte. Once the transfer starts, the read of bUart_DEST_BLOCK_SIZE gives the total number of data bytes to be written in the Transmit FIFO in order to end the block transfer. 13’d0 = 0 byte to transfer or end of block transfer 13’d1 = 1 byte to transfer 13’d2 = 2 bytes to transfer ... ... 13’d8191 = 8191 bytes to transfer	R/W

Table 1.38 rUart_TDMACR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b2, b1	bUart_DEST_BURST_SIZE	DEST_BURST_SIZE Destination Burst Transaction Size in Transmit FIFO. UART is the flow controller. Thus, the user must write this field before or at the same time the DMA mode is enabled. Number of data byte, to be written to the Transmit FIFO every time a transmit burst transaction request are made on DMA request. 2'b00 = 1 byte 2'b01 = 4 bytes 2'b10 = 8 bytes 2'b11 = Reserved, not used	R/W
b0	bUart_TDMAE	Transmit DMA Enables/Disables. 1'b0 = Disable the DMA in Transmit mode 1'b1 = Enable the DMA in Transmit mode The bUart_TDMAE is automatically cleared by hardware to disable the DMA in Transmit mode after the last transfer in Transmit FIFO has completed (DEST_BLOCK_SIZE bytes written in Transmit FIFO). Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. When disable, no DMA request is asserted. When enable, UART manages the DMA request with DMA controllers. ***Caution***) If this bit is clear during a DMA transfer, the current transfer (Burst or Single) is finished before the stop of DMA mode. To complete the DMA Block transfer, write the bUart_DEST_BLOCK_SIZE with the bUart_CURRENT_DEST_BLOCK_SIZE, and bUart_DEST_BURST_SIZE with the appropriate value. The bUart_CURRENT_DEST_BLOCK_SIZE value will be consistent only when the current transfer is finished.	R/W

1.4.31 rUart_RDMACR — DMA Control Register in Receive Mode

- Only for UART4..8

CAUTION

Only these UARTs can manage DMA request with DMA controller.

Address: 5000 0110h (UART4)
5000 1110h (UART5)
5000 2110h (UART6)
5000 3110h (UART7)
5000 4110h (UART8)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	bUart_CURRENT_SRC_BLOCK_SIZE												
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bUart_SRC_BLOCK_SIZE													bUart_SRC_BURST_SIZE		bUart_RDMAE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.39 rUart_RDMACR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b29	Reserved		R
b28 to b16	bUart_CURRENT_SRC_BLOCK_SIZE	Current remaining of SRC_BLOCK_SIZE. This field is decremented each time the block transfer ends. bUart_CURRENT_SRC_BLOCK_SIZE is reloaded with bUart_SRC_BLOCK_SIZE value, when the firmware: Set “1” on bUart_RDMAE (rising edge)	R
b15 to b3	bUart_SRC_BLOCK_SIZE	SRC_BLOCK_SIZE Source Block Transfer Size in Receive FIFO. UART is the flow controller. Thus, the user must write this field before or at the same time the DMA mode is enabled. The number programmed into SRC_BLOCK_SIZE indicates the total number of single transactions to perform for each block transfer. The size of single transaction is one byte. 13’d0 = 0 byte to transfer or end of block transfer 13’d1 = 1 byte to transfer 13’d2 = 2 bytes to transfer ... ... 13’d8191 = 8191 bytes to transfer	R/W

Table 1.40 rUart_RDMACR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b2, b1	bUart_SRC_BURST_SIZE	SRC_BURST_SIZE Source Burst Transaction Size in Receive FIFO. UART is the flow controller. Thus, the user must write this field before or at the same time the DMA mode is enabled. Number of data byte, to be read to the Receive FIFO every time a receive burst transaction request are made on DMA request. 2'b00 = 1 byte 2'b01 = 4 bytes 2'b10 = 8 bytes 2'b11 = Reserved, not used	R/W
b0	bUart_RDMAE	Receive DMA Enables/Disables. 1'b0 = Disable the DMA in Receive mode 1'b1 = Enable the DMA in Receive mode The bUart_RDMAE is automatically cleared by hardware to disable the DMA in Receive mode after the last transfer in Receive FIFO has completed (SRC_BLOCK_SIZE bytes read in Receive FIFO). Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. When disable, no DMA request is asserted. When enable, UART manages the DMA request with DMA controllers. ***Caution***) If this bit is clear during a DMA transfer, the current transfer (Burst or Single) is finished before the stop of DMA mode. To complete the DMA Block transfer, write the bUart_SRC_BLOCK_SIZE with the bUart_CURRENT_SRC_BLOCK_SIZE, and bUart_SRC_BURST_SIZE with the appropriate value. The bUart_CURRENT_SRC_BLOCK_SIZE value will be consistent only when the current transfer is finished.	R/W

1.5 Operation

1.5.1 Main Function Blocks Description

1.5.1.1 UART (RS232) Serial Protocol

Because the serial communication between the UART and the selected device is asynchronous, additional bits (start and stop) are added to the serial data to indicate the beginning and end. Utilizing these bits allows two devices to be synchronized. This structure of serial data accompanied by start and stop bits is referred to as a character, as shown in figure below.

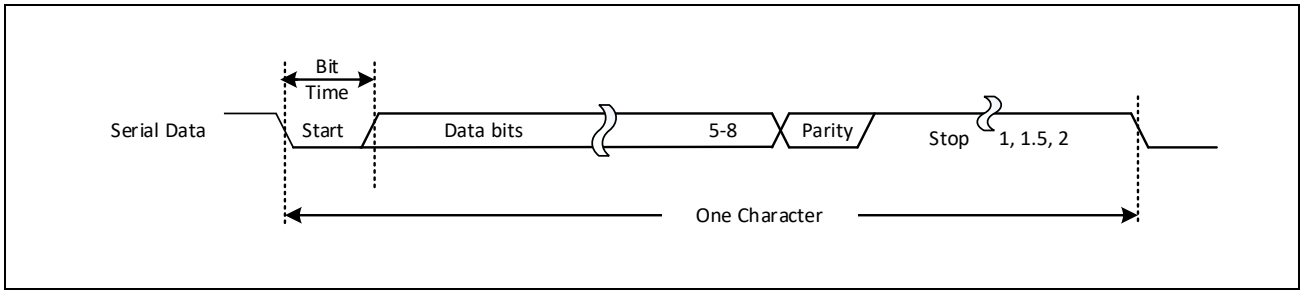

Figure 1.3 Serial Data Format

An additional parity bit may be added to the serial character. This bit appears after the last data bit and before the stop bit(s) in the character structure to provide the UART with the ability to perform simple error checking on the received data.

The UART Line Control Register (rUart_LCR) is used to control the serial character characteristics. The individual bits of the data word are sent after the start bit, starting with the least significant bit (LSB). These are followed by the optional parity bit, followed by the stop bit(s), which can be 1, 1.5 or 2.

CAUTION

- For details on idle time between transmitted transfers, refer to **Section 1.5.1.5, Back to Back Character Stream Transmission**.
- The STOP bit duration implemented by UART can appear longer due to:
  a) Idle time inserted between characters for some configurations
  b) Baud rate divisor values in the transmit direction

All the bits in the transmission are transmitted for exactly the same time duration; the exception to this is the half-stop bit when 1.5 stop bits are used. This duration is referred to as a Bit Period or Bit Time; one Bit Time equals a baud clock period. Internal operation is performed with the base clock (16 times frequency of the baud clock).
To ensure stability on the line, the receiver samples the serial input data at approximately the midpoint of the Bit Time once the start bit has been detected. Because the exact number of base clocks is known for which each bit was transmitted, calculating the midpoint for sampling is not difficult; that is, a baud clock after the midpoint sample of the start bit.
Together with serial input de-bouncing, this sampling helps to avoid the detection of false start bits. Short glitches are filtered out by de-bouncing, and no transition is detected on the line. If a glitch is wide enough to avoid filtering by de-bouncing, a falling edge is detected. However, a start bit is detected only if the line is again sampled low after half a bit time has elapsed.

The figure below shows the sampling points of the first couple of bits in a serial character.

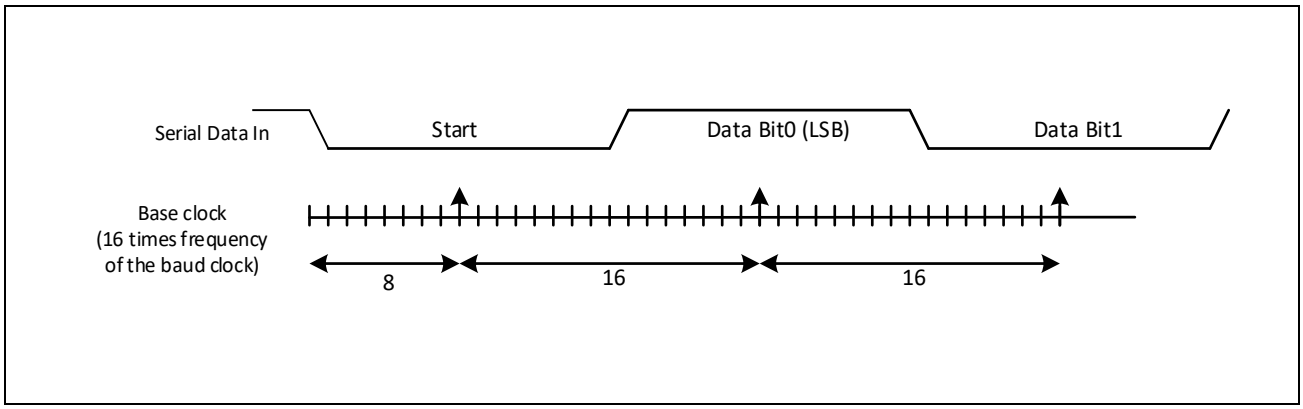

Figure 1.4 Receiver Serial Data Sample

As part of the standard 16550, a baud clock module provides timing information. The baud clock of the UART is controlled by UART_SCLK and the baud rate divisor (rUart_DLH and rUart_DLL). The baud clock is equal to UART_SCLK frequency divided by sixteen times the value of the baud rate divisor, as follows:

$$\text{Baud clock} = \frac{\text{UART\_SCLK}}{16 \times \text{baud rate divisor}}$$

1.5.1.2 Baud Rate Tolerance to 19200 baud

Evaluation of tolerance on baud rate in reception at 19200 baud (UART_SCLK: 48 MHz)

Configuration UART baud rate in reception:

- The firmware must write rUart_DLL & rUart_DLH registers a value of 16'd156
  - (48/16)/156 = 0.0192307 → 19230 baud
  - We have an error of +0.16% on baud rate configuration

Maximum configuration UART baud rate in transmission with reception:

- The firmware must write rUart_DLL & rUart_DLH registers a value of 16'd151
  - (48/16)/151 → 19867 baud
  - We have an error of +3.47% on baud rate configuration

Minimum configuration UART baud rate in transmission with reception:

- The firmware must write rUart_DLL & rUart_DLH registers a value of 16'd161
  - (48/16)/161 → 18633 baud
  - We have an error of −2.95% on baud rate configuration

1.5.1.3 FIFO Management

Two separates FIFOs (16 × 8 bits) are available to buffer transmit and receive data to reduce CPU interrupts with programmable FIFO enable/disable. This means that the CPU does not have to access the UART each time a single byte of data is received.

The programmable FIFO Access mode is available for test purposes, which allows the receive FIFO to be written by the CPU and the transmit FIFO to be read by the CPU.

The FIFO Access mode can be enabled with the FIFO Access Register, (rUart_FAR register). Once enabled, the control portions of the transmit and receive FIFOs are reset and the FIFOs are treated as empty.

Data can be written to the transmit FIFO as normal, however no serial transmission occurs in this mode (normal operation halted) and hence no data leave the FIFO. The data that has been written to the transmit FIFO can be read back with the Transmit FIFO Read (rUart_TFR register), which when read gives the current data at the top of the transmit FIFO.

Similarly, data can be read from the receive FIFO as normal. Since the normal operation of the UART is halted in this mode, data must be written to the receive FIFO so it may be read back.

Data is written to the receive FIFO with the Receive FIFO Write (rUart_RFW register). The upper two bits of the 10-bit register (bUart_RFFE & bUart_RFPE) are used to write framing error and parity error detection information to the receive FIFO.

Setting bUart_RFFE to indicate a framing error and bUart_RFPE to indicate a parity error. Although these bits cannot be read back via the Receive Buffer Register they can be checked by reading the Line Status Register and checking the corresponding bits when the data in question is at the top of the receive FIFO.

1.5.1.4 Clock Management

UART uses two asynchronous domain clocks

- UART_PCLK APB clock domain in Subsystem
- UART_SCLK Serial reference clock

A synchronization module is implemented for synchronization of all control and data across the two system clock boundaries. It generates an additional latency between two domain clocks.

Here are a few things to keep in mind:

- There is slightly more time required after initial serial control register programming before serial data can be transmitted or received.
- The serial clock modules must have time to see new register values and reset their respective state machines. This total time is guaranteed to be no more than eight clock cycles of the slower of the two system clocks. Therefore, no data should be transmitted or received before this maximum time expires, after initial configuration.

1.5.1.5 Back to Back Character Stream Transmission

This section describes:

- Scenarios under which the UART is capable of transmitting back to back characters on the serial interface, with no idle time between them
- Worst case idle time that exists between back to back characters

When the Transmit FIFO contains multiple data entries, the UART transmits the characters in the FIFO back to back on the serial bus. However, because synchronization of all control and data across the two system clock boundaries. We

have an additional latency between two domain clocks. This delay can cause an IDLE period between the end of the current STOP bit and the beginning of the next START bit; this appears as an extended STOP bit duration on the serial bus.

Because synchronization delay between the transmitter in the UART_SCLK domain and the TX FIFO in the UART_PCLK domain when querying if another character is ready for transmission. The transmitter begins the handshake one base clock (16 times frequency of the baud clock) before the end of the current STOP bit. The duration of the synchronization delay (sync_delay) is given by the following equations:

sync_delay = 4 × UART_SCLK + 5 × UART_PCLK periods

If the sync_delay duration is longer than one base clock (16 times frequency of the baud clock) period, an IDLE period will be inserted between the end of a STOP bit and the beginning of the next START bit. To prevent insertion of the IDLE period, the following condition must be true:

sync_delay ≤ base clock period

- The worst case timing of the inserted IDLE period is given by:
  worst_case_idle_duration = sync_delay + (15 × base clock period)
  The worst_case_idle_duration can be added to the programmed STOP bit duration to give the overall STOP bit period

1.5.1.6 Interrupts

The assertion of the UART_Int occurs whenever one of the several prioritized interrupt types are enabled and active. The following interrupt types can be enabled with the rUart_IER register:

- Receiver Line Status
- Received Data Available
- Character Timeout (in FIFO mode only)
- Transmit Holding Register Empty at/below threshold (in Programmable THRE interrupt mode)
- Modem Status
- Busy Detect Indication
- Receiver Time Out0..1 on UART_RXD
- Transceiver Time Out2..3 on UART_TXD

These interrupt types are covered in more detail in **Table 1.41, Interrupt Control Functions**.

When an interrupt occurs, the master accesses the rUart_IIR register to determine the source of the interrupt before dealing with it accordingly.

In the FIFO mode (bUart_FIFOE = 1), the Receiver Line Status interrupt with the following sources: break Interrupt, framing error, parity error is revealed when the character with the error arrives at the top of the FIFO.

Table 1.41 Interrupt Control Functions

bUart_IID	Priority Level	Interrupt Type	Interrupt Source	Interrupt Enable	Interrupt Reset Control
4'b0001	—	None	None	—	—
4'b0110	Highest	Receiver Line Status	Overrun/parity/framing errors or break interrupt In the FIFO mode (bUart_FIFOE set to one), the Receiver Line Status interrupt with the following sources: break Interrupt, framing error, parity error is revealed when the character with the error arrives at the top of the FIFO.	bUart_ELSI	Reading the Line Status Register.
4'b0100	Second	Received Data available	Receiver data available (FIFOs disabled) or Receive FIFO trigger level reached (FIFOs enabled).	bUart_ERBFI	Reading the Receive Buffer Register (FIFOs disabled) or the FIFO drops below the trigger level (FIFOs enabled).
4'b1100	Second	Character Timeout Indication	No characters in or out of the Receive FIFO during the last 4-character times and there is at least 1 character in it during this time. (in FIFO mode only)	bUart_ERBFI	Reading the Receive Buffer Register.
4'b0010	Third	Transmit Holding Register empty or Transmit FIFO level at or below threshold	Transmit Holding Register empty (Programmable THRE Mode disabled) or Transmit FIFO level at or below threshold (ProgrammableTHRE Mode enabled) After a clear on THRE interrupt by reading Interrupt Identification Register, although the interrupt source is always true, an internal mask will de-assert the THRE interrupt. This interrupt will be re-asserted at the start of each transfer if the interrupt source is always true.	bUart_ETBEI bUart_PTIME	Reading the Interrupt Identification Register (if THRE is the source of interrupt) or writing into Transmit Holding Register (FIFOs or Programmable THRE Mode disabled) or Transmit FIFO level above threshold (FIFOs and Programmable THRE Mode enabled).
4'b0000	Fourth	Modem Status	Clear to send or data set ready or ring indicator or data carrier detect. ***Note*)** If auto flow control mode is enabled, a change in bUart_CTS (bUart_DCTS set) does not cause an interrupt.	bUart_EDSSI	Reading the Modem Status Register.
4'b0111	Fifth	Busy Detect Indication	Master has tried to write to the Line Control Register while the UART is busy (bUart_BUSY is set to one).	—	Reading the UART Status Register.
4'b0101	Sixth	Receiver Transceiver Time-Out	This feature detects an idle condition on the UART_RXD or UART_TXD line. When a Time-Out is detected, the bUart_TIMEOUTInt[n] (n = 0..3) bit in the status register (rUart_STATUSTO) rises and generates an interrupt, thus indicating to the driver an end of frame. See **Section 1.5.1.10, Transceiver & Receiver Time-Out for MODBUS Management**.	bUart_ETIMEOUT[n] (n = 0..3)	Reading the Time-Out Counter Status Register (rUart_STATUSTO).

1.5.1.7 Auto Flow Control

The UART can be configured to have a 16750 compatible Auto RTS and Auto CTS serial data flow control mode available. Auto Flow Control mode can be enabled with bUart_AFCE bit in the Modem Control Register (rUart_MCR).

Auto RTS becomes active when the following occurs:

- bUart_RTS and bUart_AFCE bit of rUart_MCR register are both set
- FIFOs are enabled (bUart_FIFOE bit is set)

When Auto RTS is enabled (active), the UART_RTS_N output is forced inactive (high) when the FIFO is almost full, where “almost full” refers to two available slots in the FIFO. When UART_RTS_N is connected to the UART_CTS_N input of another UART device, the other UART stops sending serial data until the receive FIFO has available space (until it is completely empty).

The selectable receive FIFO threshold values are:

- 1
- 1/4
- 1/2
- “2 less than full”

Since one additional character may be transmitted to the UART after UART_RTS_N has become inactive (due to data already having entered the transmitter block in the other UART), setting the threshold to “2 less than full” allows maximum use of the FIFO with a safety zone of one character.

Once the receive FIFO becomes completely empty by reading the Receiver Buffer Register (rUart_RBR), UART_RTS_N again becomes active (low), signaling the other UART to continue sending data.

It is important to note that even if everything else is selected and the rUart_MCR register is set correctly, if the FIFOs are disabled through bUart_FIFOE, Auto Flow Control is also disabled. When Auto RTS is disabled, UART_RTS_N is controlled solely by bUart_RTS.

The figure below shows a timing diagram of Auto RTS operation.

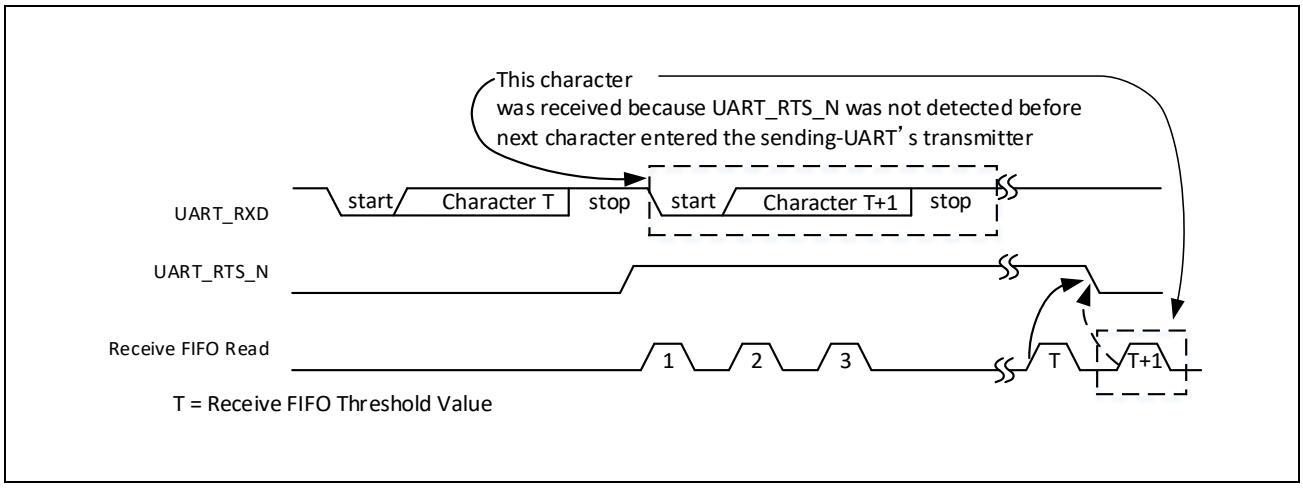

Figure 1.5 Auto RTS Timing

Auto CTS becomes active when the following occurs:

- bUart_AFCE bit of rUart_MCR register is set
- FIFOs are enabled (bUart_FIFOE bit is set)

When Auto CTS is enabled (active), the UART transmitter is disabled whenever the UART_CTS_N input becomes inactive (high). This prevents overflowing the FIFO of the receiving UART.

If the UART_CTS_N input is not inactivated before the middle of the last STOP bit, another character is transmitted before the transmitter is disabled. While the transmitter is disabled, the transmit FIFO can still be written to, and even overflowed.

Therefore, when using this mode, the following happens:

- The UART status register can be read to check if the transmit FIFO is full (bUart_TFNF = 0).
- The current FIFO level can be read via the rUart_TFL register.
- The Programmable THRE Interrupt mode must be enabled to access the “FIFO full” status via the Line Status Register (rUart_LSR).

When using the “FIFO full” status, Software can poll this before each write to the Transmitter FIFO. See **Section 1.5.1.8, Programmable THRE interrupt** for details. When the UART_CTS_N input becomes active (low) again, transmission resumes.

It is important to note that even if everything else is selected, if the FIFOs are disabled via bUart_FIFOE, Auto Flow Control is also disabled. When Auto CTS is disabled, the transmitter is unaffected by UART_CTS_N.

A timing diagram showing Auto CTS operation can be seen in figure below.

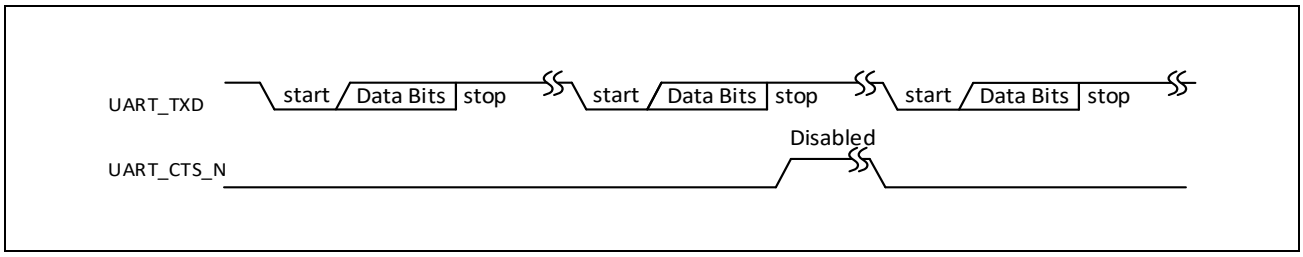

Figure 1.6 Auto CTS Timing

1.5.1.8 Programmable THRE interrupt

The UART have a Programmable THRE Interrupt mode available to increase system performance and can be enabled via the Interrupt Enable Register (bUart_PTIME of rUart_IER).

When FIFOs and the Programmable THRE Mode are enabled, THRE Interrupts mode (when also enabled) and UART DMA request are active at, and below, a programmed transmit FIFO empty threshold level, as opposed to empty, as shown in the flowchart in figure below.

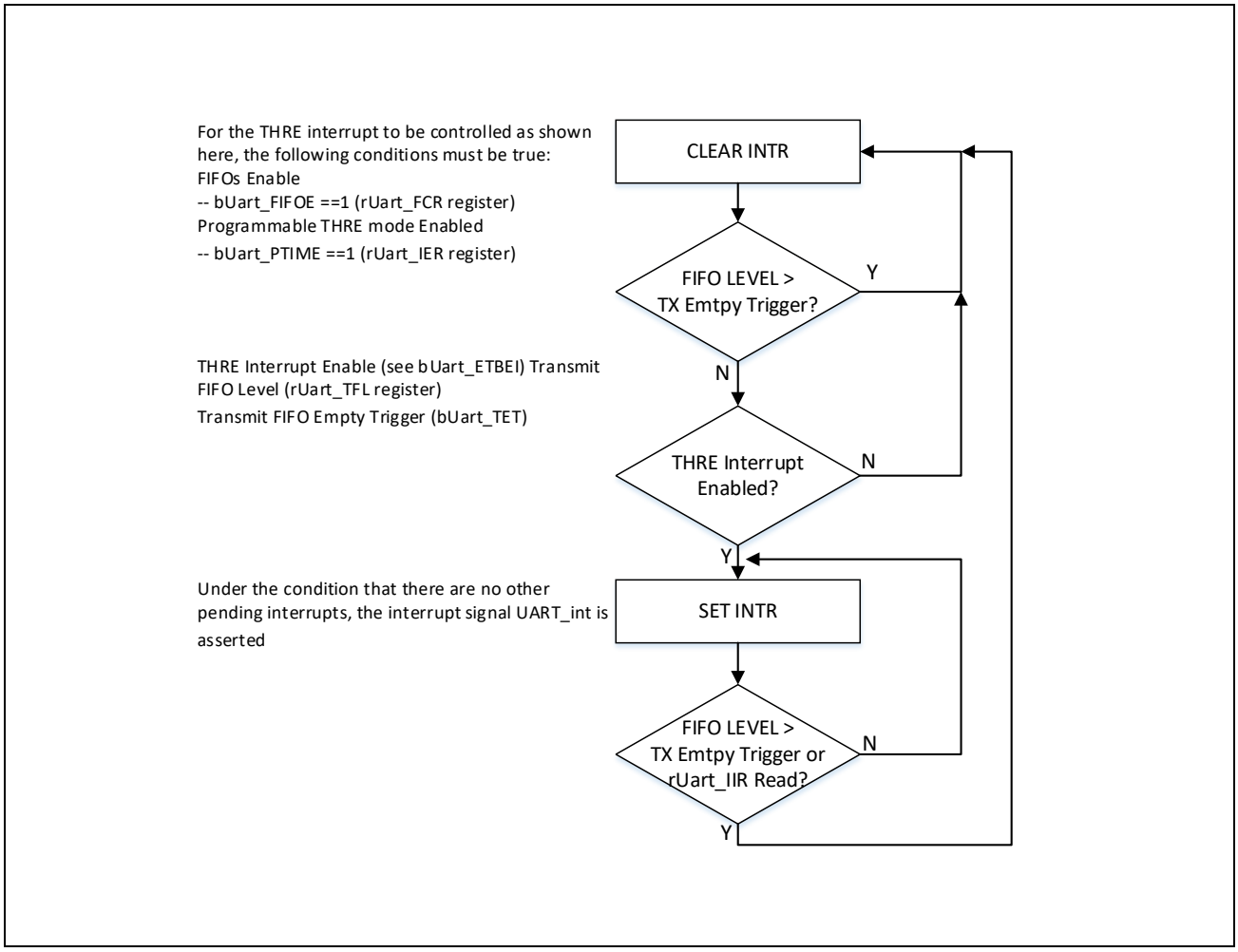

Figure 1.7 Flowchart of Interrupt Generation, Programmable THRE Interrupt Mode & FIFO Enable

This threshold trigger level (Transmit FIFOs Empty trigger) is programmed into bUart_TET bits. The available empty thresholds are: empty, 2, 1/4 and 1/2. See rUart_FCR register for threshold setting details.

Selection of the best threshold value depends on the system's ability to begin a new transmission sequence in a timely manner. However, one of these thresholds should prove optimum in increasing system performance by preventing the transmit FIFO from running empty.

In addition to the interrupt change, bUart_THRE bit in Line Status Register (rUart_LSR register) also switches function from indicating transmit FIFO empty to FIFO full. This allows Software to fill the FIFO each transmit sequence by polling bUart_THRE before writing another character. The flow then allows the transmit FIFO to be filled whenever an interrupt occurs and there is data to transmit, rather than waiting until the FIFO is completely empty. Waiting until the FIFO is empty causes a reduction in performance whenever the system is too busy to respond immediately. Further system efficiency is achieved when this mode is enabled in combination with Auto Flow Control.

Even if everything else is selected and enabled, if the FIFOs are disabled using the bUart_FIFOE bit, the Programmable THRE Interrupt mode is also disabled. When not selected or disabled, THRE interrupts and the bUart_THRE bit function normally, signifying an empty THR or FIFO.

The figure below illustrates the flowchart of THRE interrupt generation when not in programmable THRE interrupt mode.

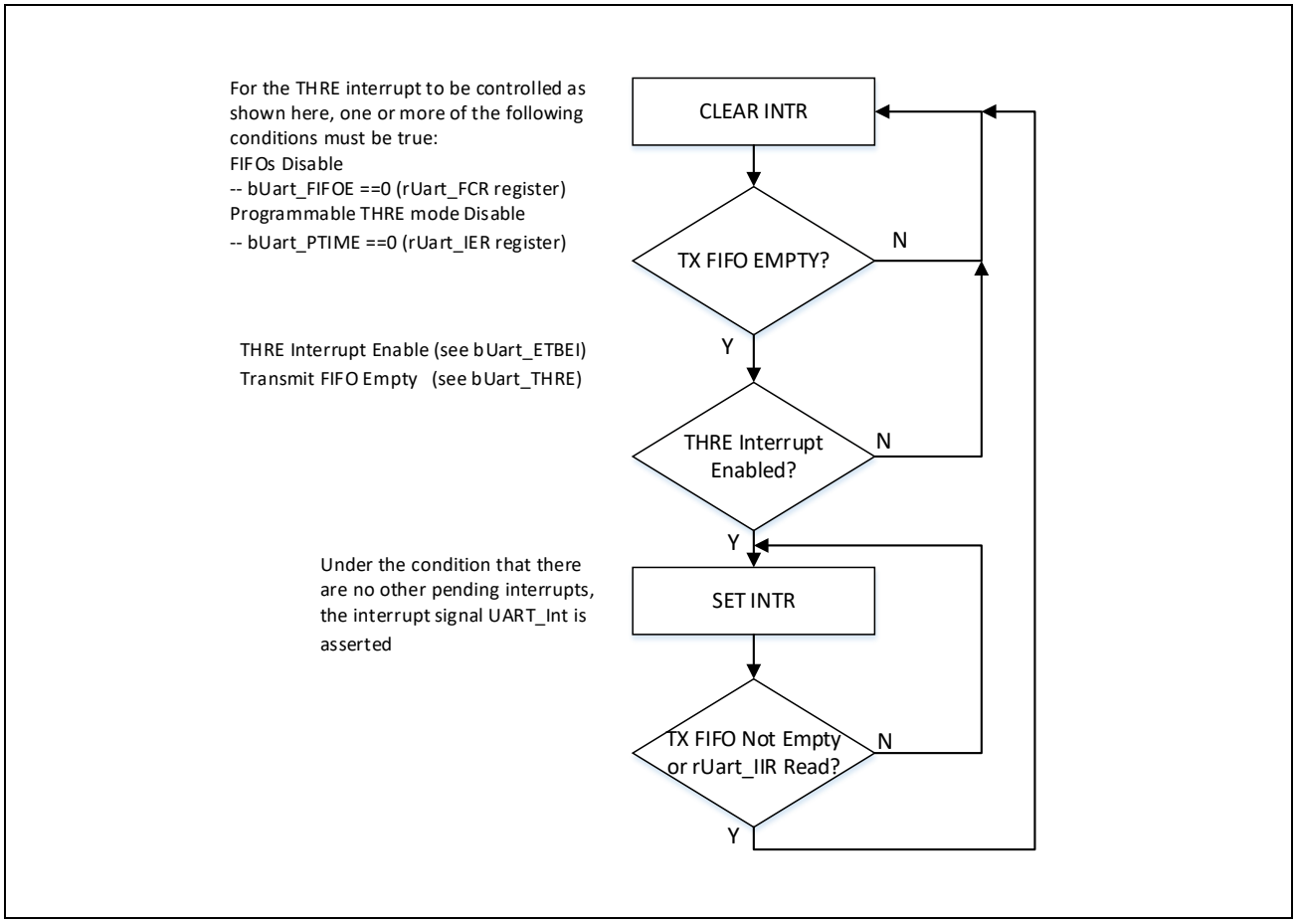

Figure 1.8 Flowchart of Interrupt Generation, Programmable THRE Interrupt Mode or FIFO Disable

After a clear on THRE interrupt by reading rUart_IIR, although the interrupt source is always true, an internal mask will de-assert the THRE interrupt. This interrupt will be re-asserted at the start of each transfer if the interrupt source is always true.

If the FIFOs are disabled via bUart_FIFOE (rUart_FCR register), the Programmable THRE Interrupt mode is also disabled.

When Programmable THRE Interrupt mode are disabled, THRE interrupts and bUart_THRE bit in Line Status Register (rUart_LSR register) function normally (both reflecting an empty rUart_THR or Transmit FIFO).

1.5.1.9 DMA Management (Only UART4, 5, 6, 7, 8)

(1) Overview on DMA Operation

These UARTs uses two DMA channels, one for the transmit data and one for the receive data. DMA controller must be configured in peripheral flow controller mode.

These UARTs are configured to have additional DMA interface signals to indicate when data is ready to be read or when the transmit FIFO is empty:

- Transmit DMA request is asserted under the following conditions:
  - When the Transmit Holding Register (rUart_THR) is empty in non FIFO mode, in this mode the DEST_BURST_SIZE must be programmed to 2'b00 (1 byte).
  - When the transmit FIFO is empty in FIFO mode with Programmable THRE interrupt mode disabled
  - When the transmit FIFO is at, or below the programmed threshold with Programmable THRE interrupt mode enabled.
- Receive DMA request is asserted under the following conditions:
  - When there is a single character available in the Receive Buffer Register (rUart_RBR) in non FIFO mode, in this mode the SRC_BURST_SIZE must be programmed to 2'b00 (1 byte).
  - When the Receiver FIFO is at or above the programmed trigger level in FIFO mode

With the presence of the DMA additional handshaking signals, the UART does not have to rely on internal status and level values to recognize the completion of a request and hence remove the request. Instead, the de-assertion of the DMA transmit and receive request is controlled by the assertion of the DMA acknowledge respectively.

When the UART is configured to have the additional DMA signals, the data flow (transfer lengths) responsibility falls on the UART and is controlled by the programmed burst transaction lengths & block size.

The extra handshaking signals are explained in the DMA flow below for a UART that is configured with FIFOs and Programmable THRE interrupt mode.

DMA controller must be configured in peripheral flow controller mode, because UART is a peripheral flow controller.

The UART must be programmed by the processor with the number of data items (block size) that are to be transmitted or received by the UART. This is programmed into the DEST_BLOCK_SIZE/SRC_BLOCK_SIZE field of the rUart_TDMACR & rUart_RDMACR registers of UART for Transmit FIFO and Receive FIFO, respectively. The block is broken into a number of transactions, each initiated by a request from the UART.

The DMA Controller & the UART must also be programmed with the number of data byte by burst transaction (in this case, UART FIFO entries) to be transferred for each DMA request. This is also known as the burst transaction length, and is programmed into:

DMA controller:

The DEST_MSIZE/SRC_MSIZE fields of the DMA CTL[n] register for Transmit FIFO and Receive FIFO, respectively.

UART:

The DEST_BURST_SIZE/SRC_BURST_SIZE fields of the rUart_TDMACR & rUart_RDMACR registers of UART for Transmit FIFO and Receive FIFO, respectively.

CAUTION

The burst transaction size must have the same values on DMA controller and UART.

(2) Transmit Watermark Level and Transmit FIFO Underflow

During UART serial transfers, transmit FIFO requests are made to the DMA whenever the number of entries in the transmit FIFO is less than or equal to the decoded level of the Transmit Empty Trigger (bUart_TET) of the rUart_FCR register.

This is known as the watermark level. The DMA responds by writing a burst of data to the transmit FIFO buffer, of length CTL[n].DEST_MSIZE = DEST_BURST_SIZE.

Data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously. That is, when the FIFO begins to empty another DMA request should be triggered. Otherwise the FIFO runs out of data (underflow). To prevent this condition, a software must set the watermark level correctly.

(3) Choosing the Transmit Watermark Level

Consider the example where the assumption is made:

DEST_BURST_SIZE = DMAC.CTL[n].DEST_MSIZE = FIFO_DEPTH − bUart_TET

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.

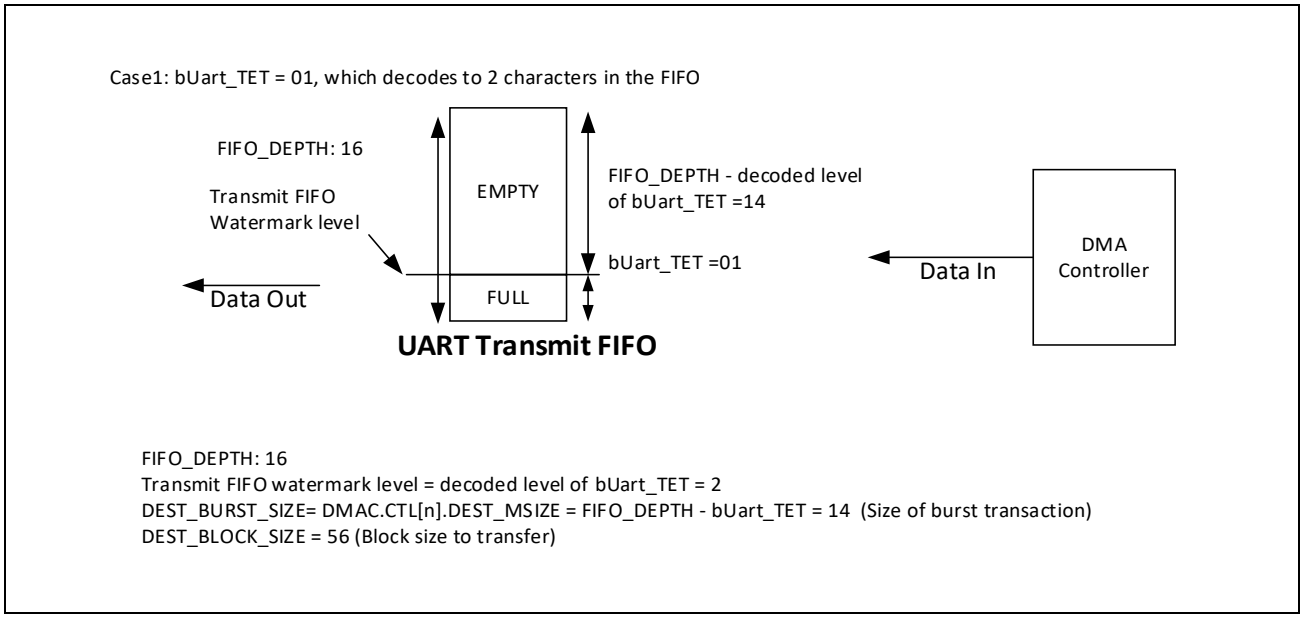

Figure 1.9 Case 1: Transmit Watermark Level

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

DEST_BLOCK_SIZE/DEST_BURST_SIZE = 56/14 = 4.

The number of burst transactions in the DMA block transfer is 4. But the watermark level, decoded level of bUart_TET, is quite low. Therefore, the probability of an UART underflow is high where the UART 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.

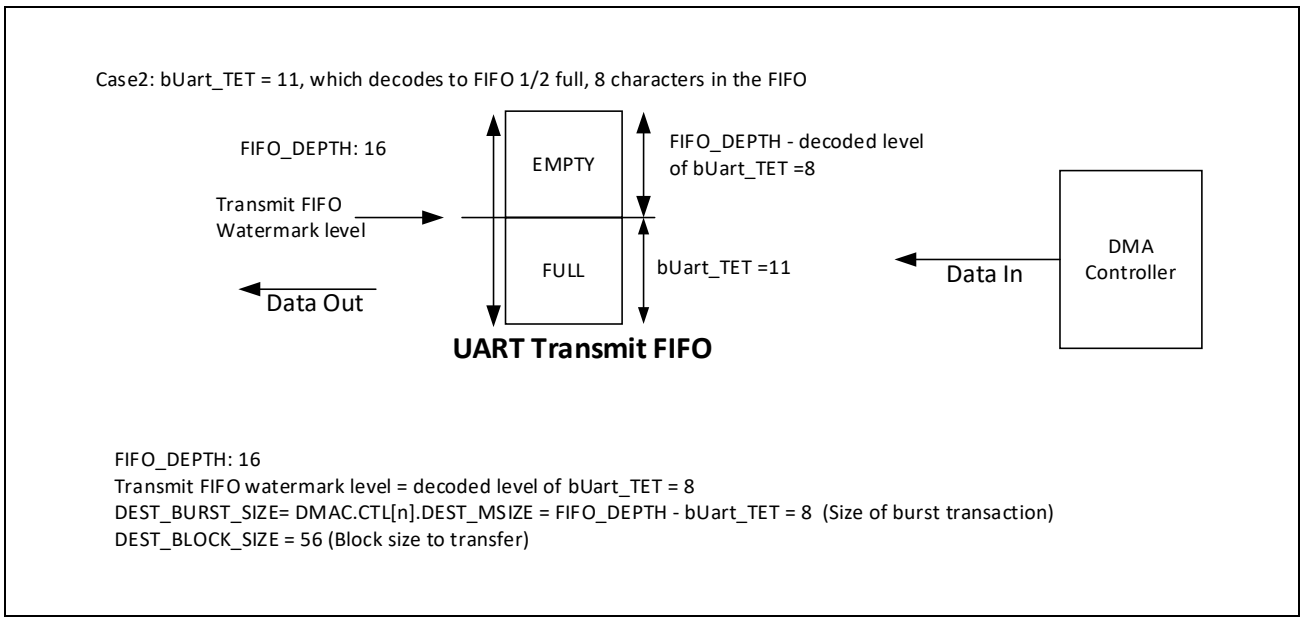

Figure 1.10 Case 2: Transmit Watermark Level

Number of burst transactions in Block n:

DEST_BLOCK_SIZE/DEST_BURST_SIZE = 56/8 = 7

In this block transfer, there are 7 destination burst transactions in a DMA block transfer. But the watermark level, decoded level of bUart_TET, is high. Therefore, the probability of an UART underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the UART 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 request 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 UART 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 bus layer, increases the rate at which the DMA controller can respond to burst transaction requests. This in turn allows the user to decrease the watermark level, which improves bus utilization without compromising the probability of an underflow occurring.

(4) Selecting DEST_MSIZE and Transmit FIFO Overflow

It may cause overflow when there is not enough space in the UART transmit FIFO to service the destination burst request.

Therefore, for optimal operation, we must configure:

- DMAC.CTL[n].DEST_MSIZE = DEST_BURST_SIZE = 4
- Set bUart_TET = 2'b11, FIFO 1/2 full

or

- DMAC.CTL[n].DEST_MSIZE = DEST_BURST_SIZE = 8
- Set bUart_TET = 2'b11, FIFO 1/2 full

CAUTION

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

(5) Receive Watermark Level and Receive FIFO Overflow

During UART serial transfers, receive FIFO requests are made to the DMAC whenever the number of entries in the receive FIFO is at or above the decoded level of Receiver Trigger (bUart_RCVR) of the rUart_FCR. This is known as the watermark level.

The DMAC responds by reading a burst of data in the receive FIFO buffer of length CTL[n].SRC_MSIZE = SRC_BURST_SIZE.

Data should be fetched by the DMAC 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.

(6) Choosing the Receive Watermark Level

Similar to choosing the transmit watermark level described earlier, the receive watermark level, decoded level of bUart_RCVR, should be set to minimize the probability of overflow. It is a tradeoff between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

(7) Selecting SRC_MSIZE and Receive FIFO Underflow

It may cause underflow when there is not enough data to service the source burst request.

Therefore, for optimal operation, we must configure:

- DMAC.CTL[n].SRC_MSIZE = SRC_BURST_SIZE = 4
- Set bUart_RCVR= 2'b01, FIFO 1/4 full

or

- DMAC.CTL[n].SRC_MSIZE = SRC_BURST_SIZE = 8
- Set bUart_RCVR= 2'b10, FIFO 1/2 full

CAUTION

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

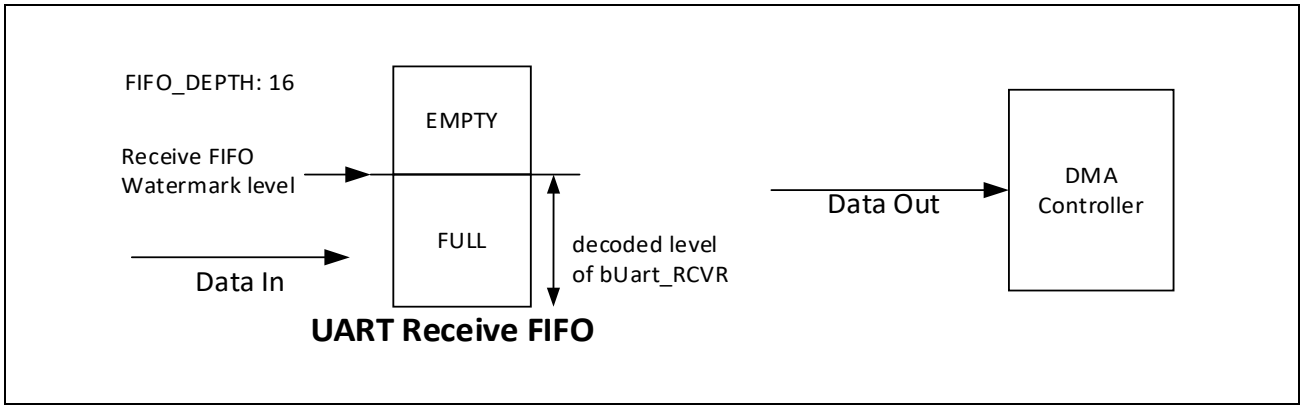

Figure 1.11 Case3: Receive Watermark Level

(8) Potential Deadlock Condition in UART with DMA Coupling on Receive Mode

If the DMA burst transaction length is identical to the UART Receive FIFO threshold (bUart_RCVR bit in bUart_RCVR bit of the rUart_FCR register), there is risk of a deadlock condition occurring when a character is received after UART_RTS_N is de-asserted.

The UART de-asserts UART_RTS_N when the Receive FIFO threshold is reached. However, it is possible the component at the other end of the line starts transmitting a new character before it detects the de-assertion of its UART_CTS_N input. When this happens, the character transmission completes normally, which means an extra character will be received and pushed into the Receive FIFO (unless it is already full).

At the same time that UART_RTS_N is de-asserted, the UART asserts receive DMA request, requesting a DMA burst transaction from the DMA controller. After the DMA controller completes this burst transaction (with length equal to the Receive FIFO threshold), there is one character left in the Receive FIFO, preventing UART_RTS_N from being asserted again.

The UART asserts single transaction DMA request, requesting a DMA single transaction to the DMA controller. However, unless it is operating in the single transaction region, the DMAC ignores single transaction requests.

A deadlock condition is then reached:

- The UART does not receive any extra characters because the UART_RTS_N signal is de-asserted. No data can be pushed into the Receive FIFO to fill it up to the threshold level again and generate a new burst transaction request from the DMAC, only single transaction requests can be generated.
- Unless it has reached the single transaction region, the DMAC ignores single transaction requests and does not read from the Rx FIFO. The Receive FIFO cannot be emptied, which prevents the UART_RTS_N signal from being asserted again.

This deadlock condition can be avoided if:

- The Receive FIFO threshold level is set to a value smaller than the DMA burst transaction size. This ensures that the Receive FIFO is always empty after a DMA burst transaction completes, regardless of whether or not one extra character is received and UART_RTS_N is asserted accordingly.

- The DMA block size is set to a value smaller than twice the DMA burst transaction length. This guarantees that the DMAC enters the single transaction region after the DMA burst transaction completes. It then accepts single transaction requests from the UART, allowing the Receive FIFO to be emptied.

This deadlock condition is not expected to occur frequently under normal operating conditions.

A timeout interrupt would be generated in this case, which can be used to detect the occurrence of this deadlock condition.

1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management

MODBUS Serial Line protocol is a Master-Slave protocol. This protocol takes place at level 2 of the OSI model.

A master-slave type system has one node (the master node) that issues explicit commands to one of the “slave” nodes and processes responses. Slave nodes will not typically transmit data without a request from the master node, and do not communicate with other slaves.

At the physical level, MODBUS over Serial Line systems may use different physical interfaces (RS485, RS232). Transmit Data Enable (DE) signals are controlled by software using UART [m]_RTS_N or GPIO.

Two Wire interface is the most common. As an add-on option, a TIA/EIA-485 (RS485) Four Wire interface may also be implemented. A TIA/EIA-232-E (RS232) serial interface may also be used as an interface, when only short point to point communication is required.

See typical two-wire interface in figure below.

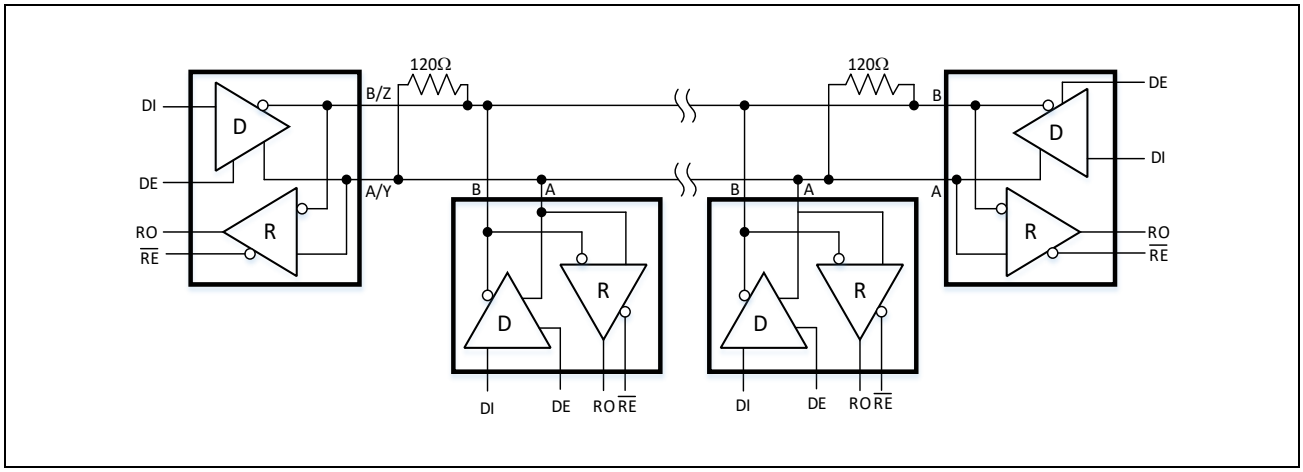

Figure 1.12 Typical Half-Duplex RS485 Network

A MODBUS message is placed by the transmitting device into a frame that has a known beginning and ending point. This allows devices that receive a new frame to begin at the start of the message, and to know when the message is completed. Partial messages must be detected and errors must be set as a result.

In RTU mode, message frames are separated by a silent interval of at least 3.5-character times. In the following sections, this time interval is called t3.5. See figure below.

The entire message frame must be transmitted as a continuous stream of characters.

If a silent interval of more than 1.5-character times (t1.5) occurs between two characters, the message frame is declared incomplete and should be discarded by the receiver.

[Remark]

The implementation of RTU reception driver may imply the management of a lot of interruptions due to the t1.5 and t3.5 timers. With high communication baud rates, this leads to a heavy CPU load.

Consequently, these two timers must be strictly respected when the baud rate is equal or lower than 19200 baud. For baud rates greater than 19200 baud, fixed values for the 2 timers should be used: it is recommended to use a value of 750 µs for the inter character time out (t1.5) and a value of 1.750 ms for inter frame delay (t3.5).

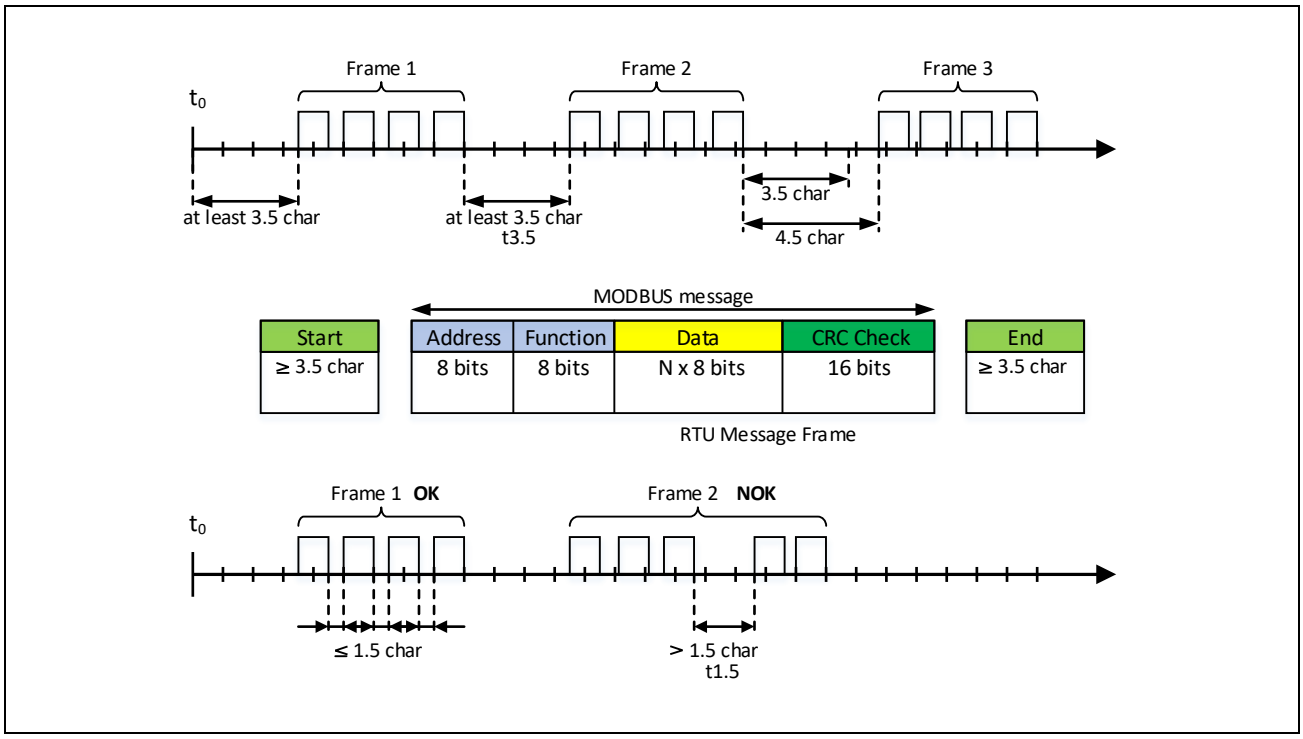

Figure 1.13 MODBUS Frame & Constraint Timing

CAUTION

To work properly this equation must be true:

UART_PCLK ≥ 4 × (UART_SCLK / baud rate divisor)

If UART_PCLK = 7.5 MHz and UART_SCLK = 83.33 MHz then

- The minimum baud rate divisor value = (83.33/7.5) × 4 = 45
- The maximum Baud rate is then UART_SCLK/(16 × 45) = 115 Kbaud

(1) Receiver Time-Out

Receiver Time-Out provides support in handling the interframe time (t1.5, t3.5) in MODBUS link.

There are two Receiver Time-Out.

This feature detects an idle condition (Silent Interval Detection) on the UART_RXD line.

The Time-Out delay period (during which the receiver waits for a new character) is programmed in the bUart_TO[n] (n = 0..1) field of the Time-Out Register (rUart_TO). This time is equal to rUart_TO[n] × “Baud clock period”.

When a Time-Out is detected (0 value on Time-Out Counter[n] with n = 0..1), the bUart_TIMEOUTInt[n] (n = 0..1) bit in the status register (rUart_STATUSTO) rises and can generate an interrupt (if not masked), thus indicating to the driver an end of frame.

See **Section 1.5.1.1, UART (RS232) Serial Protocol**.

The baud clock is set by the baud rate divisor (bUart_DLL and bUart_DLH).

CAUTION

If the rUart_DLL and/or rUart_DLH registers are written during a busy state, the Time-Out functionality will not work properly until the rUart_DLL and rUart_DLH registers are written during a no busy state.

If the bUart_TO[n] (n = 0..1) field is programmed at 0, the Receiver Time-Out is disabled and no Time-Out is detected. The bUart_TIMEOUTInt[n] (n = 0..1) bit in rUart_STATUSTO remains at 0.

Otherwise, the counter [n] loads an 8-bit counter with the value programmed in bUart_TO[n] (n = 0..1). This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, the bUart_TIMEOUTInt[n] (n = 0..1) bit in the status register (rUart_STATUSTO) rises and activates an interrupt (if not masked).

When the counter reaches 0, it remains locked until reception of a LOAD command (See synoptic below). The bUart_TIMEOUTStatus[n] (n = 0..1) bit in the status register (rUart_STATUSTO) is available to have a status on Time-Out Counter[n] (n = 0..1). This register is usually read by the Software driver during an interrupt service routine or polling:

- 1’b0: The Time-Out Counter value is different of “0”.
- 1’b1: The Time-Out Counter value is equal of “0”.

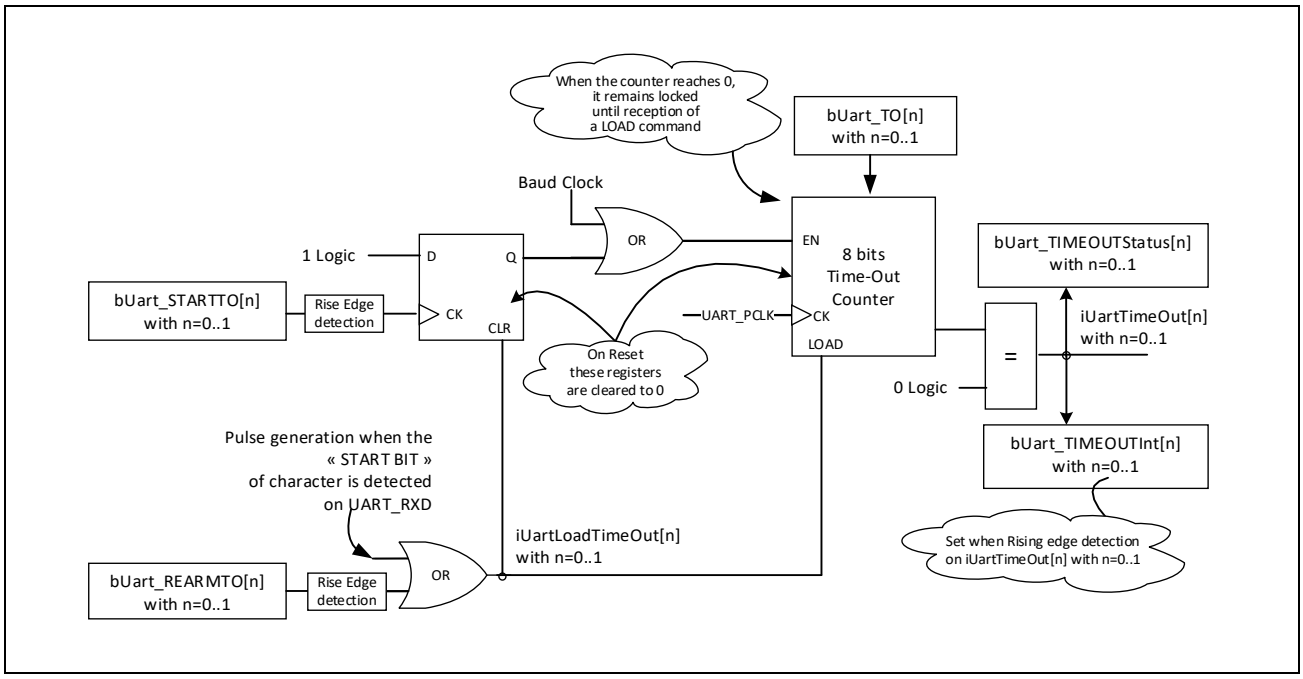

Figure 1.14 Receiver Time-Out Synoptic

When bUart_STARTTO[n] (n = 0..1) is set to 1 (Rising edge detection), the Time-Out counter is decremented to 0 by UART_PCLK and it stops counting until a first character is received. This feature enables waiting for the start of the next frame when the idle state on UART_RXD is detected.

When a start bit of a character is detected or bUart_REARMTO[n] (n = 0..1) is set to 1 (Rising edge detection), the Time-Out counter starts counting down from the value of bUart_TO[n] (n = 0..1).

See details in **Figure 1.16, Receiver & Transceiver Time-Out0..3, Timing Description**.

(2) Transceiver Time-Out

Transceiver Time-Out provides support in handling the interframe time (t1.5, t3.5) in MODBUS link.

There are two Transceiver Time-Out.

This feature detects an idle condition (Silent Interval Detection) on the UART_TXD line.

The Time-Out delay period (during which the transceiver waits for a new character) is programmed in the bUart_TO[n] (n = 2..3) field of the Time-Out Register (rUart_TO). This time is equal to rUart_TO[n] × “Baud clock period”.

When a Time-Out is detected (0 value on Time-Out Counter[n] with n = 2..3), the bUart_TIMEOUTInt[n] (n = 2..3) bit in the status register (rUart_STATUSTO) rises and can generate an interrupt (if not masked), thus indicating to the driver an end of frame.

If the bUart_TO[n] (n = 2..3) field is programmed at 0, the clocking of Time-Out Counter is stopped, the counter keeps current value. The bUart_TIMEOUTInt[n] (n = 2..3) bit in rUart_STATUSTO keeps current value.

Otherwise, the counter [n] loads an 8-bit counter with the value programmed in bUart_TO[n] (n = 2..3). This counter is decremented at each bit period and reloaded each time a new character is transmitted. If the counter reaches 0, the bUart_TIMEOUTInt[n] (n = 2..3) bit in the status register (rUart_STATUSTO) rises and activates an interrupt (if not masked).

When the counter [n] reaches 0, it remains locked until reception of a LOAD command (See synoptic below). The bUart_TIMEOUTStatus[n] (n = 2..3) bit in the status register (rUart_STATUSTO) is available to have a status on Time-Out Counter [n] (n = 2..3):

- 1’b0: The Time-Out Counter value is different of “0”.
- 1’b1: The Time-Out Counter value is equal of “0”.

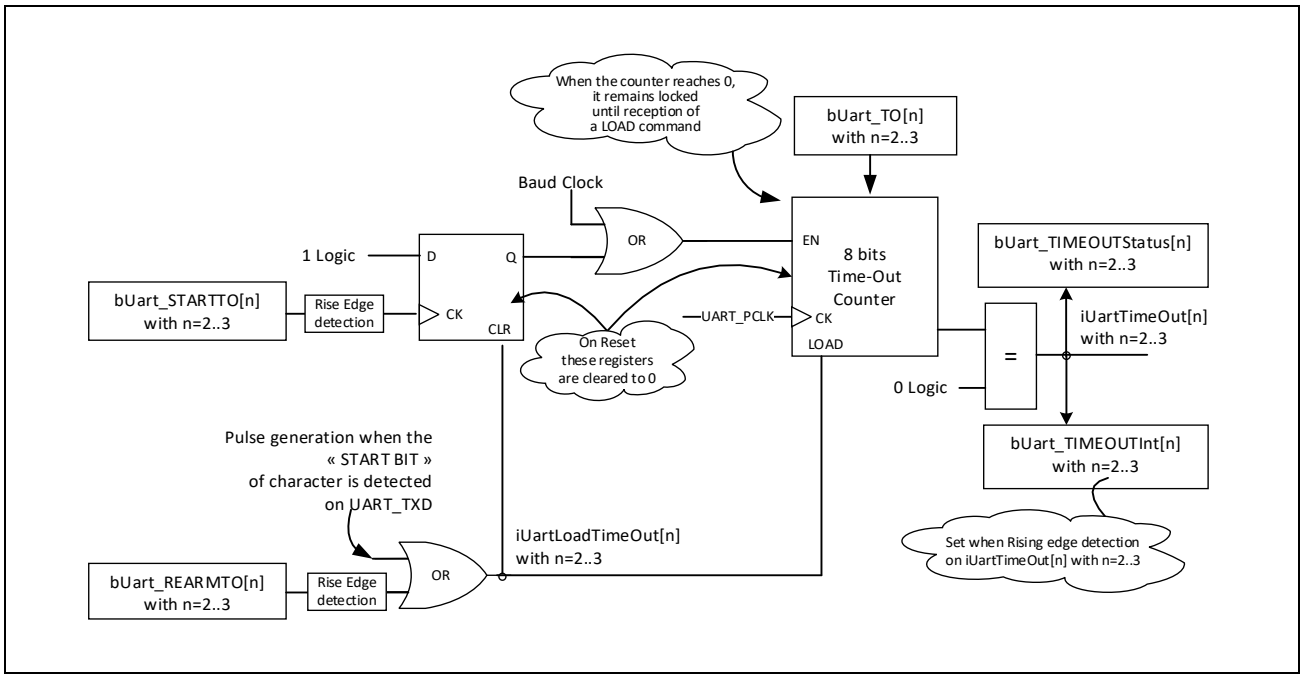

Figure 1.15 Transceiver Time-Out Synoptic

When bUart_STARTTO[n] (n = 2..3) is set to 1 (Rising edge detection), the Time-Out counter is decremented to 0 by UART_PCLK and it stops counting until a first character is transmitted. This feature enables waiting for the start of the next frame when the idle state on UART_TXD is detected.

When a start bit of a character is detected or bUart_REARMTO[n] (n = 2..3) is set to 1 (Rising edge detection), the Time-Out counter starts counting down from the value of bUart_TO[n] (n = 2..3).

See details in **Figure 1.16, Receiver & Transceiver Time-Out0..3, Timing Description**.

(3) Time-out counter timing

The figure below describes the timing of Time-Out Counter[n] (n = 0..3).

When the start bit of a character is detected, the Time-Out counter loads the value of rUart_TO register and starts counting down.

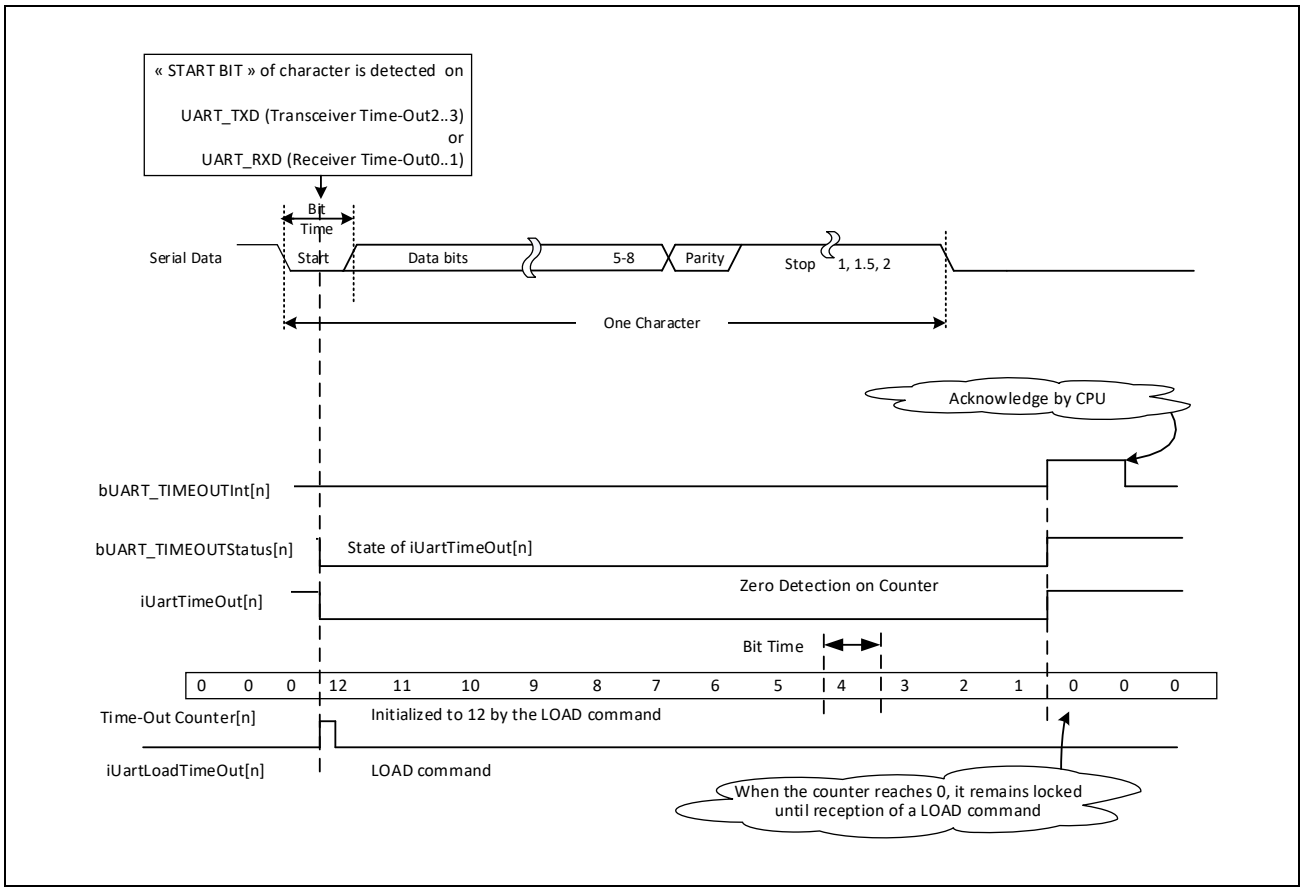

Figure 1.16 Receiver & Transceiver Time-Out0..3, Timing Description

1.5.2 Usage Notes

When we will use “Two Wire” interface, we have need to manage the “DE (Data Enable)” signal in Half-Duplex mode.

Data Enable signal can be connected using internal UART_RTS_N signal. The output from internal iUART_DE signal is not available.

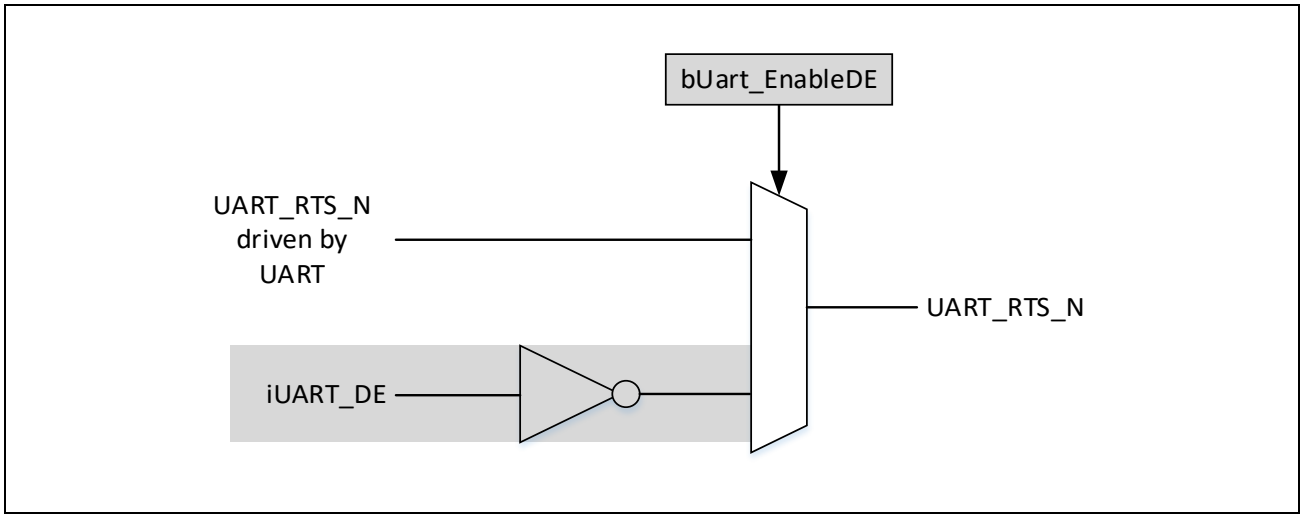

Figure 1.17 UART_RTS_N Management in Half-Duplex Mode

Section 2 SPI

2.1 Overview

The RZ/N1 provides 4 blocks dedicated for SPI Master and 2 blocks dedicated for SPI slave.

- Master mode
  - SPI1, SPI2, SPI3, SPI4 only
- Slave mode
  - SPI5, SPI6 only
- Transmit FIFO (TX FIFO) 16 words of 16 bits
- Receive FIFO (RX FIFO) 16 words of 16 bits.
- Serial clock bit rate, dynamic control of the serial bit rate of the data transfer.
  - Master mode only
- Programmable data-size for frames (from 4 to 16 bits)
- Slave selects number
  - 4 (SPI Master mode, SPI1, SPI2, SPI3, SPI4)
  - 1 (SPI Slave mode, SPI5, SPI6)
- Selectable serial interface operation
  - Motorola SPI
  - Texas Instruments Synchronous Serial Protocol
  - National Semiconductor Microwire
- DMA coupling
  - Peripheral flow controller mode
  - 2 DMA channel available (One for transmit, one for receive)

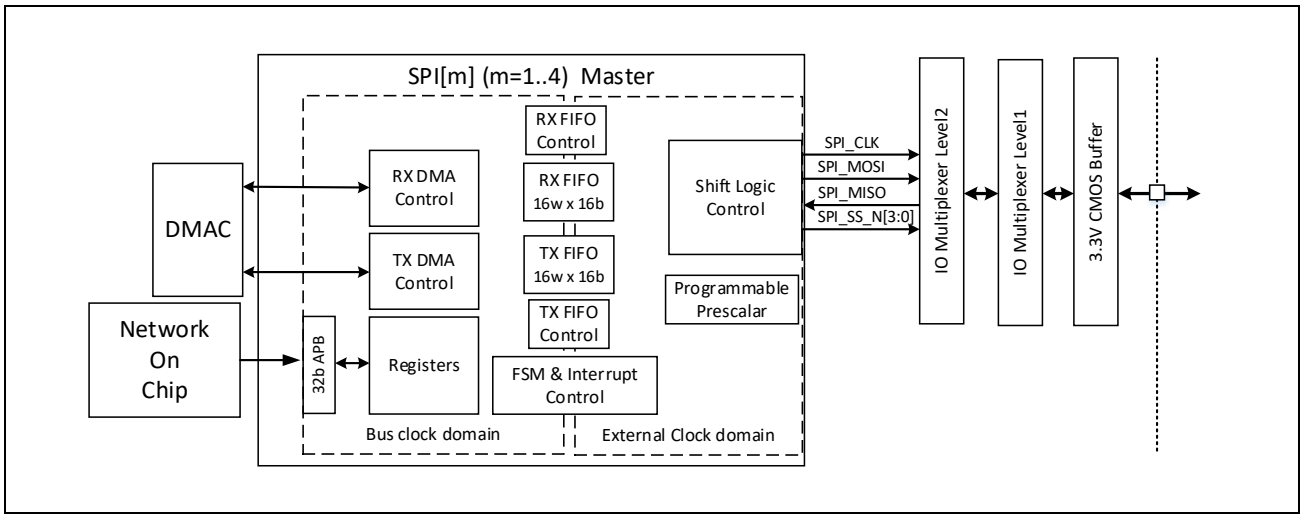

Figure 2.1 SPI Master Synoptic

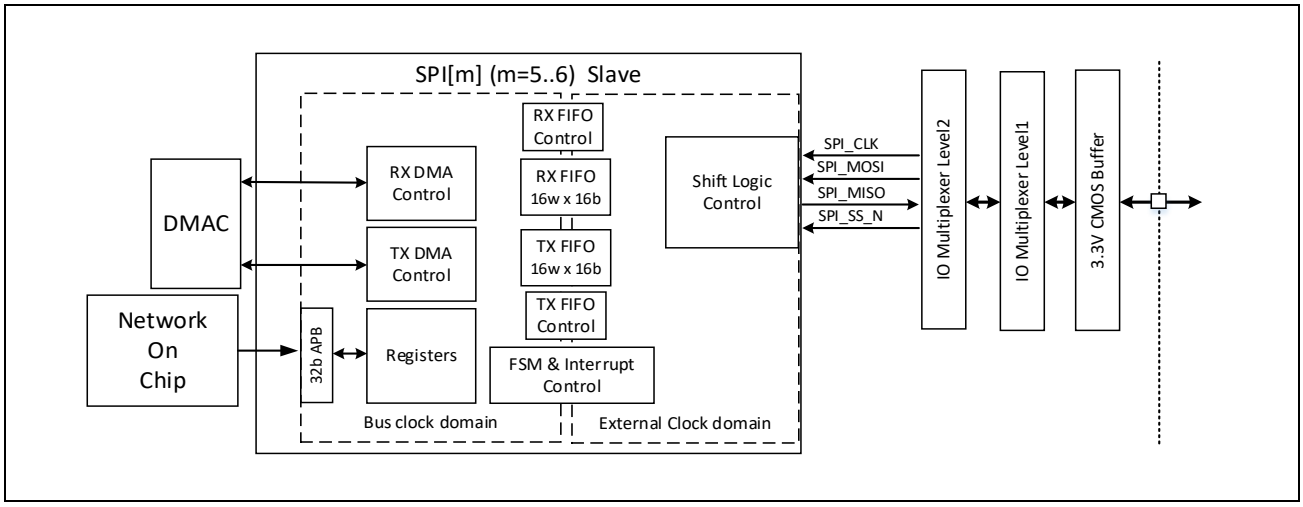

Figure 2.2 SPI Slave Synoptic

2.2 Signal Interfaces

Signal Name	Input Output	Description
Clock		
SPI[m]_PCLK	Input	Internal bus clock (APB)
SPI[m]_SCLK	Input	Serial reference clock*1
Interrupt		
SPI[m]_Int	Output	Level sensitive interrupt output, Active High
External Signal (SPI Master Mode)		
SPI[m]_CLK (M)	Output	Serial clock
SPI[m]_MOSI (M)	Output	Master transmit data
SPI[m]_MISO (M)	Input	Master receive data
SPI[m]_SS_N[3:0] (M)	Output	Slave selection (Chip select), Active Low
External Signal (SPI Slave Mode)		
SPI[m]_CLK (S)	Input	Serial clock
SPI[m]_MOSI (S)	Input	Slave receive data
SPI[m]_MISO (S)	Output	Slave transmit data
SPI[m]_SS_N (S)	Input	Slave selection (Chip select), Active Low

Note: m = 1..6.
Index removed style is used in this chapter.
Ex) SPI_PCLK

Note 1. Input frequency from 7.81 MHz to 125 MHz
Different frequency settings are possible for SPI[m]_SCLK (m = 1..4) and SPI[m]_SCLK (m = 5..6).

2.3 Register Map

2.3.1 Register Map SPI1 (Master)

Table 2.1 Register Map SPI1 (Master)

Address	Register Symbol	Register Name
5000 5000h	rSpi_CTRLR0	Control Register 0
5000 5004h	rSpi_CTRLR1	Control Register 1
5000 5008h	rSpi_SSIENR	Enable Register
5000 500Ch	rSpi_MWCR	Microwire Control Register
5000 5010h	rSpi_SER	Slave Enable Register
5000 5014h	rSpi_BAUDR	Baud Rate Select
5000 5018h	rSpi_TXFTLR	Transmit FIFO Threshold Level
5000 501Ch	rSpi_RXFTLR	Receive FIFO Threshold Level
5000 5020h	rSpi_TXFLR	Transmit FIFO Level Register
5000 5024h	rSpi_RXFLR	Receive FIFO Level Register
5000 5028h	rSpi_SR	Status Register
5000 502Ch	rSpi_IMR	Interrupt Mask Register
5000 5030h	rSpi_ISR	Interrupt Status Register
5000 5034h	rSpi_RISR	Raw Interrupt Status Register
5000 5038h	rSpi_TXOICR	Transmit FIFO Overflow Interrupt Clear Register
5000 503Ch	rSpi_RXOICR	Receive FIFO Overflow Interrupt Clear Register
5000 5040h	rSpi_RXUICR	Receive FIFO Underflow Interrupt Clear Register
5000 5048h	rSpi_ICR	Interrupt Clear Register
5000 504Ch	rSpi_DMACR	DMA Control Register
5000 5050h	rSpi_DMATDLR	DMA Transmit Data Level
5000 5054h	rSpi_DMARDLR	DMA Receive Data Level
5000 5060h	rSpi_DR	Data Register
5000 50F0h	rSpi_RX_SAMPLE_DLY	RXD Sample Delay Register
5000 5100h	rSpi_TDMACR	DMA Control Register in Transmit Mode
5000 5104h	rSpi_RDMACR	DMA Control Register in Receive Mode

2.3.2 Register Map SPI2 (Master)

Table 2.2 Register Map SPI2 (Master)

Address	Register Symbol	Register Name
5000 6000h	rSpi_CTRLR0	Control Register 0
5000 6004h	rSpi_CTRLR1	Control Register 1
5000 6008h	rSpi_SSIENR	Enable Register
5000 600Ch	rSpi_MWCR	Microwire Control Register
5000 6010h	rSpi_SER	Slave Enable Register
5000 6014h	rSpi_BAUDR	Baud Rate Select
5000 6018h	rSpi_TXFTLR	Transmit FIFO Threshold Level
5000 601Ch	rSpi_RXFTLR	Receive FIFO Threshold Level
5000 6020h	rSpi_TXFLR	Transmit FIFO Level Register
5000 6024h	rSpi_RXFLR	Receive FIFO Level Register
5000 6028h	rSpi_SR	Status Register
5000 602Ch	rSpi_IMR	Interrupt Mask Register
5000 6030h	rSpi_ISR	Interrupt Status Register
5000 6034h	rSpi_RISR	Raw Interrupt Status Register
5000 6038h	rSpi_TXOICR	Transmit FIFO Overflow Interrupt Clear Register
5000 603Ch	rSpi_RXOICR	Receive FIFO Overflow Interrupt Clear Register
5000 6040h	rSpi_RXUICR	Receive FIFO Underflow Interrupt Clear Register
5000 6048h	rSpi_ICR	Interrupt Clear Register
5000 604Ch	rSpi_DMACR	DMA Control Register
5000 6050h	rSpi_DMATDLR	DMA Transmit Data Level
5000 6054h	rSpi_DMARDLR	DMA Receive Data Level
5000 6060h	rSpi_DR	Data Register
5000 60F0h	rSpi_RX_SAMPLE_DLY	RXD Sample Delay Register
5000 6100h	rSpi_TDMACR	DMA Control Register in Transmit Mode
5000 6104h	rSpi_RDMACR	DMA Control Register in Receive Mode

2.3.3 Register Map SPI3 (Master)

Table 2.3 Register Map SPI3 (Master)

Address	Register Symbol	Register Name
5000 7000h	rSpi_CTRLR0	Control Register 0
5000 7004h	rSpi_CTRLR1	Control Register 1
5000 7008h	rSpi_SSIENR	Enable Register
5000 700Ch	rSpi_MWCR	Microwire Control Register
5000 7010h	rSpi_SER	Slave Enable Register
5000 7014h	rSpi_BAUDR	Baud Rate Select
5000 7018h	rSpi_TXFTLR	Transmit FIFO Threshold Level
5000 701Ch	rSpi_RXFTLR	Receive FIFO Threshold Level
5000 7020h	rSpi_TXFLR	Transmit FIFO Level Register
5000 7024h	rSpi_RXFLR	Receive FIFO Level Register
5000 7028h	rSpi_SR	Status Register
5000 702Ch	rSpi_IMR	Interrupt Mask Register
5000 7030h	rSpi_ISR	Interrupt Status Register
5000 7034h	rSpi_RISR	Raw Interrupt Status Register
5000 7038h	rSpi_TXOICR	Transmit FIFO Overflow Interrupt Clear Register
5000 703Ch	rSpi_RXOICR	Receive FIFO Overflow Interrupt Clear Register
5000 7040h	rSpi_RXUICR	Receive FIFO Underflow Interrupt Clear Register
5000 7048h	rSpi_ICR	Interrupt Clear Register
5000 704Ch	rSpi_DMACR	DMA Control Register
5000 7050h	rSpi_DMATDLR	DMA Transmit Data Level
5000 7054h	rSpi_DMARDLR	DMA Receive Data Level
5000 7060h	rSpi_DR	Data Register
5000 70F0h	rSpi_RX_SAMPLE_DLY	RXD Sample Delay Register
5000 7100h	rSpi_TDMACR	DMA Control Register in Transmit Mode
5000 7104h	rSpi_RDMACR	DMA Control Register in Receive Mode

2.3.4 Register Map SPI4 (Master)

Table 2.4 Register Map SPI4 (Master)

Address	Register Symbol	Register Name
5000 8000h	rSpi_CTRLR0	Control Register 0
5000 8004h	rSpi_CTRLR1	Control Register 1
5000 8008h	rSpi_SSIENR	Enable Register
5000 800Ch	rSpi_MWCR	Microwire Control Register
5000 8010h	rSpi_SER	Slave Enable Register
5000 8014h	rSpi_BAUDR	Baud Rate Select
5000 8018h	rSpi_TXFTLR	Transmit FIFO Threshold Level
5000 801Ch	rSpi_RXFTLR	Receive FIFO Threshold Level
5000 8020h	rSpi_TXFLR	Transmit FIFO Level Register
5000 8024h	rSpi_RXFLR	Receive FIFO Level Register
5000 8028h	rSpi_SR	Status Register
5000 802Ch	rSpi_IMR	Interrupt Mask Register
5000 8030h	rSpi_ISR	Interrupt Status Register
5000 8034h	rSpi_RISR	Raw Interrupt Status Register
5000 8038h	rSpi_TXOICR	Transmit FIFO Overflow Interrupt Clear Register
5000 803Ch	rSpi_RXOICR	Receive FIFO Overflow Interrupt Clear Register
5000 8040h	rSpi_RXUICR	Receive FIFO Underflow Interrupt Clear Register
5000 8048h	rSpi_ICR	Interrupt Clear Register
5000 804Ch	rSpi_DMACR	DMA Control Register
5000 8050h	rSpi_DMATDLR	DMA Transmit Data Level
5000 8054h	rSpi_DMARDLR	DMA Receive Data Level
5000 8060h	rSpi_DR	Data Register
5000 80F0h	rSpi_RX_SAMPLE_DLY	RXD Sample Delay Register
5000 8100h	rSpi_TDMACR	DMA Control Register in Transmit Mode
5000 8104h	rSpi_RDMACR	DMA Control Register in Receive Mode

2.3.5 Register Map SPI5 (Slave)

Table 2.5 Register Map SPI5 (Slave)

Address	Register Symbol	Register Name
5000 9000h	rSpi_CTRLR0	Control Register 0
5000 9008h	rSpi_SSIENR	Enable Register
5000 900Ch	rSpi_MWCR	Microwire Control Register
5000 9018h	rSpi_TXFTLR	Transmit FIFO Threshold Level
5000 901Ch	rSpi_RXFTLR	Receive FIFO Threshold Level
5000 9020h	rSpi_TXFLR	Transmit FIFO Level Register
5000 9024h	rSpi_RXFLR	Receive FIFO Level Register
5000 9028h	rSpi_SR	Status Register
5000 902Ch	rSpi_IMR	Interrupt Mask Register
5000 9030h	rSpi_ISR	Interrupt Status Register
5000 9034h	rSpi_RISR	Raw interrupt Status Register
5000 9038h	rSpi_TXOICR	Transmit FIFO Overflow Interrupt Clear Register
5000 903Ch	rSpi_RXOICR	Receive FIFO Overflow Interrupt Clear Register
5000 9040h	rSpi_RXUICR	Receive FIFO Underflow Interrupt Clear Register
5000 9048h	rSpi_ICR	Interrupt Clear Register
5000 904Ch	rSpi_DMACR	DMA Control Register
5000 9050h	rSpi_DMATDLR	DMA Transmit Data Level
5000 9054h	rSpi_DMARDLR	DMA Receive Data Level
5000 9060h	rSpi_DR	Data Register
5000 9100h	rSpi_TDMACR	DMA Control Register in Transmit Mode
5000 9104h	rSpi_RDMACR	DMA Control Register in Receive Mode

2.3.6 Register Map SPI6 (Slave)

Table 2.6 Register Map SPI6 (Slave)

Address	Register Symbol	Register Name
5000 A000h	rSpi_CTRLR0	Control Register 0
5000 A008h	rSpi_SSIENR	Enable Register
5000 A00Ch	rSpi_MWCR	Microwire Control Register
5000 A018h	rSpi_TXFTLR	Transmit FIFO Threshold Level
5000 A01Ch	rSpi_RXFTLR	Receive FIFO Threshold Level
5000 A020h	rSpi_TXFLR	Transmit FIFO Level Register
5000 A024h	rSpi_RXFLR	Receive FIFO Level Register
5000 A028h	rSpi_SR	Status Register
5000 A02Ch	rSpi_IMR	Interrupt Mask Register
5000 A030h	rSpi_ISR	Interrupt Status Register
5000 A034h	rSpi_RISR	Raw Interrupt Status Register
5000 A038h	rSpi_TXOICR	Transmit FIFO Overflow Interrupt Clear Register
5000 A03Ch	rSpi_RXOICR	Receive FIFO Overflow Interrupt Clear Register
5000 A040h	rSpi_RXUICR	Receive FIFO Underflow Interrupt Clear Register
5000 A048h	rSpi_ICR	Interrupt Clear Register
5000 A04Ch	rSpi_DMACR	DMA Control Register
5000 A050h	rSpi_DMATDLR	DMA Transmit Data Level
5000 A054h	rSpi_DMARDLR	DMA Receive Data Level
5000 A060h	rSpi_DR	Data Register
5000 A100h	rSpi_TDMACR	DMA Control Register in Transmit Mode
5000 A104h	rSpi_RDMACR	DMA Control Register in Receive Mode

2.4 Register Description

2.4.1 rSpi_CTRLR0 — Control Register 0

This register controls the serial data transfer. It is impossible to write to this register when the SPI controller is enabled by bSpi_SSIENR.

Address: 5000 5000h (SPI1)
5000 6000h (SPI2)
5000 7000h (SPI3)
5000 8000h (SPI4)
5000 9000h (SPI5)
5000 A000h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bSpi_CFS				bSpi_SRL	bSpi_SLV_OE	bSpi_TMOD		bSpi_SCPOL	bSpi_SCPH	bSpi_FRF		bSpi_DFS			
Value after reset	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1

Table 2.7 rSpi_CTRLR0 Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Read as 0	R
b15 to b12	bSpi_CFS	Control Frame Size Selects the length of the control word for the Microwire frame format. 4’b0000: 1-bit control word 4’b0001: 2-bit control word ... ... 4’b1110: 15-bit control word 4’b1111: 16-bit control word	R/W
b11	bSpi_SRL	Shift Register Loop Used for testing purposes only. When set, connects the transmit shift register output to the receive shift register input. Can be used in both SPI slave and master. 0: Normal Mode Operation 1: Test Mode Operation When SPI slave is configured as Test Mode, the SPI_CLK and SPI_SS_N signals must be provided by an external source. In this mode, the slave cannot generate these signals because there is nothing to which to loop back.	R/W
b10	bSpi_SLV_OE	Slave Output Enable (SPI slave only) This bit enables or disables the setting of the SPI_MISO output from the SPI slave. When bSpi_SLV_OE = 1, the SPI_MISO output can never be active, a floating state is always present. This is useful when the master transmits in broadcast mode (master transmits data to all slave devices). Only one slave may respond with data on the master SPI_MISO line. This bit is enabled after reset and must be disabled by Software (when broadcast mode is used), if you do not want this device to respond with data. 1’b0: Slave SPI_MISO is enabled 1’b1: Slave SPI_MISO is disabled	R/W

Table 2.7 rSpi_CTRLR0 Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b9, b8	bSpi_TMOD	Transfer Mode Selects the mode of transfer for serial communication. This field does not affect the duplex setting. Only indicates whether the receive or transmit data are valid. In Transmit Only Mode, data received from the external device is not valid and is not stored in the receive FIFO memory, it is overwritten on the next transfer. In Receive Only Mode, transmitted data are not valid. After the first write to the transmit FIFO, the same control word is retransmitted for the duration of the transfer. In Transmit and Receive Mode, both transmit and receive data are valid. The transfer continues until the transmit FIFO is empty. Data received from the external device are stored into the receive FIFO memory, where it can be accessed by the host processor. In EEPROM Read Mode, receive data is not valid while control data is being transmitted. When all control data is sent to the EEPROM, receive data becomes valid and transmit data becomes invalid. All data in the transmit FIFO is considered control data in this mode.This transfer mode is only valid when the SPI controller is a master. 2'b00: Transmit and Receive 2'b01: Transmit Only 2'b10: Receive Only 2'b11: EEPROM Read	R/W
b7	bSpi_SCPOL	Serial Clock Polarity Valid when the frame format (bSpi_FRF) is set to Motorola SPI. Used to select the polarity of the inactive serial clock, which is held inactive when the SPI controller is not actively transferring data on the serial bus. 1'b0: Inactive state of the serial clock is low 1'b1: Inactive state of the serial clock is high	R/W
b6	bSpi_SCPH	Serial Clock Phase Valid when the frame format (bSpi_FRF) is set to Motorola SPI. The serial clock phase selects the relationship of the serial clock with the slave select signal. When bSpi_SCPH = 0, data are captured on the first edge of the serial clock. When bSpi_SCPH = 1, the serial clock starts toggling one cycle after the slave select line is activated, and data are captured on the second edge of the serial clock. 1'b0: The first serial clock toggles at middle of the data bit 1'b1: The first serial clock toggles at start of the data bit	R/W
b5, b4	bSpi_FRF	Frame Format 2'b00: Motorola Serial Peripheral Interface 2'b01: Texas Instruments Synchronous Serial Protocol 2'b10: National Semiconductor Microwire 2'b11: Reserved	R/W
b3 to b0	bSpi_DFS	Data Frame Size Selects the data frame length. When the data frame size is programmed to be less than 16 bits, the receive data are automatically right-justified by the receive logic, with the upper bits of the receive FIFO zero padded. You must right-justify transmit data before writing into the transmit FIFO. The transmit logic ignores the upper unused bits when transmitting the data. 4'b0000: Reserved - undefined operation 4'b0001: Reserved - undefined operation 4'b0010: Reserved - undefined operation 4'b0011: 4-bit serial data transfer 4'b0100: 5-bit serial data transfer ... ... 4'b1110: 15-bit serial data transfer 4'b1111: 16-bit serial data transfer	R/W

2.4.2 rSpi_CTRLR1 — Control Register 1

It is impossible to write to this register when the SPI controller is enabled by bSpi_SSIENR.

CAUTION

This register exists in SPI Master only. Data size is up to 64 KB in continuous transfer regardless of frame size.

Address: 5000 5004h (SPI1)
5000 6004h (SPI2)
5000 7004h (SPI3)
5000 8004h (SPI4)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bSpi_NDF															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.8 rSpi_CTRLR1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Read as 0.	R
b15 to b0	bSpi_NDF	Number of Data Frames When bSpi_TMOD = 2'b10 or bSpi_TMOD = 2'b11, this register field sets the number of data frames to be continuously received by SPI controller. The SPI controller continues to receive serial data until the number of data frames received is equal to this register value plus 1, which enables you to receive up to 64 KB of data in a continuous transfer.	R/W

2.4.3 rSpi_SSIENR — Enable Register

Address: 5000 5008h (SPI1)
5000 6008h (SPI2)
5000 7008h (SPI3)
5000 8008h (SPI4)
5000 9008h (SPI5)
5000 A008h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_SSIENR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.9 rSpi_SSIENR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Read as 0.	R
b0	bSpi_SSIENR	SPI controller Enable Enables and disables all SPI operations. When disabled, all serial transfers are halted immediately. Transmit and receive FIFO buffers are cleared. It is impossible to program some of the SPI control registers when enabled. 1’b0: Disable 1’b1: Enable	R/W

2.4.4 rSpi_MWCR — Microwire Control Register

It is impossible to write to this register when the SPI controller is enabled by bSpi_SSIENR.

Address: 5000 500Ch (SPI1)
5000 600Ch (SPI2)
5000 700Ch (SPI3)
5000 800Ch (SPI4)
5000 900Ch (SPI5)
5000 A00Ch (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_MWHS	bSpi_MDD	bSpi_MWMOD
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.10 rSpi_MWCR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b3	Reserved	Read as 0.	R
b2	bSpi_MWHS	Microwire Handshaking (SPI master only) Used to enable and disable the “busy/ready” handshaking interface for the Microwire protocol. When enabled, the SPI controller checks for a ready status from the target slave, after the transfer of the last data/control bit, before clearing the BUSY status in the rSpi_SR register. 0: handshaking interface is disabled 1: handshaking interface is enabled	R/W
b1	bSpi_MDD	Microwire Control Defines the direction of the data word when the Microwire serial protocol is used. 0: the data word is received by the SPI controller from the external serial device. 1: the data word is transmitted from the SPI controller to the external serial device.	R/W
b0	bSpi_MWMOD	Microwire Transfer Mode Defines whether the Microwire transfer is sequential or no-sequential. When sequential mode is used, only one control word is needed to transmit or receive a block of data words. When non-sequential mode is used, there must be a control word for each data word that is transmitted or received. 0: non-sequential transfer 1: sequential transfer	R/W

2.4.5 rSpi_SER — Slave Enable Register

Do not write to this register when SPI controller is busy. See **Section 2.5.3, Control Slave Select Line by Hardware or Software Mode**.

CAUTION

This register exists in SPI Master only

Address: 5000 5010h (SPI1)
5000 6010h (SPI2)
5000 7010h (SPI3)
5000 8010h (SPI4)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bSpi_CtrlSS				bSpi_SoftwareSS				bSpi_HardwareSS			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.11 rSpi_SER Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b12	Reserved	Read as 0.	R
b11 to b8	bSpi_CtrlSS	Slave Select Mode Enable The register enables the individual slave select output lines from the SPI master. Up to 4 slave select output signals are available on the SPI master. Each bSpi_CtrlSS[3:0] in register allow the control of slave respective select lines SPI_SS_N[3:0] from the SPI master in hardware or Software mode. 0: Hardware mode • These lines SPI_SS_N[3:0] are controlled by bSpi_HardwareSS[3:0] bits respectively from serializer module and activated when a serial transfer begins. 1: Software mode • These lines SPI_SS_N[3:0] are directly controlled by bSpi_SoftwareSS[3:0] bits respectively.	R/W
b7 to b4	bSpi_SoftwareSS	Software mode: Slave Select Enable Flag The register enables the individual slave select output lines from the SPI master. Up to 4 slave select output signals are available on the SPI master. Each bSpi_SoftwareSS[3:0] in this register corresponds to a slave respective select line SPI_SS_N[3:0] from the SPI master. This mode are activated when bSpi_CtrlSS[3:0] is set to 1 respectively for each line. The corresponding slave select SPI_SS_N[3:0] line from the master are directly connected to a peripheral device and is controlled by bSpi_SoftwareSS[3:0] respectively. ***Caution***) You should select one slave in bSpi_HardwareSS[3:0] to start transfer. 1: Selected 0: Not Selected	R/W

Table 2.11 rSpi_SER Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3 to b0	bSpi_HardwareSS	Hardware mode: Slave Select Enable Flag The register enables the individual slave select output lines from the SPI master. Up to 4 slave select output signals are available on the SPI master. Each bSpi_HardwareSS[3:0] in this register corresponds to a respective slave select line SPI_SS_N[3:0] from the SPI master. This mode are activated when bSpi_CtrlSS[3:0] is clear to 0 respectively for each line. In this mode, when a bit in this register is set to 1, the corresponding slave select SPI_SS_N[3:0] are respectively and activated when a serial transfer begins. These lines are controlled by the serializer module. It should be noted that setting or clearing bits in this register have no effect on the corresponding slave select outputs until a transfer is started. Before beginning a transfer, you should enable the bit in this register that corresponds to the slave device with which the master wants to communicate. When not operating in broadcast mode (master transmits data to all slave devices), only one bit in this field should be set. 1: Selected 0: Not Selected	R/W

2.4.6 rSpi_BAUDR — Baud Rate Select

It is impossible to write to this register when the SPI controller is enabled by bSpi_SSIENR.

CAUTION

This register exists in SPI Master only.

Address: 5000 5014h (SPI1)
5000 6014h (SPI2)
5000 7014h (SPI3)
5000 8014h (SPI4)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bSpi_SCKDV															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.12 rSpi_BAUDR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Read as 0.	R
b15 to b0	bSpi_SCKDV	SPI Clock Divider The 16-bit field in this register defines the SPI_CLK divider value. The LSB for this field is always set to 0 and is unaffected by a write operation, which ensures an even value is held in this register. If the value is 0, the serial output clock (SPI_CLK) is disabled. The frequency of the SPI_CLK is derived from the following equation: Frequency (SPI_CLK) = Frequency (SPI_SCLK) / bSpi_SCKDV Where bSpi_SCKDV is any even value between 2 and 65534.	R/W

2.4.7 rSpi_TXFTLR — Transmit FIFO Threshold Level

This register controls the threshold value for the transmit FIFO memory.

Do not write to this register when the SPI controller is enabled by bSpi_SSIENR.

Address: 5000 5018h (SPI1)
5000 6018h (SPI2)
5000 7018h (SPI3)
5000 8018h (SPI4)
5000 9018h (SPI5)
5000 A018h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_TFT			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.13 rSpi_TXFTLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b4	Reserved	Read as 0.	R
b3 to b0	bSpi_TFT	Transmit FIFO Threshold Controls the level of entries (or below) at which the transmit FIFO controller triggers an interrupt. If you attempt to set this value greater than or equal to the depth of the FIFO, this field is not written and retains its current value. When the number of transmit FIFO entries is less than or equal to this value, the transmit FIFO empty interrupt is triggered. 4'd0: iSpi_TXE_Int is asserted when 0 data entries are present in transmit FIFO. 4'd1: iSpi_TXE_Int is asserted when 1 or less data entries are present in transmit FIFO. ... ... 4'd14: iSpi_TXE_Int is asserted when 14 or less data entries are present in transmit FIFO. 4'd15: iSpi_TXE_Int is asserted when 15 or less data entries are present in transmit FIFO.	R/W

2.4.8 rSpi_RXFTLR — Receive FIFO Threshold Level

This register controls the threshold value for the receive FIFO memory.
Do not write to this register when the SPI controller is enabled by bSpi_SSIENR.

Address: 5000 501Ch (SPI1)
5000 601Ch (SPI2)
5000 701Ch (SPI3)
5000 801Ch (SPI4)
5000 901Ch (SPI5)
5000 A01Ch (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_RFT			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.14 rSpi_RXFTLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b4	Reserved	Read as 0.	R
b3 to b0	bSpi_RFT	Receive FIFO Threshold Controls the level of entries (or above) at which the receive FIFO controller triggers an interrupt. If you attempt to set this value greater than the depth of the FIFO, this field is not written and retains its current value. When the number of receive FIFO entries is greater than or equal to this value + 1, the receive FIFO full interrupt is triggered. 4'd0: iSpi_RXF_Int is asserted when 1 or more data entries are present in receive FIFO. 4'd1: iSpi_RXF_Int is asserted when 2 or more data entries are present in receive FIFO. ... ... 4'd14: iSpi_RXF_Int is asserted when 15 or more data entries are present in receive FIFO. 4'd15: iSpi_RXF_Int is asserted when 16 or more data entries are present in receive FIFO.	R/W

2.4.9 rSpi_TXFLR — Transmit FIFO Level Register

This register contains the number of valid data entries in the transmit FIFO memory.

Address: 5000 5020h (SPI1)
5000 6020h (SPI2)
5000 7020h (SPI3)
5000 8020h (SPI4)
5000 9020h (SPI5)
5000 A020h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bSpi_TXTFL				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.15 rSpi_TXFLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Read as 0.	R
b4 to b0	bSpi_TXTFL	Transmit FIFO Level The number of valid data entries in the transmit FIFO.	R

2.4.10 rSpi_RXFLR — Receive FIFO Level Register

This register contains the number of valid data entries in the receive FIFO memory. This register can be read at any time.

Address: 5000 5024h (SPI1)
5000 6024h (SPI2)
5000 7024h (SPI3)
5000 8024h (SPI4)
5000 9024h (SPI5)
5000 A024h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bSpi_RXTFL				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.16 rSpi_RXFLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Read as 0.	R
b4 to b0	bSpi_RXTFL	Receive FIFO Level The number of valid data entries in the receive FIFO.	R

2.4.11 rSpi_SR — Status Register

The status register may be read at any time.

Address: 5000 5028h (SPI1)
5000 6028h (SPI2)
5000 7028h (SPI3)
5000 8028h (SPI4)
5000 9028h (SPI5)
5000 A028h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	bSpi_TXE	bSpi_RFF	bSpi_RFNE	bSpi_TFE	bSpi_TFNF	bSpi_BUSY
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Table 2.17 rSpi_SR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b6	Reserved	Read as 0.	R
b5	bSpi_TXE	Transmission Error. (SPI slave only) Set if the transmit FIFO is empty when a transfer is started. Data from the previous transmission is resent on the SPI_MISO line. This bit is cleared when read. 0: No error 1: Transmission error	R
b4	bSpi_RFF	Receive FIFO Full When the receive FIFO is completely full, this bit is set. When the receive FIFO contains one or more empty location, this bit is cleared. 0: Receive FIFO is not full 1: Receive FIFO is full	R
b3	bSpi_RFNE	Receive FIFO Not Empty Set when the receive FIFO contains one or more entries and is cleared when the receive FIFO is empty. This bit can be polled by Software to completely empty the receive FIFO. 0: Receive FIFO is empty 1: Receive FIFO is not empty	R
b2	bSpi_TFE	Transmit FIFO Empty When the transmit FIFO is completely empty, this bit is set. When the transmit FIFO contains one or more valid entries, this bit is cleared. This bit does not request an interrupt. 0: Transmit FIFO is not empty 1: Transmit FIFO is empty	R
b1	bSpi_TFNF	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	R

Table 2.17 rSpi_SR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b0	bSpi_BUSY	SPI Busy Flag When set, indicates that a serial transfer is in progress, when cleared indicates that the SPI controller is idle or disabled. 0: SPI controller is idle or disabled 1: SPI controller is actively transferring data	R

2.4.12 rSpi_IMR — Interrupt Mask Register

Address: 5000 502Ch (SPI1)
5000 602Ch (SPI2)
5000 702Ch (SPI3)
5000 802Ch (SPI4)
5000 902Ch (SPI5)
5000 A02Ch (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bSpi_RXFIM	bSpi_RXOIM	bSpi_RXUIM	bSpi_TXOIM	bSpi_TXEIM
Value after reset	0	0	0	0	0	0	0	0	0	0	1/0	1	1	1	1	1

Table 2.18 rSpi_IMR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b6	Reserved	Read as 0.	R
b5	Reserved	Reset value is 1 for SPI master, 0 for SPI slave. Keep initial value.	R/W
b4	bSpi_RXFIM	Receive FIFO Full Interrupt Mask 1’b0: iSpi_RXF_Int interrupt is masked 1’b1: iSpi_RXF_Int interrupt is not masked	R/W
b3	bSpi_RXOIM	Receive FIFO Overflow Interrupt Mask 1’b0: iSpi_RXO_Int interrupt is masked 1’b1: iSpi_RXO_Int interrupt is not masked	R/W
b2	bSpi_RXUIM	Receive FIFO Underflow Interrupt Mask 1’b0: iSpi_RXU_Int interrupt is masked 1’b1: iSpi_RXU_Int interrupt is not masked	R/W
b1	bSpi_TXOIM	Transmit FIFO Overflow Interrupt Mask 1’b0: iSpi_TXO_Int interrupt is masked 1’b1: iSpi_TXO_Int interrupt is not masked	R/W
b0	bSpi_TXEIM	Transmit FIFO Empty Interrupt Mask 1’b0: iSpi_TXE_Int interrupt is masked 1’b1: iSpi_TXE_Int interrupt is not masked	R/W

2.4.13 rSpi_ISR — Interrupt Status Register

This register reports the status of the SPI interrupts after they have been masked.

Address: 5000 5030h (SPI1)
5000 6030h (SPI2)
5000 7030h (SPI3)
5000 8030h (SPI4)
5000 9030h (SPI5)
5000 A030h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bSpi_RXFIS	bSpi_RXOIS	bSpi_RXUIS	bSpi_TXOIS	bSpi_TXEIS
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.19 rSpi_ISR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Read as 0.	R
b4	bSpi_RXFIS	Receive FIFO Full Interrupt Status 1'b0: iSpi_RXF_Int interrupt not active after masking 1'b1: iSpi_RXF_Int interrupt active after masking	R
b3	bSpi_RXOIS	Receive FIFO Overflow Interrupt Status 1'b0: iSpi_RXO_Int interrupt not active after masking 1'b1: iSpi_RXO_Int interrupt active after masking	R
b2	bSpi_RXUIS	Receive FIFO Underflow Interrupt Status 1'b0: iSpi_RXU_Int interrupt not active after masking 1'b1: iSpi_RXU_Int interrupt active after masking	R
b1	bSpi_TXOIS	Transmit FIFO Overflow Interrupt Status 1'b0: iSpi_TXO_Int interrupt not active after masking 1'b1: iSpi_TXO_Int interrupt active after masking	R
b0	bSpi_TXEIS	Transmit FIFO Empty Interrupt Status 1'b0: iSpi_TXE_Int interrupt not active after masking 1'b1: iSpi_TXE_Int interrupt active after masking	R

2.4.14 rSpi_RISR — Raw Interrupt Status Register

This register reports the status of the SPI interrupts prior to masking.

Address: 5000 5034h (SPI1)
5000 6034h (SPI2)
5000 7034h (SPI3)
5000 8034h (SPI4)
5000 9034h (SPI5)
5000 A034h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bSpi_RXFIR	bSpi_RXOIR	bSpi_RXUIR	bSpi_TXOIR	bSpi_TXEIR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.20 rSpi_RISR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Read as 0.	R
b4	bSpi_RXFIR	Receive FIFO Full Interrupt Status 1'b0: iSpi_RXF_Int interrupt not active prior to masking 1'b1: iSpi_RXF_Int interrupt active prior to masking	R
b3	bSpi_RXOIR	Receive FIFO Overflow Interrupt Status 1'b0: iSpi_RXO_Int interrupt not active prior to masking 1'b1: iSpi_RXO_Int interrupt active prior to masking	R
b2	bSpi_RXUIR	Receive FIFO Underflow Interrupt Status 1'b0: iSpi_RXU_Int interrupt not active prior to masking 1'b1: iSpi_RXU_Int interrupt active prior to masking	R
b1	bSpi_TXOIR	Transmit FIFO Overflow Interrupt Status 1'b0: iSpi_TXO_Int interrupt not active prior to masking 1'b1: iSpi_TXO_Int interrupt active prior to masking	R
b0	bSpi_TXEIR	Transmit FIFO Empty Interrupt Status 1'b0: iSpi_TXE_Int interrupt not active prior to masking 1'b1: iSpi_TXE_Int interrupt active prior to masking	R

2.4.15 rSpi_TXOICR — Transmit FIFO Overflow Interrupt Clear Register

Address: 5000 5038h (SPI1)
5000 6038h (SPI2)
5000 7038h (SPI3)
5000 8038h (SPI4)
5000 9038h (SPI5)
5000 A038h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_TXOICR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.21 rSpi_TXOICR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Read as 0.	R
b0	bSpi_TXOICR	Clear Transmit FIFO Overflow Interrupt This register reflects the status of the interrupt. A read from this register clears the iSpi_TXO_Int interrupt, writing has no effect.	R

2.4.16 rSpi_RXOICR — Receive FIFO Overflow Interrupt Clear Register

Address: 5000 503Ch (SPI1)
5000 603Ch (SPI2)
5000 703Ch (SPI3)
5000 803Ch (SPI4)
5000 903Ch (SPI5)
5000 A03Ch (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_RXOICR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.22 rSpi_RXOICR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Read as 0.	R
b0	bSpi_RXOICR	Clear Receive FIFO Overflow Interrupt This register reflects the status of the interrupt. A read from this register clears the iSpi_RXO_Int interrupt, writing has no effect.	R

2.4.17 rSpi_RXUICR — Receive FIFO Underflow Interrupt Clear Register

Address: 5000 5040h (SPI1)
5000 6040h (SPI2)
5000 7040h (SPI3)
5000 8040h (SPI4)
5000 9040h (SPI5)
5000 A040h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_RXUICR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.23 rSpi_RXUICR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Read as 0.	R
b0	bSpi_RXUICR	Clear Receive FIFO Underflow Interrupt This register reflects the status of the interrupt. A read from this register clears the iSpi_RXU_Int interrupt, writing has no effect.	R

2.4.18 rSpi_ICR — Interrupt Clear Register

Address: 5000 5048h (SPI1)
5000 6048h (SPI2)
5000 7048h (SPI3)
5000 8048h (SPI4)
5000 9048h (SPI5)
5000 A048h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_ICR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.24 rSpi_ICR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Read as 0.	R
b0	bSpi_ICR	Clear Interrupts This register is set if any of the interrupts below are active. A read clears the interrupts iSpi_TXO_Int, iSpi_RXU_Int and iSpi_RXO_Int interrupts. Writing to this register has no effect.	R

2.4.19 rSpi_DMACR — DMA Control Register

Address: 5000 504Ch (SPI1)
5000 604Ch (SPI2)
5000 704Ch (SPI3)
5000 804Ch (SPI4)
5000 904Ch (SPI5)
5000 A04Ch (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_TDMAE	bSpi_RDMAE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.25 rSpi_DMACR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved	Read as 0.	R
b1	bSpi_TDMAE	Transmit DMA Enable This bit enables/disables the transmit FIFO DMA channel. 0: Transmit DMA disabled 1: Transmit DMA enabled	R/W
b0	bSpi_RDMAE	Receive DMA Enable This bit enables/disables the receive FIFO DMA channel 0: Receive DMA disabled 1: Receive DMA enabled	R/W

2.4.20 rSpi_DMATDLR — DMA Transmit Data Level

Address: 5000 5050h (SPI1)
5000 6050h (SPI2)
5000 7050h (SPI3)
5000 8050h (SPI4)
5000 9050h (SPI5)
5000 A050h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_DMATDLR			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.26 rSpi_DMATDLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b4	Reserved	Read as 0.	R
b3 to b0	bSpi_DMATDLR	Transmit Data Level This 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 request is generated when the number of valid data entries in the transmit FIFO is equal to or below this field value, and bSpi_TDMAE= 1. 4'd0: DMA request is asserted when 0 data entries are present in the transmit FIFO 4'd1: DMA request is asserted when 1 or less data entries are present in the transmit FIFO ... ... 4'd14: DMA request is asserted when 14 or less data entries are present in the transmit FIFO 4'd15: DMA request is asserted when 15 or less data entries are present in the transmit FIFO	R/W

2.4.21 rSpi_DMARDLR — DMA Receive Data Level

Address: 5000 5054h (SPI1)
5000 6054h (SPI2)
5000 7054h (SPI3)
5000 8054h (SPI4)
5000 9054h (SPI5)
5000 A054h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	bSpi_DMARDLR			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.27 rSpi_DMARDLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b4	Reserved	Read as 0.	R
b3 to b0	bSpi_DMARDLR	Receive Data Level This field controls the level at which a DMA request is made by the receive logic. The watermark level = bSpi_DMARDLR+1 that is, DMA request is generated when the number of valid data entries in the receive FIFO is equal to or above this field value + 1, and bSpi_RDMAE = 1. 4'd0: DMA request is asserted when 1 or more data entries are present in the receive FIFO 4'd1: DMA request is asserted when 2 or more data entries are present in the receive FIFO ... ... 4'd14: DMA request is asserted when 15 or more data entries are present in the receive FIFO 4'd15: DMA request is asserted when 16 or more data entries are present in the receive FIFO	R/W

2.4.22 rSpi_DR — Data Register

The SPI data register is a 16-bit read/write buffer for the transmit / receive FIFOs. When the register is read, data in the receive FIFO buffer is accessed. When it is written to, data are moved into the transmit FIFO buffer, a write can occur only when bSpi_SSIENR = 1. FIFOs are reset when bSpi_SSIENR = 0.

NOTE

The rSpi_DR register in the SPI controller occupies thirty-six 32-bit locations of the memory map to facilitate AHB burst transfers. Writing to any of these address locations has the same effect as pushing the data from the APB bus into the transmit FIFO. Reading from any of these locations has the same effect as popping data from the receive FIFO onto the APB bus. The FIFO buffers on the SPI controller are not addressable.
If the Data Register (DR) is accessed from an AHB master (such as a DMA controller or a processor), the AHB transfer type may be a burst. During AHB burst transfers, the address increments after each beat of the burst.

Address: 5000 5060h (SPI1)
5000 6060h (SPI2)
5000 7060h (SPI3)
5000 8060h (SPI4)
5000 9060h (SPI5)
5000 A060h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bSpi_DR															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.28 rSpi_DR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Read as 0.	R
b15 to b0	bSpi_DR	Data Register When writing to this register, you must right-justify the data. Read data are automatically right-justified. Read: Receive FIFO buffer Write: Transmit FIFO buffer ***Caution***) In DMA mode, bSpi_TDMAE and bSpi_TDMAE1 or/and bSpi_RDMAE and bSpi_RDMAE1 set to 1, the DMA controller must be programmed with following parameters: • Only one address is used for rSpi_DR register. (12'h060 for example) The DMA is configured in 16 bits word mode only.	R/W

2.4.23 rSpi_RX_SAMPLE_DLY — RXD Sample Delay Register

CAUTION

This register exists in SPI Master only.

Address: 5000 50F0h (SPI1)
5000 60F0h (SPI2)
5000 70F0h (SPI3)
5000 80F0h (SPI4)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bSpi_RX_Sample_Delay							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.29 rSpi_RX_SAMPLE_DLY Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bSpi_RX_Sample_Delay	Receive Data (SPI_MISO) Sample Delay This register is used to delay the sample of the SPI_MISO input signal. Each value represents a single SPI_SCLK delay on the sample of the SPI_MISO signal. See **Section 2.5.5, Data Input Sample Delay**. ***Note*)** If this register is programmed with a value that exceeds the depth of the internal shift registers (64), a zero (0) delay will be applied to the SPI_MISO sample.	R/W

2.4.24 rSpi_TDMACR — DMA Control Register in Transmit Mode

Address: 5000 5100h (SPI1)
5000 6100h (SPI2)
5000 7100h (SPI3)
5000 8100h (SPI4)
5000 9100h (SPI5)
5000 A100h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	bSpi_CURRENT_DEST_BLOCK_SIZE												
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bSpi_DEST_BLOCK_SIZE													bSpi_DEST_BURST_SIZE		bSpi_TDMAE1
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.30 rSpi_TDMACR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31	Reserved	Read as 0.	R
b30, b29	Reserved	Keep initial value	R/W
b28 to b16	bSpi_CURRENT_DEST_BLOCK_SIZE	Current remaining of DEST_BLOCK_SIZE This field is decremented each time the block transfer ends. bSpi_CURRENT_DEST_BLOCK_SIZE is reloaded with bSpi_DEST_BLOCK_SIZE value, when the firmware: Set “1” on bSpi_TDMAE1 (rising edge)	R
b15 to b3	bSpi_DEST_BLOCK_SIZE	DEST_BLOCK_SIZE Destination Block Transfer Size in Transmit FIFO. SPI controller is the flow controller. Therefore, the user must write this field before or at the same time the DMA mode is enabled. The number programmed into DEST_BLOCK_SIZE indicates the total number of single transactions to perform for each block transfer. The size of single transaction is one 16 bits word. Once the transfer starts, the read of bSpi_DEST_BLOCK_SIZE gives the total number of data bytes to be written in the Transmit FIFO in order to end the block transfer. 13’d0: 0 word to transfer or end of block transfer 13’d1: 1 word to transfer 13’d2: 2 words to transfer ... ... 13’d8191: 8191 words to transfer	R/W
b2, b1	bSpi_DEST_BURST_SIZE	DEST_BURST_SIZE Destination Burst Transaction Size in Transmit FIFO. The SPI controller is the flow controller. Therefore, the user must write this field before or at the same time the DMA mode is enabled. 2’b00: 1 word 2’b01: 4 words 2’b10: 8 words 2’b11: Reserved, not used	R/W

Table 2.30 rSpi_TDMACR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b0	bSpi_TDMAE1	Transmit DMA Enables/Disables 0: Disable the DMA in Transmit mode 1: Enable the DMA in Transmit mode The bSpi_TDMAE1 is automatically cleared by hardware to disable the DMA in Transmit mode after the last transfer in Transmit FIFO has completed (DEST_BLOCK_SIZE words written in Transmit FIFO) Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. ***Caution*)** • Prior to enable this bit, software must enable the bSpi_TDMAE of the rSpi_DMACR register to enable the DMA mode. After that, the DMA mode is controlled only with the bSpi_TDMAE1. • If this bit is clear during a DMA transfer, the current transfer (Burst or Single) is finished before the stop of DMA mode. To complete the DMA Block transfer, write the bSpi_DEST_BLOCK_SIZE with the bSpi_CURRENT_DEST_BLOCK_SIZE, and bSpi_DEST_BURST_SIZE with the appropriate value. The bSpi_CURRENT_DEST_BLOCK_SIZE value will be consistent only when the current transfer is finished.	R/W

2.4.25 rSpi_RDMACR — DMA Control Register in Receive Mode

Address: 5000 5104h (SPI1)
5000 6104h (SPI2)
5000 7104h (SPI3)
5000 8104h (SPI4)
5000 9104h (SPI5)
5000 A104h (SPI6)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	bSpi_CURRENT_SRC_BLOCK_SIZE												
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bSpi_SRC_BLOCK_SIZE													bSpi_SRC_BURST_SIZE		bSpi_RDMAE1
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.31 rSpi_RDMACR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31	Reserved	Read as 0.	R
b30, b29	Reserved	Keep initial value	R/W
b28 to b16	bSpi_CURRENT_SRC_BLOCK_SIZE	Current remaining of SRC_BLOCK_SIZE This field is decremented each time the block transfer ends. bSpi_CURRENT_SRC_BLOCK_SIZE is reloaded with bSpi_SRC_BLOCK_SIZE value, when the firmware: Set “1” on bSpi_RDMAE1 (rising edge)	R
b15 to b3	bSpi_SRC_BLOCK_SIZE	SRC_BLOCK_SIZE Source Block Transfer Size in Receive FIFO. SPI controller is the flow controller. The user must write this field before or at the same time the DMA mode is enabled. The number programmed into SRC_BLOCK_SIZE indicates the total number of single transactions to perform for each block transfer. The size of single transaction is one 16 bits word. 13’d0: 0 word to transfer or end of block transfer 13’d1: 1 word to transfer 13’d2: 2 words to transfer ... ... 13’d8191: 8191 words to transfer	R/W
b2, b1	bSpi_SRC_BURST_SIZE	SRC_BURST_SIZE Source Burst Transaction Size in Receive FIFO. The SPI controller is the flow controller. The user must write this field before or at the same time the DMA mode is enabled. 2’b00: 1 word 2’b01: 4 words 2’b10: 8 words 2’b11: Reserved, not used	R/W

Table 2.31 rSpi_RDMACR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b0	bSpi_RDMAE1	Receive DMA Enables/Disables 0: Disable the DMA in Receive mode 1: Enable the DMA in Receive mode The bSpi_RDMAE1 is automatically cleared by hardware to disable the DMA in Receive mode after the last transfer in Receive FIFO has completed (SRC_BLOCK_SIZE words read in Receive FIFO) Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. ***Caution*)** • Prior to enable this bit, Software must enable the bSpi_RDMAE of the rSpi_DMACR register to enable the DMA mode. After that, the DMA mode is controlled only with the bSpi_RDMAE1. • If this bit is clear during a DMA transfer, the current transfer (Burst or Single) is finished before the stop of DMA mode. To complete the DMA Block transfer, write the bSpi_SRC_BLOCK_SIZE with the bSpi_CURRENT_SRC_BLOCK_SIZE, and bSpi_SRC_BURST_SIZE with the appropriate value. The bSpi_CURRENT_SRC_BLOCK_SIZE value will be consistent only when the current transfer is finished.	R/W

2.5 Operation

2.5.1 General description

A serial master or serial slave peripheral device connecting the SPI controller must have at least one of the following interfaces:

- Motorola Serial Peripheral Interface
  - A four-wire, full-duplex serial protocol from Motorola. There are four possible combinations for the serial clock phase and polarity. The clock phase (bSpi_SCPH) determines whether the serial transfer begins with the falling edge of the slave select signal or the first edge of the serial clock. The slave select line is held high when the SPI controller is idle or disabled. For more information, refer to **Section 2.5.9, Motorola Serial Peripheral Interface**.
- Texas Instruments Serial Protocol (SSP)
  - A four-wire, full-duplex serial protocol. The slave select line used for SPI and Microwire protocols doubles as the frame indicator for the SSP protocol. For more information, refer to **Section 2.5.10, Texas Instruments Synchronous Serial Protocol**.
- National Semiconductor Microwire
  - A half-duplex serial protocol, which uses a control word transmitted from the serial master to the target serial slave. For more information, refer to **Section 2.5.11, National Semiconductor Microwire**.

You can program the bSpi_FRF (frame format) in the Control Register 0 (rSpi_CTRLR0) to select which protocol is used. The serial protocols supported by SPI controller allow for serial slaves to be selected or addressed using either hardware or Software.

When implemented in hardware, serial slaves are selected under the control of dedicated hardware select lines. The number of select lines generated from the serial master is equal to the number of serial slaves present on the bus. The serial master device asserts the select line of the target serial slave before data transfer begins. This architecture is illustrated in **Section 2.5.2, Typical Connection between SPI Master & Slave**.

When implemented in Software, the input select line of each serial slave should originate either from a single slave select output signal on the serial master (you must configure the master to have one slave select output). The main program in the Software domain controls selection of the target slave device, this architecture is illustrated in **Section 2.5.2, Typical Connection between SPI Master & Slave**. Software would use rSpi_SSIENR register in all slaves in order to control which slave is to respond to the serial transfer request from the master device.

2.5.2 Typical Connection between SPI Master & Slave

The SPI controller enables serial communication between serial master with serial slave peripheral devices. SPI master initiates and controls all serial transfers.

The figure below shows an example of connection between SPI master and all other SPI slave devices. The serial clock, generated and controlled by the SPI master is driven out on the SPI_CLK line. When the SPI controller is disabled (bSpi_SSIENR = 0), no serial transfers can occur and SPI_CLK is held in “inactive” state, as defined by the serial protocol under which it operates.

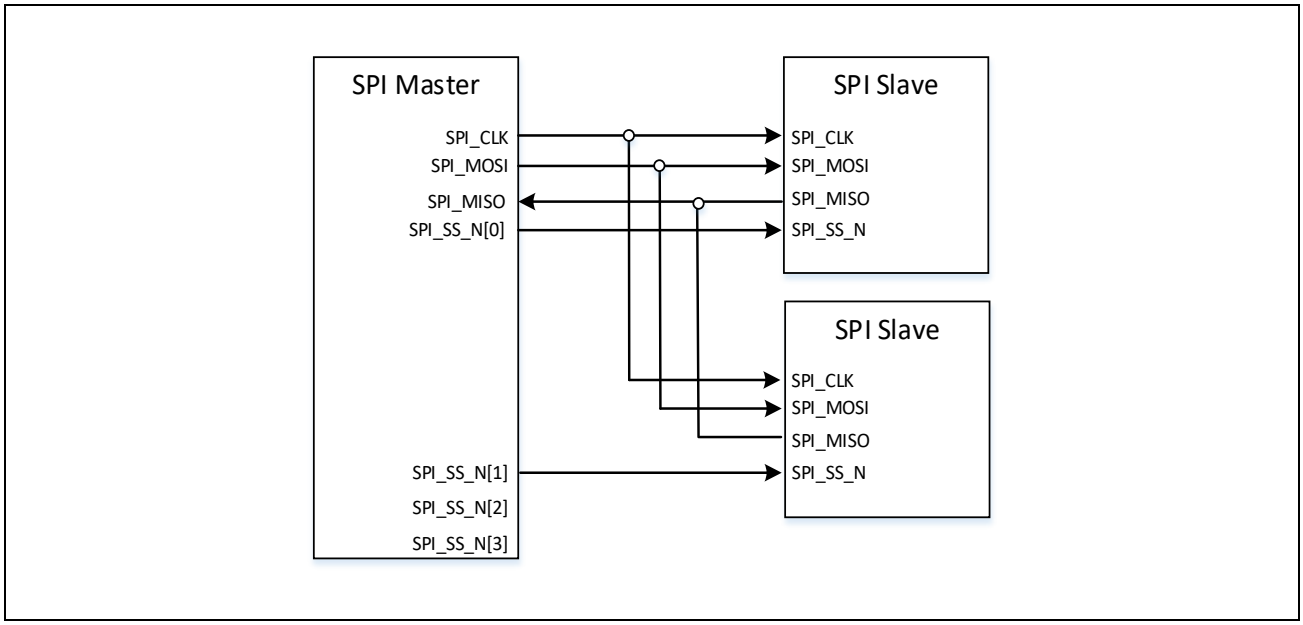

Figure 2.3 Typical Connection between SPI Master & Slave

2.5.3 Control Slave Select Line by Hardware or Software Mode

The SPI controller enables serial communication between serial master with serial slave peripheral devices can be controlled in two basics modes:

- Select line controlled in hardware mode from serializer module
- Select line controlled in Software mode

See following bits in rSpi_SER registers:

- bSpi_CtrlSS[3:0]
- bSpi_HardwareSS[3:0]
- bSpi_SoftwareSS[3:0]

CAUTION

- If software slave select control bits (bSpi_SoftwareSS[3:0]) are controlled in software mode, these bits should be toggled properly when SPI_SS_N (M) must toggle between two SPI data frames (the length of each data frame ranges from 4 to 16 bits).
- SPI_SS_N (S) needs toggle between two SPI data frames, when frame format is Motorola SPI and bSpi_SCPH in rSpi_CTRLR0 register is 0.

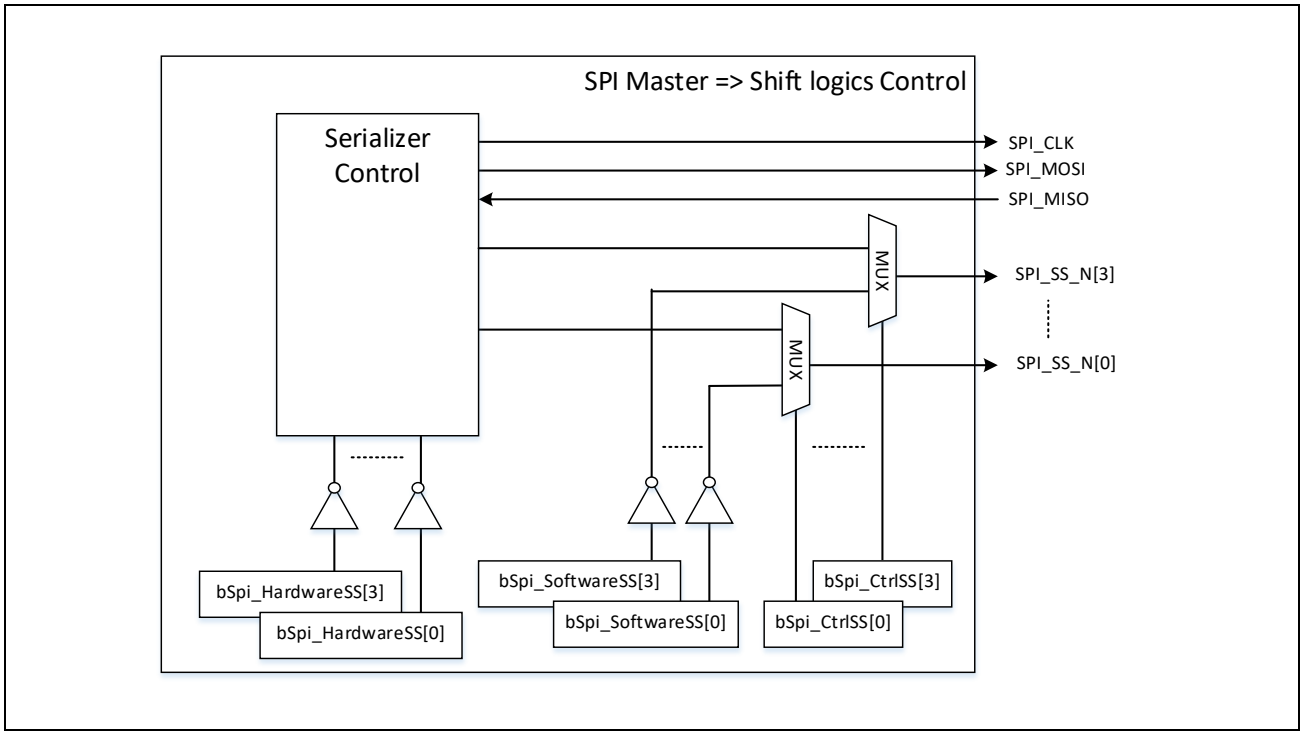

Figure 2.4 Control Slave Select Line by Hardware or Software Mode

2.5.4 Programmable Prescaler Clock

The frequency of the SPI_SCLK (serial reference clock, 125 MHz max) must be less than or equal to the frequency of SPI_PCLK (APB), which guarantees that control signals from the serial reference clock SPI_SCLK domain are synchronized to the SPI_PCLK domain. When SPI_PCLK and SPI_SCLK are asynchronous, synchronization logic transfers control signals from one clock domain to the other.

The recommended frequency ratio restriction between the serial clock (SPI_CLK) and the serial reference clock (SPI_SCLK) are described as:

- Master:
  SPI_SCLK ≥ 2 × (maximum SPI_CLK out)
- Slave (receive only):
  SPI_SCLK ≥ 8 × (maximum SPI_CLK in)
- Slave:
  SPI_SCLK ≥ 10 × (maximum SPI_CLK in)

CAUTION

Actual maximum Frequency (SPI_CLK) should be set within AC spec of the SPI.

2.5.5 Data Input Sample Delay

The SPI master includes additional logic in order to delay the default sample time of the SPI_MISO (M) signal. This additional logic can help to increase the maximum achievable frequency on the serial bus.

Round trip routing delays on the SPI_CLK (M) signal from the master and the SPI_MISO signal from the slave can mean that the timing of the SPI_MISO signal -as seen by the master- has moved away from the normal sampling time.

The figure below illustrates this situation.

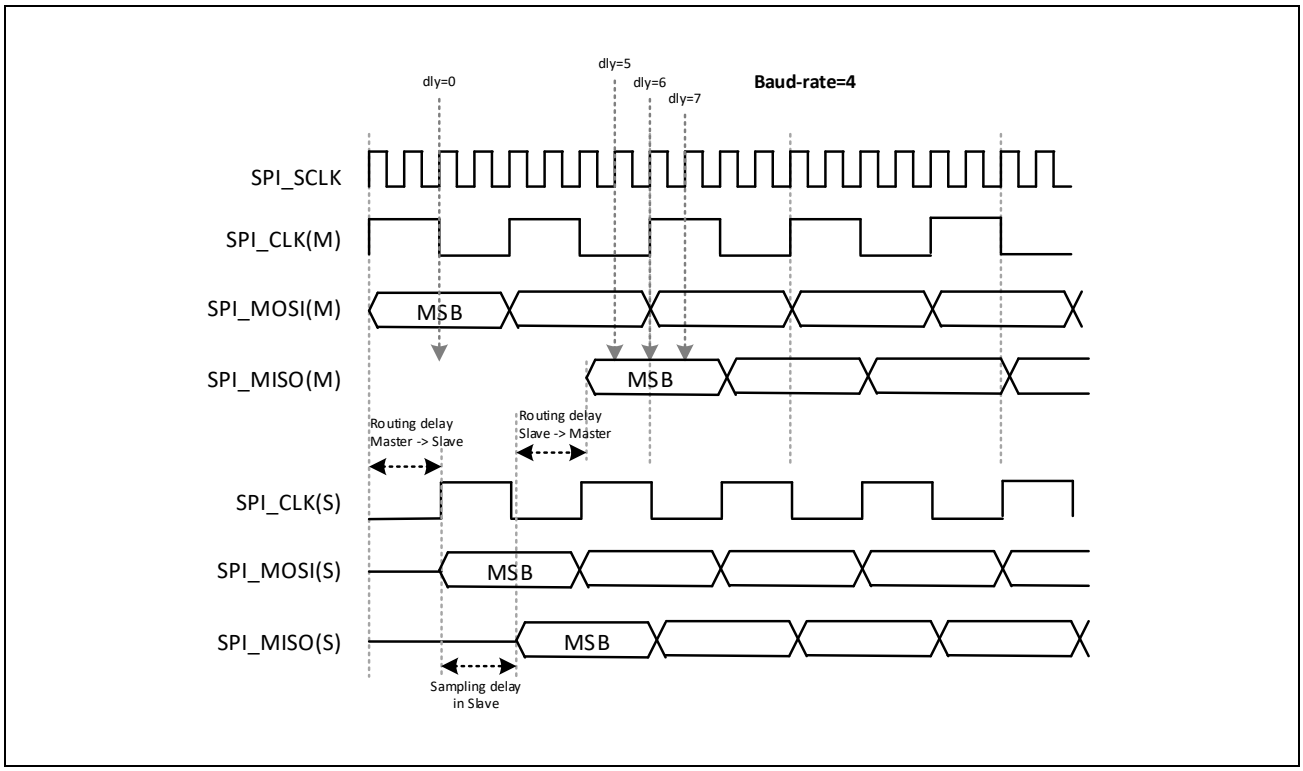

Figure 2.5 SPI Data Input Sample Delay

CAUTION

The reference for dly = 0 depends on the type of format. The figure above was given for a capture on the second edge of serial clock (e.g. Motorola Format with bSpi_SCPH = 1)

The slave uses the SPI_CLK (S) signal from the master as a strobe in order to drive SPI_MISO (S) signal data onto the serial bus. Routing and sampling delays on the SPI_CLK (M) signal by the slave device can mean that the SPI_MISO has not stabilized to the correct value before the master samples this signal. The figure SPI Data Input Sample Delay shows an example of how a routing delay on the SPI_MISO signal can result in an incorrect value at the default time when the master samples the port.

User can dynamically program a delay value in order to move the sampling time of the SPI_MISO signal from the default.

The sample delay logic has a resolution of one SPI_SCLK cycle. Software can “train” the serial bus by coding a loop that continually reads from the slave and increments the master’s “Data Input Sample Delay” value until the correct data is received by the master.

2.5.6 Transmit & Receive FIFO & Control

The depth of both transmit and receive FIFO buffers is 16 words. The width of both transmit and receive FIFO buffers is fixed at 16 bits. Data frames that are less than 16 bits in size must be right justified when written into the transmit FIFO buffer. The shift control logic automatically right justifies receive data in the receive FIFO buffer.

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

The transmit and receive FIFO buffers are cleared when the SPI controller is disabled (bSpi_SSIENR = 0) or when it is reset the SPI module.

The transmit FIFO is loaded by CPU write commands to the SPI data register (rSpi_DR). Data are popped (removed) from the transmit FIFO by the shift control logic into the transmit shift register. The transmit FIFO generates a FIFO empty interrupt request (iSpi_TXE_Int) when the number of entries in the FIFO is less than or equal to the FIFO threshold value.

The threshold value, set through the rSpi_TXFTLR, determines the level of FIFO entries at which an interrupt is generated. The threshold value allows you to provide early indication to the processor that the transmit FIFO is nearly empty. A transmit FIFO overflow interrupt (iSpi_TXO_Int) is generated if you attempt to write data into an already full transmit FIFO.

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

The threshold value allows you to provide early indication to the processor that the receive FIFO is nearly full. A receive FIFO overflow interrupt (iSpi_RXO_Int) is generated when the receive shift logic attempts to load data into a completely full receive FIFO. However, this newly received data are lost. A receive FIFO underflow interrupt (iSpi_RXU_Int) is generated if you attempt to read from an empty receive FIFO. This alerts the processor that the read data are invalid.

2.5.7 Interruption Management

The SPI controller supports combined and individual interrupt requests, each of which can be masked. The combined interrupt request is the OR'ed result of all other SPI interrupts after masking.

The SPI interrupts are described as follows:

- Transmit FIFO Empty Interrupt (iSpi_TXE_Int)
  - Set when the transmit FIFO is equal to or below its threshold value and requires service to prevent an under run.
  - The threshold value, set through a Software programmable register, determines the level of transmit FIFO entries at which an interrupt is generated.
  - This interrupt is cleared by hardware when data are written into the transmit FIFO buffer, bringing it over the threshold level.
- Transmit FIFO Overflow Interrupt (iSpi_TXO_Int)
  - Set when a CPU access attempts to write into the transmit FIFO after it has been completely filled. When set, data written from the CPU is discarded.
  - This interrupt remains set until you read the transmit FIFO overflow interrupt clear register (rSpi_TXOICR).
- Receive FIFO Full Interrupt (iSpi_RXF_Int)

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

- Receive FIFO Overflow Interrupt (iSpi_RXO_Int)
  - Set when the receive logic attempts to place data into the receive FIFO after it has been completely filled. When set, newly received data are discarded.
  - This interrupt remains set until you read the receive FIFO overflow interrupt clear register (rSpi_RXOICR).
- Receive FIFO Underflow Interrupt (iSpi_RXU_Int)
  - Set when a CPU access attempts to read from the receive FIFO when it is empty.
  - When set, zeros are read back from the receive FIFO.
  - This interrupt remains set until you read the receive FIFO underflow interrupt clear register (rSpi_RXUICR).
- Combined Interrupt Request (SPI_Int)
  - OR’ed result of all the above interrupt requests after masking.
  - To mask this interrupt signal, it must mask all other SPI interrupt requests.

2.5.8 Transfer Mode

There are four possible transfer modes for performing SPI serial transactions:

- Transmit & Receive. See **Section 2.5.8.1, Transmit and Receive Mode**.
- Transmit only. See **Section 2.5.8.2, Transmit Only Mode**.
- Receive only. See **Section 2.5.8.3, Receive Only Mode**.
- EEPROM read. See **Section 2.5.8.4, EEPROM Read Mode**.

The transfer mode is set by writing bSpi_TMOD in the Control Register 0 (rSpi_CTRLR0).

CAUTION

- The transfer mode setting does not affect the duplex of the serial transfer.
- bSpi_TMOD is ignored for Microwire transfers, which are controlled by the rSpi_MWCR register.

2.5.8.1 Transmit and Receive Mode

When bSpi_TMOD = 2'b00, both transmit and receive logic are valid.

The data transfer occurs as normal according to the selected frame format (serial protocol). Transmit data are popped from the transmit FIFO and sent through the SPI_MOSI line to the target device, which replies with data on the SPI_MISO line. The receive data from the target device is moved from the receive shift register into the receive FIFO at the end of each data frame.

2.5.8.2 Transmit Only Mode

When bSpi_TMOD = 2'b01, the receive data are invalid and should not be stored in the receive FIFO.

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

2.5.8.3 Receive Only Mode

When bSpi_TMOD = 2'b10, the transmit data are invalid.

When the SPI controller is a slave, the transmit FIFO is never popped in Receive Only mode. The SPI_MISO output remains at a constant logic level during the transmission. The data transfer occurs as normal according to the selected frame format (serial protocol).

The receive data from the target device is moved from the receive shift register into the receive FIFO at the end of each data frame. You should mask interrupts originating from the transmit logic when this mode is entered.

2.5.8.4 EEPROM Read Mode

This transfer mode is only valid for master mode.

When bSpi_TMOD = 2'b11, the transmit data is used to transmit an opcode and/or an address to the EEPROM device. Typically, this takes three data frames (8-bit opcode followed by 8-bit upper address and 8-bit lower address). During the transmission of the opcode and address, no data is captured by the receive logic (as long as the SPI master is transmitting data on its SPI_MOSI line, data on the SPI_MISO line is ignored). The SPI master continues to transmit data until the transmit FIFO is empty.

Therefore, you should ONLY have enough data frames in the transmit FIFO to supply the opcode and address to the EEPROM. If more data frames are in the transmit FIFO than are needed, then read data is lost.

When the transmit FIFO becomes empty (all control information has been sent), data on the receive line (SPI_MISO) is valid and is stored in the receive FIFO. The SPI_MOSI output is held at a constant logic level. The serial transfer continues until the number of data frames received by the SPI master matches the value of the bSpi_NDF in the rSpi_CTRLR1 register + 1

CAUTION

- EEPROM read mode is only valid for master mode.
- EEPROM read mode is not supported when the SPI controller is configured to be in the Texas Instruments Serial Protocol.

2.5.9 Motorola Serial Peripheral Interface

The Motorola Serial Peripheral Interface is a serial link with four-wire, full-duplex.

There are four possible combinations for the serial clock phase and polarity:

- The clock phase (bSpi_SCPH) determines whether the serial transfer begins with the falling edge of the slave select signal or the first edge of the serial clock. The slave select line is held high when the SPI link is idle or disabled. With the Motorola format, the clock polarity (bSpi_SCPOL) configuration bit determines whether the inactive state of the serial clock is high or low. To transmit data, both SPI peripherals must have identical serial clock phase (bSpi_SCPH) and clock polarity (bSpi_SCPOL) values. The data frame can be 4 to 16 bits in length. When bSpi_SCPH = 0, data transmission begins on the falling edge of the slave select signal. The first data bit is captured by the master and slave peripherals on the first edge of the serial clock, therefore, valid data must be present on the SPI_MISO and SPI_MOSI lines prior to the first serial clock edge.
  The figure below shows the timing diagram for a single SPI data transfer when bSpi_SCPH = 0. The serial clock is shown for bSpi_SCPOL = 0 and bSpi_SCPOL = 1.

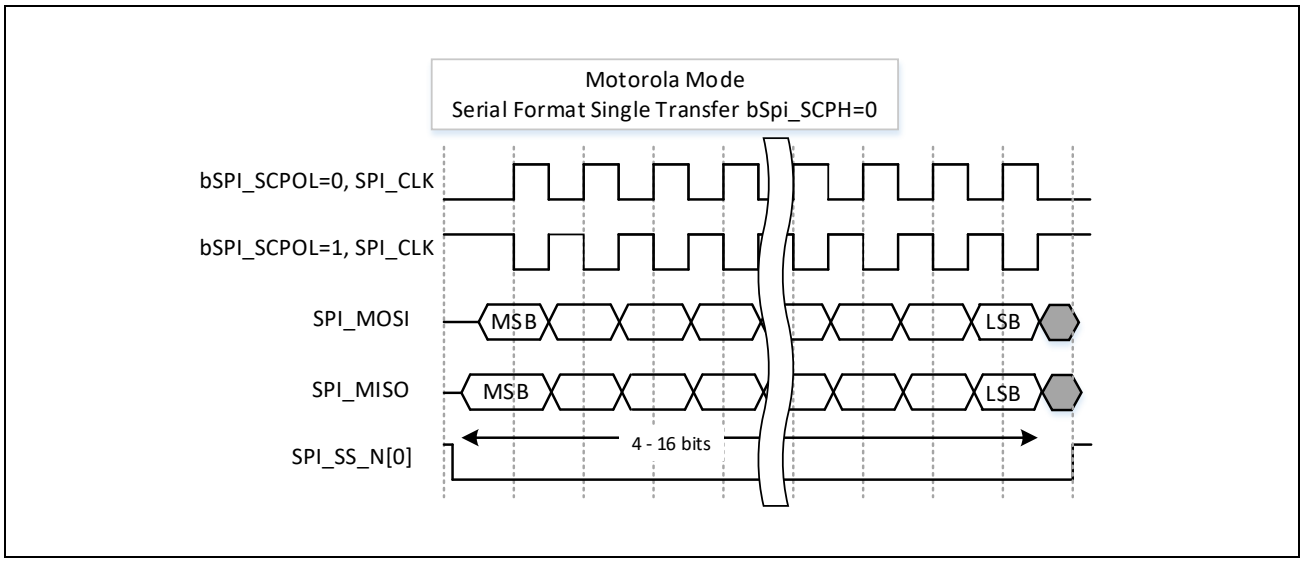

Figure 2.6 Motorola Mode, Single Transfer, bSPI_SCPH = 0

- When bSpi_SCPH = 0, this SPI slave starts data transmission on the falling edge of the slave select signal, continuous data transfers require the slave select signal to toggle before beginning the next data frame. This SPI master toggles out select lines controlled in hardware mode between data frames, and the serial clock is held to its default value while the slave select signal is not active.
  The figure below shows the timing diagram for continuous SPI data transfer when bSpi_SCPH = 0.

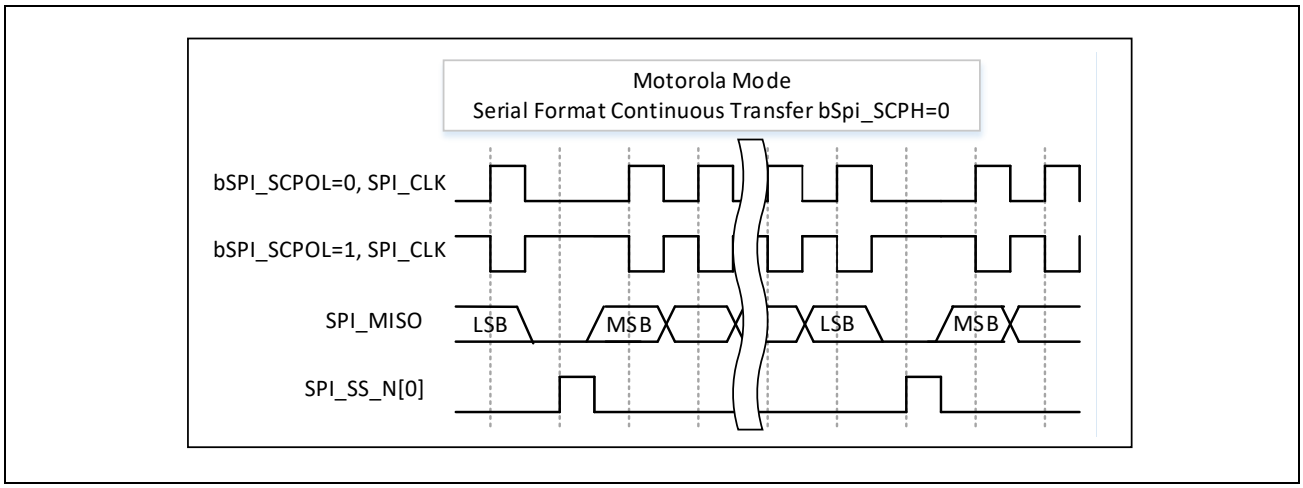

Figure 2.7 Motorola Mode, Continuous Transfer, bSpi_SCPH = 0

- When bSpi_SCPH = 1, both master and slave peripherals begin transmitting data on the first serial clock edge after the slave select line is activated. The first data bit is captured on the second (trailing) serial clock edge. Data are propagated by the master and slave peripherals on the leading edge of the serial clock. During continuous data frame transfers, the slave select line may be held active low until the last bit of the last frame has been captured.
  The figure below shows the timing diagram for a single SPI data transfer when bSpi_SCPH = 1.

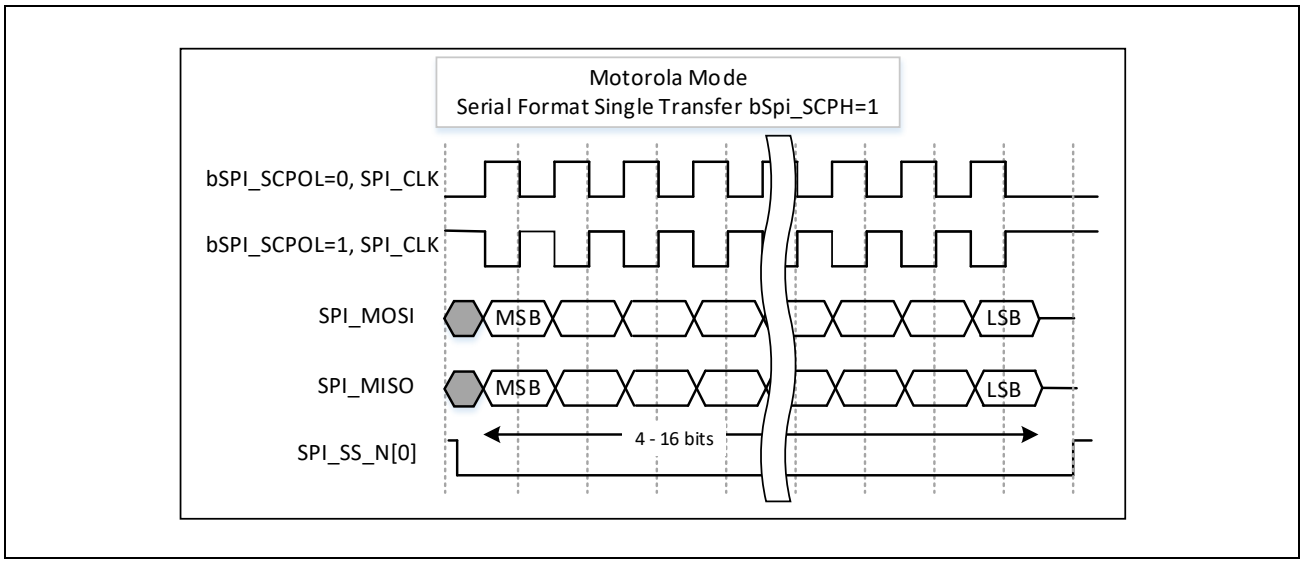

Figure 2.8 Motorola Mode, Single transfer, bSpi_SCPH = 1

- Continuous data frames are transferred in the same way as single frames, with the MSB of the next frame following directly after the LSB of the current frame. The slave select signal is held active for the duration of the transfer.
  The figure below shows the timing diagram for continuous SPI data transfer when bSpi_SCPH = 1.

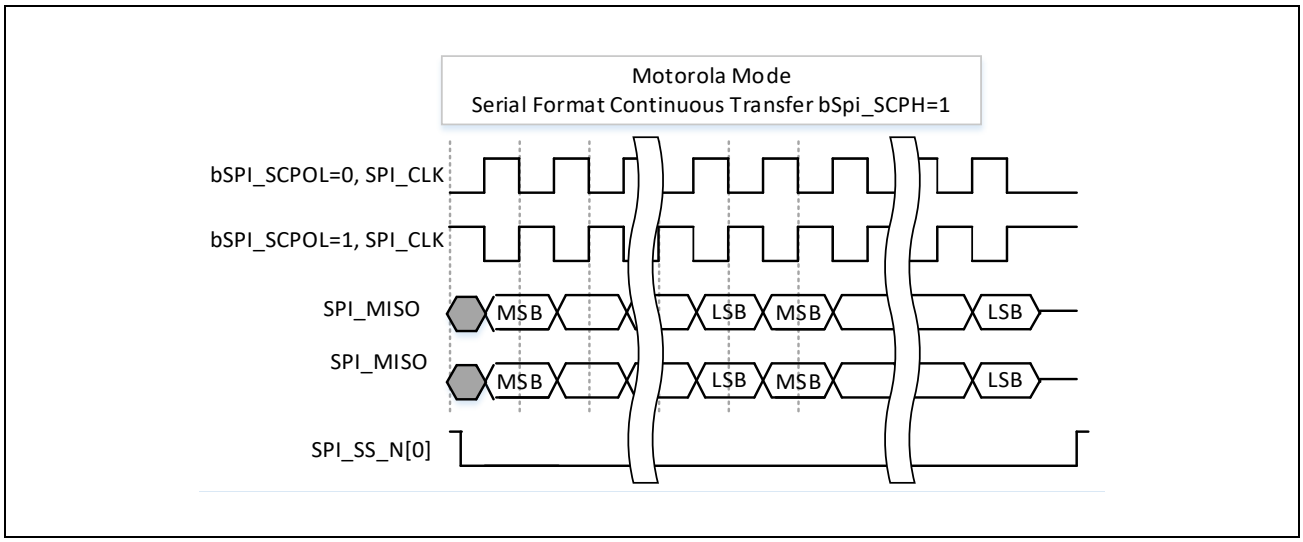

Figure 2.9 Motorola Mode, Continuous Transfer, bSpi_SCPH = 1

2.5.10 Texas Instruments Synchronous Serial Protocol

Data transfers begin by asserting the frame indicator line SPI_SS_N[0] for one serial clock period.

Data to be transmitted are driven onto the SPI_MOSI line one serial clock cycle later, similarly data from the slave are driven onto the SPI_MISO line. Data are propagated on the rising edge of the serial clock SPI_CLK and captured on the falling edge. The length of the data frame ranges from 4 to 16 bits.

The figure below shows the timing diagram for a single Texas serial transfer.

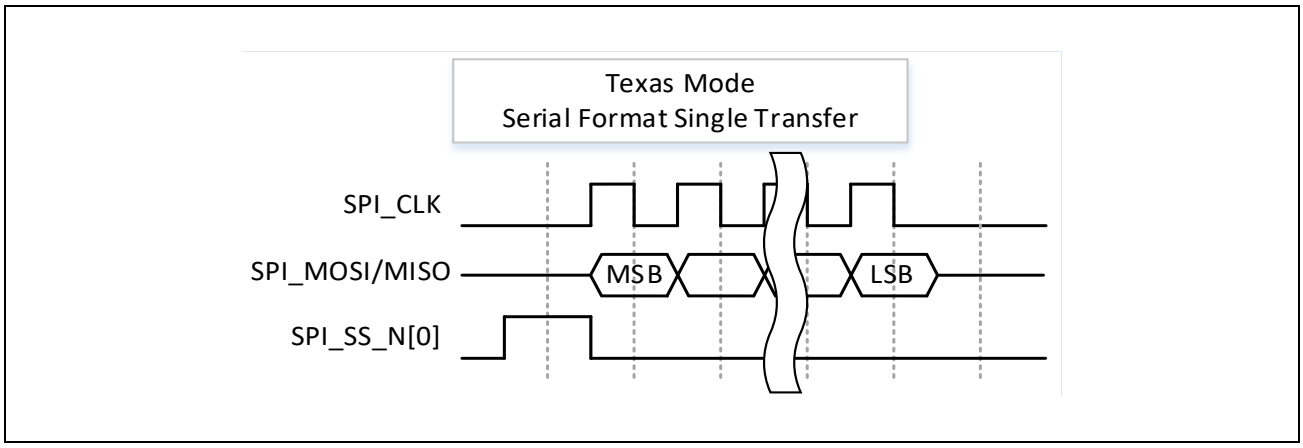

Figure 2.10 Texas Mode, Single Transfer

Continuous data frames are transferred in the same way as single data frames. The frame indicator is asserted for one clock period during the same cycle as the LSB from the current transfer, indicating that another data frame follows. The figure below shows the timing for a continuous Texas serial transfer.

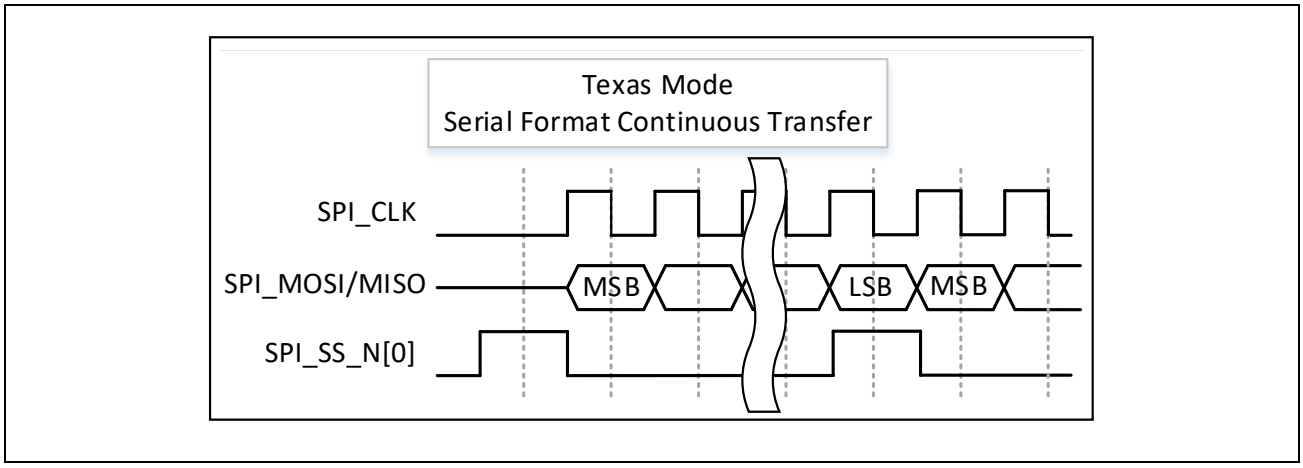

Figure 2.11 Texas Mode, Continuous Transfer

2.5.11 National Semiconductor Microwire

In this mode, when the SPI controller is a master, data transmission begins with the falling edge of the slave select signal SPI_SS_N[0]. One-half serial clock SPI_CLK period later, the first bit of the control is sent out on the SPI_MOSI line.

The length of the control word can be in the range 1 to 16 bits and is set by writing bSpi_CFS (bits 15:12) in rSpi_CTRLR0 register. The remainder of the control word is transmitted (propagated on the falling edge of SPI_CLK) by the SPI master. During this transmission, no data are present on the serial master's SPI_MISO line.

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

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

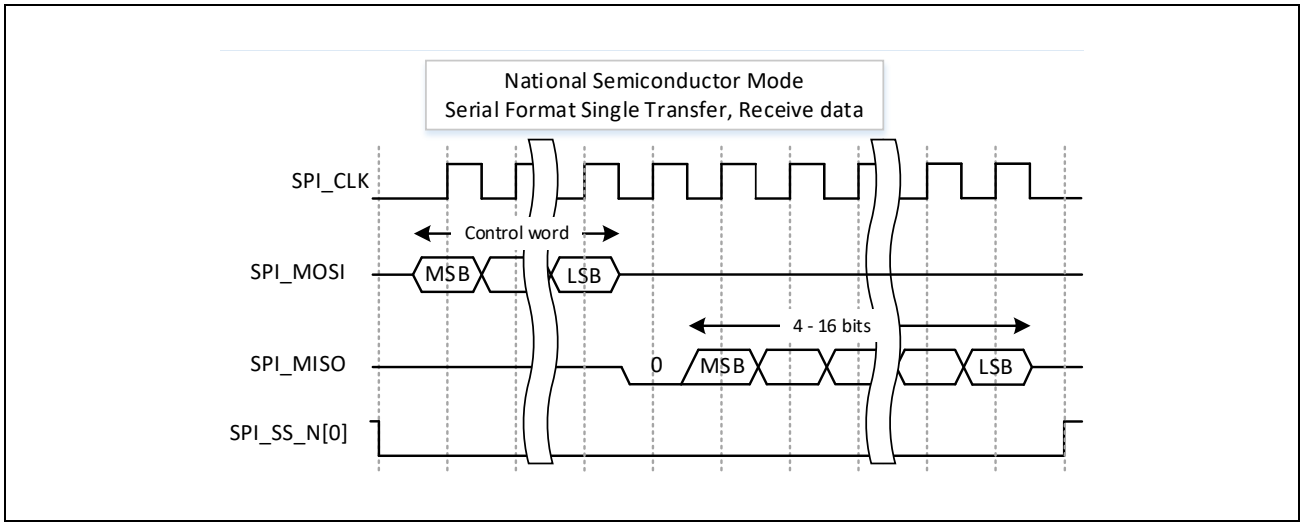

Figure 2.12 National Semiconductor Mode, Single Transfer, Receive Data

Continuous transfers from the Microwire protocol can be sequential or non-sequential, and are controlled by the bSpi_MWMOD (bit 0) in the rSpi_MWCR.

No sequential continuous transfers occur as illustrated in figure below, with the control word for the next transfer following immediately after the LSB of the current data word.

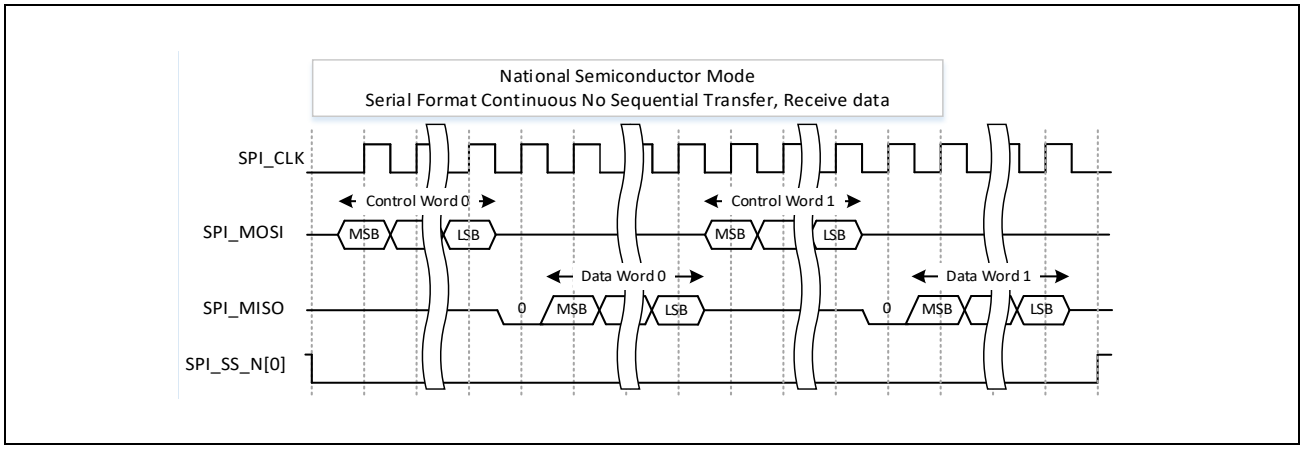

Figure 2.13 National Semiconductor Mode, Continuous No Sequential Transfer, Receive Data

During sequential continuous transfers, only one control word is transmitted from the SPI master. The transfer is started in the same manner as with no sequential read operations, but the cycle is continued to read further data. The slave device automatically increments its address pointer to the next location and continues to provide data from that location. Any number of locations can be read in this manner, the SPI master terminates the transfer when the number of words received is equal to the value in the rSpi_CTRLR1 register plus 1.

The timing diagram in figure below and example show a continuous sequential read of two data frames from the external slave device.

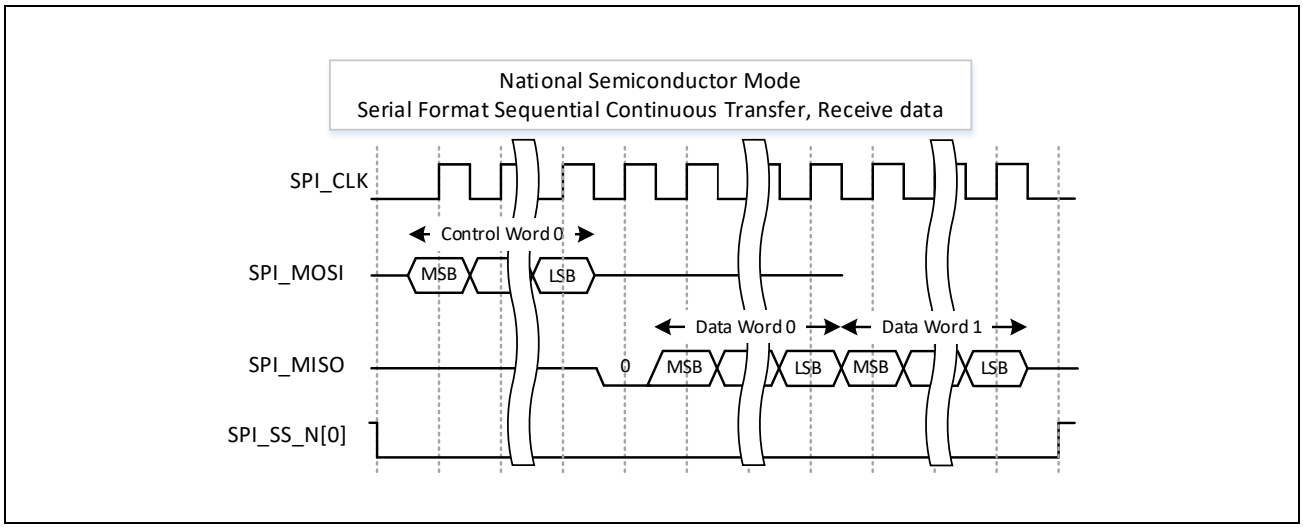

Figure 2.14 National Semiconductor Mode, Sequential Continuous Transfer, Receive Data

When bSpi_MDD = 1, this indicates that the SPI master transmits data to the external serial slave. Immediately after the LSB of the control word is transmitted, the SPI master begins transmitting the data frame to the slave peripheral. The figure below shows the timing diagram for a single serial master write to an external serial slave.

CAUTION

The SPI controller does not support continuous sequential Microwire writes, where bSpi_MDD = 1 and bSpi_MWMOD = 1.

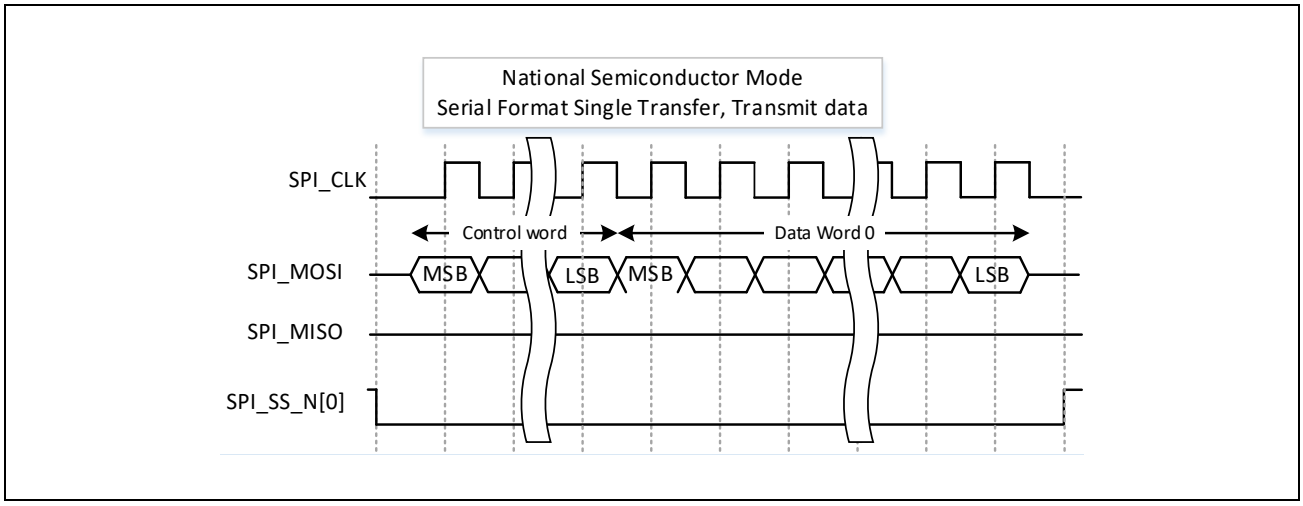

Figure 2.15 National Semiconductor Mode, Single Transfer, Transmit Data

Continuous transfers occur as shown in figure below, with the control word for the next transfer following immediately after the LSB of the current data word.

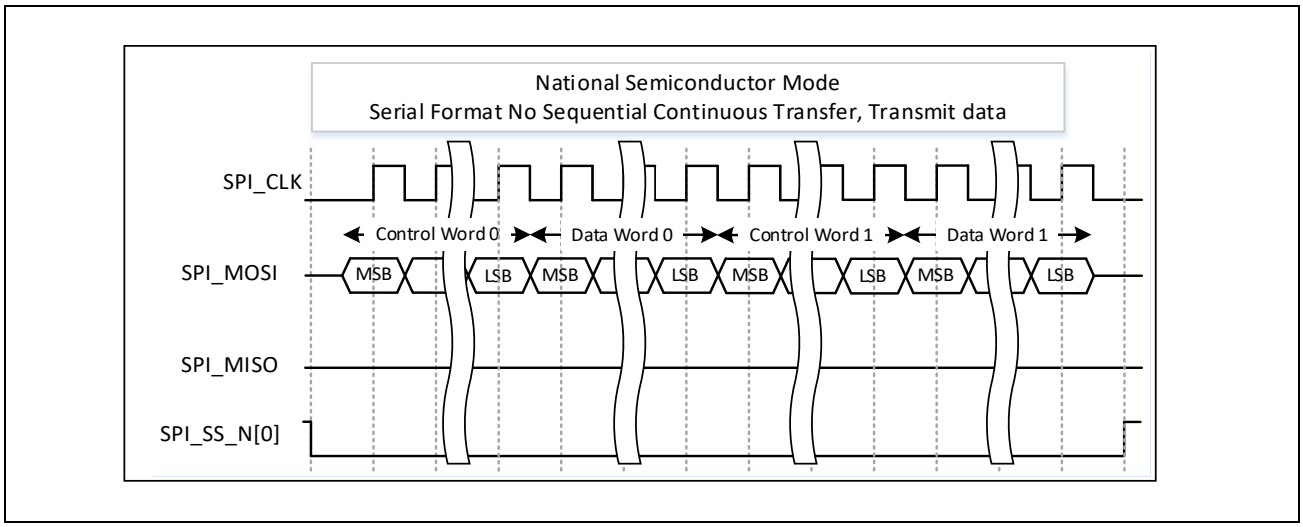

Figure 2.16 National Semiconductor Mode, Continuous No Sequential Transfer, Transmit Data

The Microwire handshaking interface can also be enabled for SPI master write operations to external serial slave devices. To enable the handshaking interface, you must write 1 into the bSpi_MWHS (bit 2) on the rSpi_MWCR register. When bSpi_MWHS is set to 1, the SPI master checks for a ready status from the slave device before completing the transfer, or transmitting the next control word for continuous transfers.

The figure below shows an example of a continuous Microwire transfer with the handshaking interface enabled.

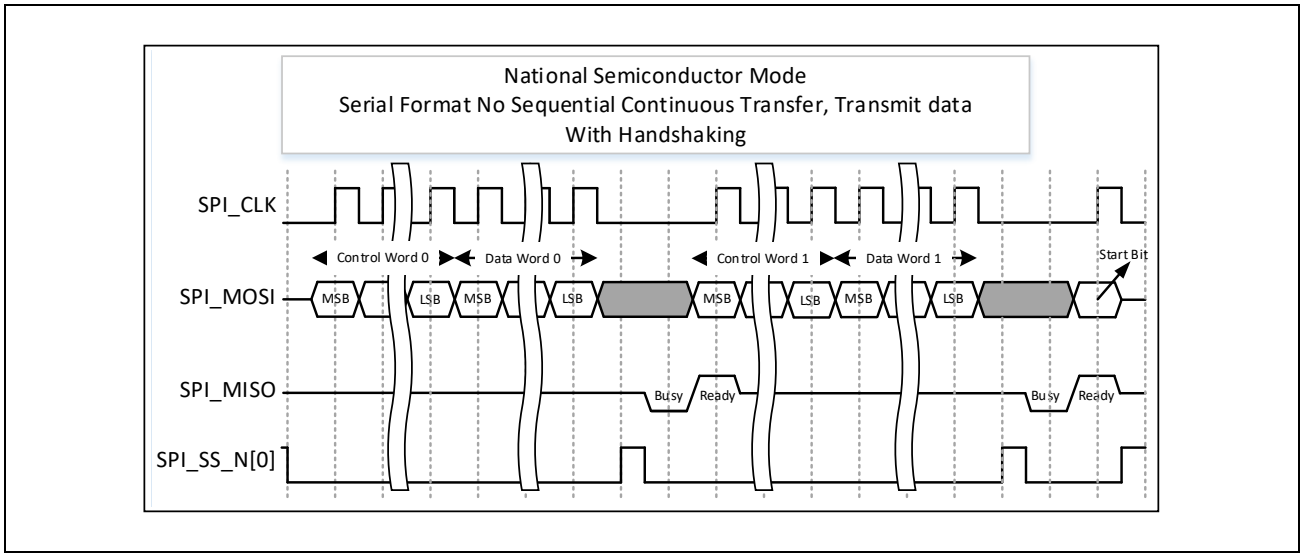

Figure 2.17 National Semiconductor Mode, Continuous No Sequential Transfer, Transmit Data, Handshaking

After the first data word has been transmitted to the serial slave device, the SPI master polls the SPI_MISO input waiting for a ready status from the slave device. Upon reception of the ready status, the SPI master begins transmission of the next control word. After transmission of the last data frame has completed, the SPI master transmits a start bit to clear the ready status of the slave device before completing the transfer.

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

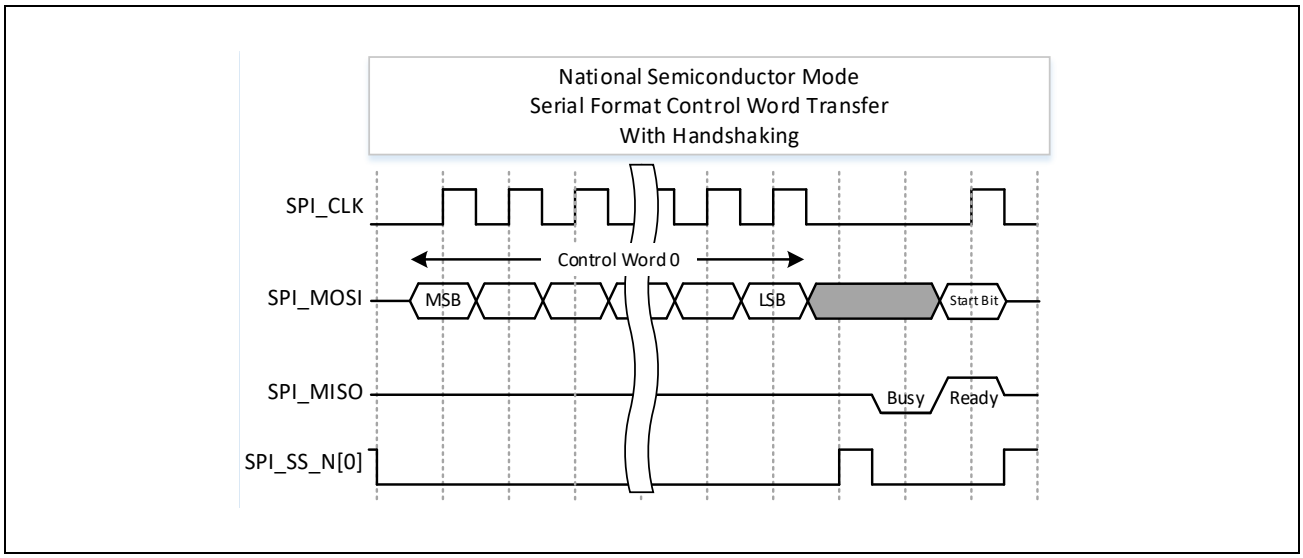

Figure 2.18 National Semiconductor Mode, Control Word Transfer, Handshaking

When the SPI controller is a slave, data transmission begins with the falling edge of the slave select signal (SPI_SS_N[0]). One-half serial clock (SPI_CLK) period later, the first bit of the control is present on the SPI_MOSI line. The length of the control word can be in the range of 1 to 16 bits and is set by writing bSpi_CFS in the rSpi_CTRLR0 register.

The bSpi_CFS must be set to the size of the expected control word from the serial master. The remainder of the control word is received (captured on the rising edge of SPI_CLK) by SPI slave. During this reception, no data are driven on the serial slave's SPI_MISO line.

The direction of the data word is controlled by the bSpi_MDD (bit 1) in the rSpi_MWCR register. When bSpi_MDD = 0, this indicates that the SPI slave is to receive data from the external serial master. Immediately after the control word is transmitted, the serial master begins to drive the data frame onto the SPI slave SPI_MOSI line. Data are propagated on the falling edge of the serial clock and captured on the rising edge. The slave select signal is held active low during the transfer and is de-asserted one-half clock cycle later after the data are transferred.

The figure below shows the timing diagram for single SPI slave read from an external serial master.

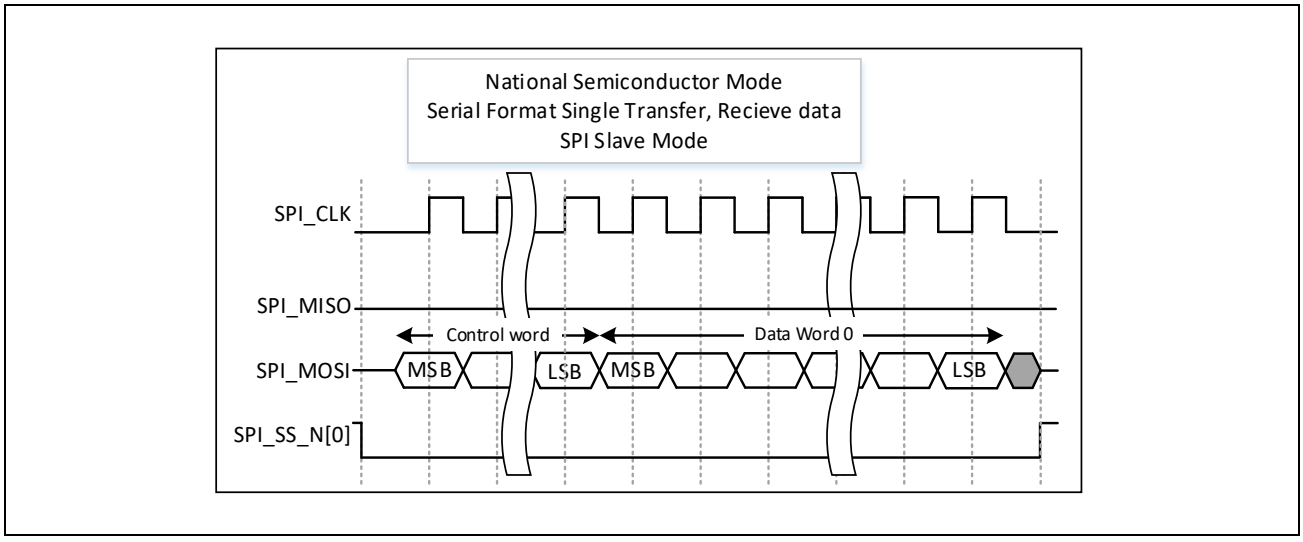

Figure 2.19 National Semiconductor Mode, Single Transfer, Receive Data, SPI controller in Slave Mode

When bSpi_MDD = 1, this indicates that the SPI slave transmits data to the external serial master. Immediately after the LSB of the control word is transmitted, the SPI slave transmits a dummy 0 bit, followed by the 4 to 16 bits data frame on the SPI_MISO line.

The figure below shows the timing diagram for a single SPI slave write to an external serial master.

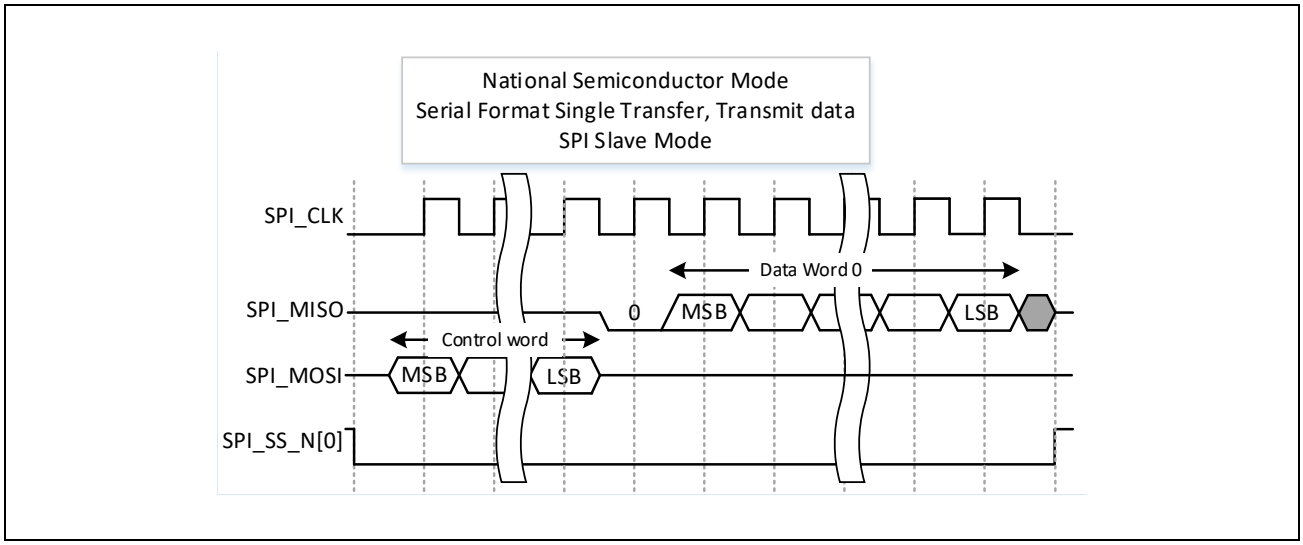

Figure 2.20 National Semiconductor Mode, Single Transfer, Transmit Data, SPI controller in Slave Mode

Continuous transfers for a SPI slave occur in the same way as those specified for the SPI master. The SPI slave does not support the handshaking interface, as there is never a busy period.

2.5.12 DMA Control

The SPI controller has DMA capability. It has a request 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. The DMA always transfers data using DMA burst transactions if possible, for efficiency.

CAUTION

In this mode, DMA controller must be configured in peripheral flow controller mode.

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

- rSpi_DMACR: Control register to enable DMA operation.
- rSpi_TDMACR, rSpi_RDMACR: Control register to:
  - Configuration of DMA operation (block size & burst transaction size).
  - Start & stop of DMA transfer.
- rSpi_DMATDLR: Register to set the transmit FIFO level at which a DMA request is made.
- rSpi_DMARDLR: Register to set the receive FIFO level at which a DMA request is made.

To enable the DMA Controller interface on the SPI controller & Enables the transmit request interface:

- Writing a 1 into the bSpi_TDMAE of rSpi_DMACR register
- Writing a 1 into the bSpi_TDMAE1 of rSpi_TDMACR register

To enable the DMA Controller interface on the SPI controller & Enables the receive request interface:

- Writing a 1 into the bSpi_RDMAE of rSpi_DMACR register
- Writing a 1 into the bSpi_RDMAE1 of rSpi_RDMACR register

To configure the block size and burst transaction length:

- Programming the block size into the DEST_BLOCK_SIZE/SRC_BLOCK_SIZE field of the rSpi_TDMACR & rSpi_RDMACR registers for Transmit FIFO and Receive FIFO, respectively.
- Programming the burst transaction length into the DEST_BURST_SIZE/SRC_BURST_SIZE field of the rSpi_TDMACR & rSpi_RDMACR registers for Transmit FIFO and Receive FIFO, respectively.

2.5.12.1 Overview on DMA Operation

The SPI controller must be programmed by the processor with the number of data items (block size) that are to be transmitted or received by the SPI controller. This is programmed into the DEST_BLOCK_SIZE/SRC_BLOCK_SIZE field of the rSpi_TDMACR & rSpi_RDMACR registers of SPI controller for Transmit FIFO and Receive FIFO, respectively. The block is broken into a number of transactions, each initiated by a request from the SPI controller.

The DMA Controller & the SPI controller must also be programmed with the number of data byte by burst transaction to be transferred for each DMA request. This is also known as the burst transaction length, and is programmed into:

DMA controller:

the DEST_MSIZE/SRC_MSIZE fields of the DMAC.CTL[n] register for Transmit FIFO and Receive FIFO, respectively.

SPI controller:

The DEST_BURST_SIZE/SRC_BURST_SIZE field of the rSpi_TDMACR & rSpi_RDMACR registers of SPI controller for Transmit FIFO and Receive FIFO, respectively.

CAUTION

- The burst transaction size must have the same values on DMA Controller and SPI controller.
- The source and destination transfer width settings in the DMA Controller, DMAC.CTL[n].SRC_TR_WIDTH and DMAC.CTL[n].DST_TR_WIDTH should be set to 3'b001 because the SPI FIFOs are 16 bits width.

2.5.12.2 Transmit Watermark Level and Transmit FIFO Underflow

During SPI serial transfers, transmit FIFO requests are made to the DMA controller whenever the number of entries in the transmit FIFO is less than or equal to the DMA transmit data level register rSpi_DMATDLR value, this is known as the watermark level. The DMA responds by writing a burst of data to the transmit FIFO buffer, of length DMAC.CTL[n].DEST_MSIZE = DEST_BURST_SIZE.

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 will run out of data (underflow). To prevent this condition, the user must set the watermark level correctly.

2.5.12.3 Choosing the Transmit Watermark Level

Consider the example where the assumption is made:

- DEST_BURST_SIZE = DMAC.CTL[n].DEST_MSIZE = FIFO_DEPTH − bSpi_DMATDLR
- Here the number of data items to be transferred in a DMA burst is equal to the empty space in the Transmit FIFO.

Consider two different watermark level settings:

- DEST_BLOCK_SIZE / DEST_BURST_SIZE = 42/14 = 3
- The number of burst transactions in the DMA block transfer is 3. But the watermark level, bSpi_DMATDLR, is quite low. Therefore, the probability of an SPI underflow is high where the SPI serial transmit line needs to transmit data, but where there is no data left in the transmit FIFO. This occurs because the DMA controller has not had time to service the DMA request before the transmit FIFO becomes empty.

In the figure below, the number of burst transactions needed equals the block size divided by the number of data items per burst.

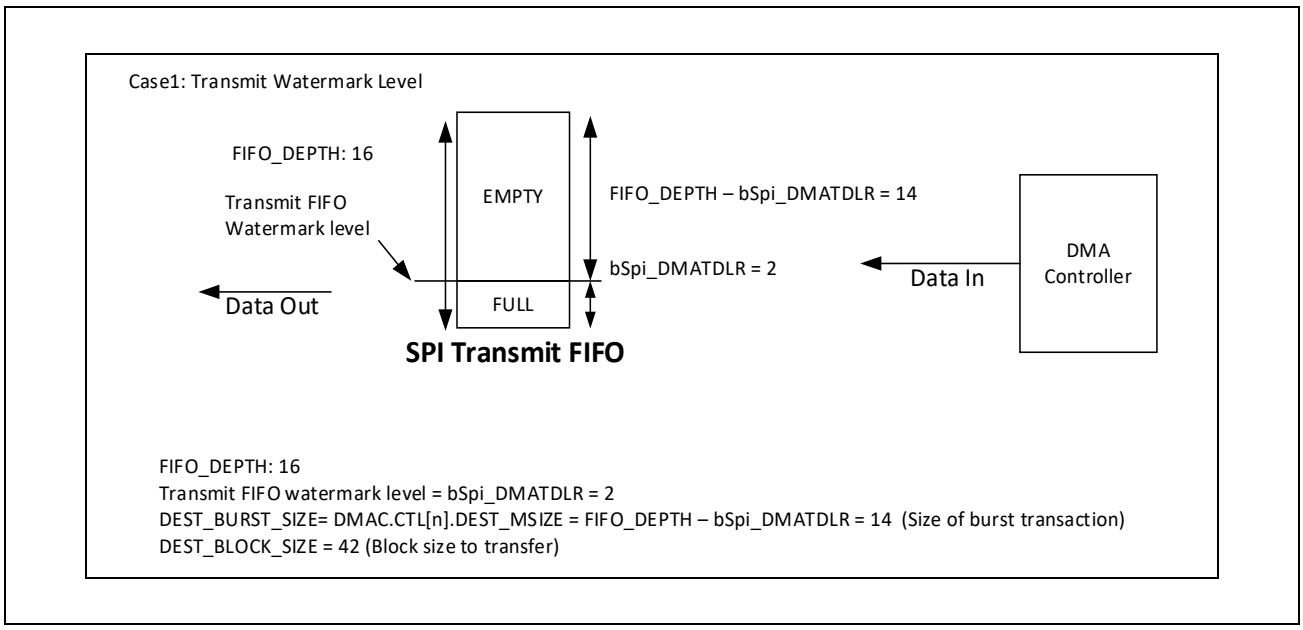

Figure 2.21 SPI Case1: Transmit Watermark Level

In the second case, the number of burst transactions in Block:

- DEST_BLOCK_SIZE / DEST_BURST_SIZE = 42/2 = 21
- In this block transfer, there are 21 destination burst transactions in a DMA block transfer. But the watermark level, bSpi_DMATDLR is very high. Therefore, the probability of an SPI underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the SPI 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 request bursts per block and worse bus utilization than the former case.

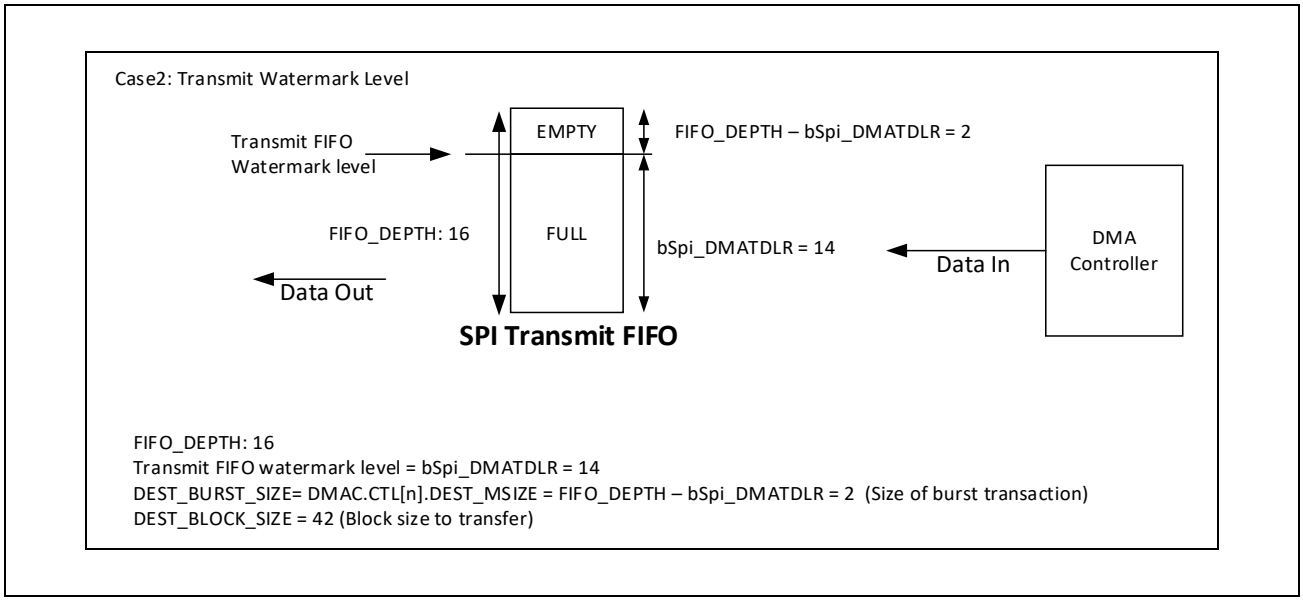

Figure 2.22 SPI Case2: Transmit Watermark Level

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 SPI controller transmits data to the rate at which the DMA controller 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 bus layer, increases the rate at which the DMA controller can respond to burst transaction requests. This in turn allows the user to decrease the watermark level, which improves bus utilization without compromising the probability of an underflow occurring.

2.5.12.4 Selecting DEST_MSIZE and Transmit FIFO Overflow

It may cause underflow when there is not enough space in the SPI transmit FIFO to service the destination burst request.

Therefore, for optimal operation, we must configure:

- DMAC.CTL[n].DEST_MSIZE = DEST_BURST_SIZE = 4
- Set bSpi_DMATDLR = 8

or

- DMAC.CTL[n].DEST_MSIZE = DEST_BURST_SIZE = 8
- Set bSpi_DMATDLR = 8

2.5.12.5 Receive Watermark Level and Receive FIFO Overflow

During SPI serial transfers, receive FIFO requests are made to the DMA controller whenever the number of entries in the receive FIFO is at or above the DMA Receive Data Level Register; that is bSpi_DMARDLR+1.

This is known as the watermark level. The DMA controller responds by reading a burst of data to the receive FIFO buffer of length SRC_BURST_SIZE = DMAC.CTL[n].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 will fill with data (overflow). To prevent this condition, the user must correctly set the watermark level.

2.5.12.6 Choosing the Receive Watermark Level

Similar to choosing the transmit watermark level described earlier, the receive watermark level, bSpi_DMARDLR+1, should be set to minimize the probability of overflow, as shown in figure below.

It is a tradeoff between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

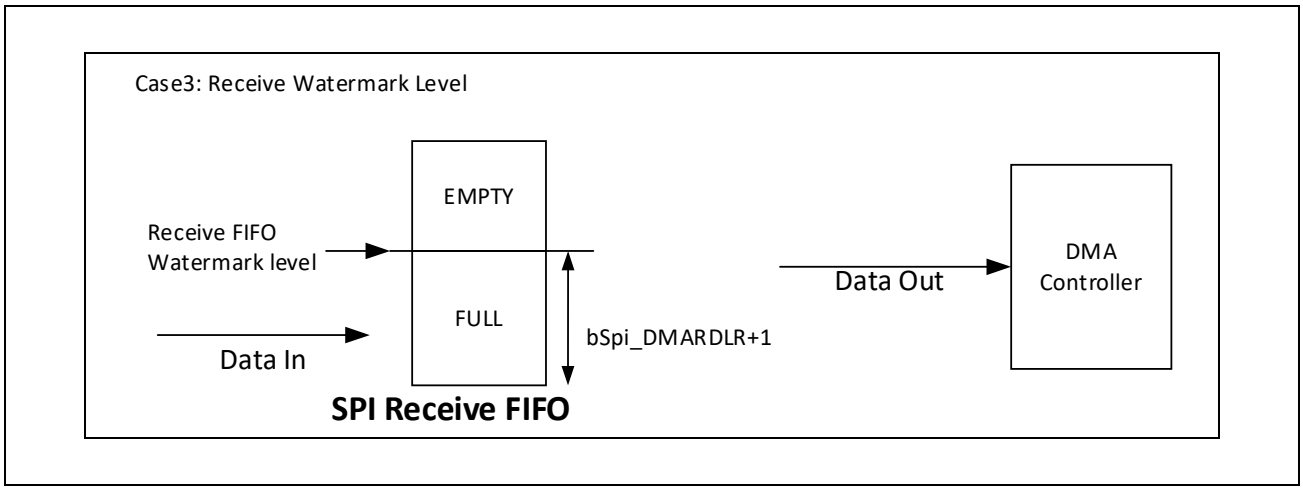

Figure 2.23 SPI Case3: Receive Watermark Level

2.5.12.7 Selecting SRC_MSIZE and Receive FIFO Underflow

It may cause underflow when there is not enough data to service the source burst request.

Therefore, for optimal operation, we must configure:

- DMAC.CTL[n].SRC_MSIZE = SRC_BURST_SIZE = 4
- Set bSpi_DMARDLR = 3

or

- DMAC.CTL[n].SRC_MSIZE = SRC_BURST_SIZE = 8
- Set bSpi_DMARDLR = 7

2.6 Usage Notes

2.6.1 Programming Consideration

CAUTION

- The BUSY status is not set when the data are written into the transmit FIFO. This bit gets set only when the target slave has been selected and the transfer is underway.
- After writing data into the transmit FIFO, the shift logic does not begin the serial transfer until a positive edge of the SPI_CLK signal is present. The delay in waiting for this positive edge depends on the baud rate of the serial transfer.
- Before polling the BUSY status, you should first poll the bSpi_TFE status (waiting for 1) or wait for rSpi_BAUDR × SPI_PCLK clock cycles.
- By polling the BUSY status (rSpi_SR.bSpi_BUSY), it is possible to determine when the serial transfer has completed.

2.6.1.1 Programming Master SPI in Motorola & Texas Mode

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

When the transfer mode is Transmit and Receive Mode or Transmit Only Mode (bSpi_TMOD = 2'b00 or bSpi_TMOD = 2'b01, respectively), transfers are terminated by the shift control logic when the transmit FIFO is empty.

For continuous data transfers, you must ensure that the transmit FIFO buffer does not become empty before all the data have been transmitted. The transmit FIFO threshold level (rSpi_TXFTLR) is used to early interrupt (iSpi_TXE_Int) the processor indicating that the transmit FIFO buffer is nearly empty. When a DMA is used for APB accesses, the transmit data level (rSpi_DMATDLR) can be used to early request the DMA Controller, indicating that the transmit FIFO is nearly empty. The FIFO can then be refilled with data to continue the serial transfer. The user may also write a block of data (at least two FIFO entries) into the transmit FIFO before enabling a serial slave by bSpi_HardwareSS. This ensures that serial transmission does not begin until the number of data frames that make up the continuous transfer are present in the transmit FIFO.

When the transfer mode is Receive Only Mode (bSpi_TMOD = 2'b10), a serial transfer is started by writing one "dummy" data word into the transmit FIFO when a serial slave is selected. The SPI_MOSI output from the SPI is held at a constant logic level for the duration of the serial transfer. The transmit FIFO is popped only once at the beginning and may remain empty for the duration of the serial transfer. The end of the serial transfer is controlled by the "number of data frames" (bSpi_NDF) field in the Control Register 1 (rSpi_CTRLR1)

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

When the transfer mode is EEPROM Read Mode (bSpi_TMOD = 2'b11), a serial transfer is started by writing the opcode and/or address into the transmit FIFO when a serial slave (EEPROM) is selected. The opcode and address are transmitted to the EEPROM device, after which read data is received from the EEPROM device and stored in the receive FIFO. The end of the serial transfer is controlled by the bSpi_NDF in the Control Register 1 (rSpi_CTRLR1).

CAUTION

EEPROM read mode is not supported when the SPI controller is configured to be in the Texas mode.

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

A typical Software flow for completing a serial transfer from the SPI master is outlined as follows:

1. If the SPI controller is enabled, disable it by writing 0 to the SPI Enable (bSpi_SSIENR).
2. Set up the SPI control registers for the transfer; these registers can be set in any order.
   - Write Control Register 0 (rSpi_CTRLR0)
     (For Motorola transfers, the serial clock polarity and serial clock phase parameters must be set identical to target slave device.)
   - If the transfer mode is Receive Only Mode, write Control Register 1 (rSpi_CTRLR1) with the number of frames in the transfer minus 1. For example, if you want to receive four data frames, write this register with 3.
   - Write rSpi_BAUDR
     To set the baud rate for the transfer.
   - Write rSpi_TXFTLR and rSpi_RXFTLR registers
     To set FIFO threshold levels.
   - Write rSpi_IMR register
     To set up interrupt masks.
   - You can write rSpi_SER register
     Can be written here to enable the target slave for selection. If a slave is enabled here, the transfer begins as soon as one valid data entry is present in the transmit FIFO. If no slaves are enabled prior to writing to the Data Register (bSpi_DR), the transfer does not begin until a slave is enabled.
3. Enable the SPI controller by writing 1 to the bSpi_SSIENR
4. Write data for transmission to the target slave into the transmit FIFO (write rSpi_DR).
   - If no slaves were enabled in the rSpi_SER register at this point, enable it now to begin the transfer.
5. Poll the bSpi_BUSY status to wait for completion of the transfer. The bSpi_BUSY status cannot be polled immediately, for more information, see **Section 2.6.1, Programming Consideration**.
   - If a transmit FIFO empty interrupt request is made, write the transmit FIFO (write rSpi_DR).
   - If a receive FIFO full interrupt request is made, read the receive FIFO (read rSpi_DR).
6. The transfer is stopped by the shift control logic when the transmit FIFO is empty.
   - If the transfer mode is Receive Only Mode (bSpi_TMOD = 2’b10), the transfer is stopped by the shift control logic when the specified number of frames have been received.
   - When the transfer is done, the BUSY status is reset to 0.
7. If the transfer mode is not Transmit Only Mode (bSpi_TMOD != 2’b01), read the receive FIFO until it is empty.
8. Disable the SPI controller by writing 0 to bSpi_SSIENR.

The figure below shows a typical Software flow for starting a SPI master Motorola & Texas mode transfer. The diagram also shows the hardware flow inside the serial master component.

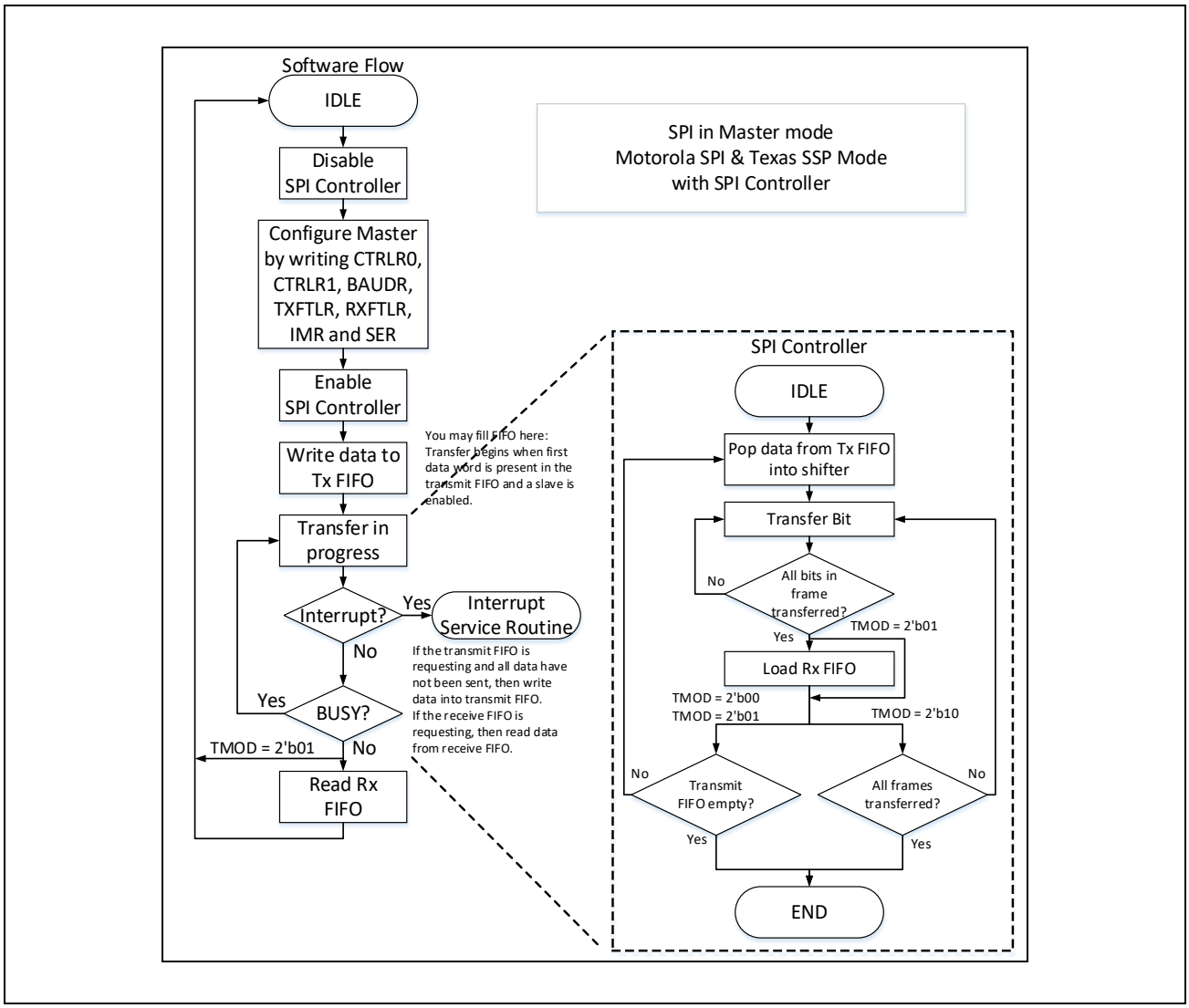

Figure 2.24 SPI Controller in Master Mode, Motorola & Texas Mode

2.6.1.2 Programming Master SPI in National Semiconductor Mode

National Semiconductor Microwire describes the Microwire serial protocol in detail, including timing diagrams and explaining how data are structured in the transmit and receive FIFOs before and after a serial transfer. Microwire serial transfers from the SPI master are controlled by the Microwire Control Register (rSpi_MWCR). The bSpi_MWHS enables and disables the Microwire handshaking interface. The bSpi_MDD controls the direction of the data frame (the control frame is always transmitted by the master and received by the slave). The bSpi_MWMOD defines whether the transfer is sequential or non-sequential.

All Microwire transfers are started by the SPI master when there is at least one control word in the transmit FIFO and a slave is enabled. When the SPI master transmits the data frame (bSpi_MDD = 1), the transfer is terminated by the shift logic when the transmit FIFO is empty.

When the SPI master receives the data frame (bSpi_MDD = 0), the termination of the transfer depends on the setting of the bSpi_MWMOD. If the transfer is non-sequential (bSpi_MWMOD = 0), it is terminated when the transmit FIFO is empty after shifting in the data frame from the slave. When the transfer is sequential (bSpi_MWMOD = 1), it is terminated by the shift logic when the number of data frames received is equal to the value in the rSpi_CTRLR1 register + 1.

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

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

1. If the SPI controller is enabled, disable it by writing 0 to bSpi_SSIENR.
2. Set up the SPI control registers for the transfer. These registers can be set in any order.
   - Write rSpi_CTRLR0 to set transfer parameters.
     If the transfer is sequential and the SPI master receives data, write rSpi_CTRLR1 with the number of frames in the transfer minus 1; for instance, if you want to receive four data frames, write this register with 3.
   - Write rSpi_BAUDR
     To set the baud rate for the transfer.
   - Write rSpi_TXFTLR and rSpi_RXFTLR registers
     To set FIFO threshold levels.
   - Write rSpi_IMR register
     To set up interrupt masks.
   - You can write rSpi_SER register
     To enable the target slave for selection. If a slave is enabled here, the transfer begins as soon as one valid data entry is present in the transmit FIFO. If no slaves are enabled prior to writing to the rSpi_DR register, the transfer does not begin until a slave is enabled.
3. Enable the SPI controller by writing 1 to the bSpi_SSIENR register.
4. If the SPI master transmits data, write the control and data words into the transmit FIFO (write rSpi_DR).
   - If the SPI master receives data, write the control word(s) into the transmit FIFO.
   - If no slaves were enabled in the rSpi_SER register at this point, enable now to begin the transfer.
5. Poll the bSpi_BUSY status to wait for completion of the transfer. The bSpi_BUSY status cannot be polled immediately, for more information, see **Section 2.6.1, Programming Consideration**.
   - If a transmit FIFO empty interrupt request is made, write the transmit FIFO (write rSpi_DR).

– If a receive FIFO full interrupt request is made, read the receive FIFO (read rSpi_DR).

6. The transfer is stopped by the shift control logic when the transmit FIFO is empty. If the transfer mode is sequential and the SPI master receives data, the transfer is stopped by the shift control logic when the specified number of data frames is received. When the transfer is done, the BUSY status is reset to 0.
7. If the SPI master receives data, read the receive FIFO until it is empty.
8. Disable the SPI controller by writing 0 to bSpi_SSIENR.

The figure below shows a typical Software flow for starting a SPI master Nation Semiconductor Microwire serial transfer. The diagram also shows the hardware flow inside the serial master component.

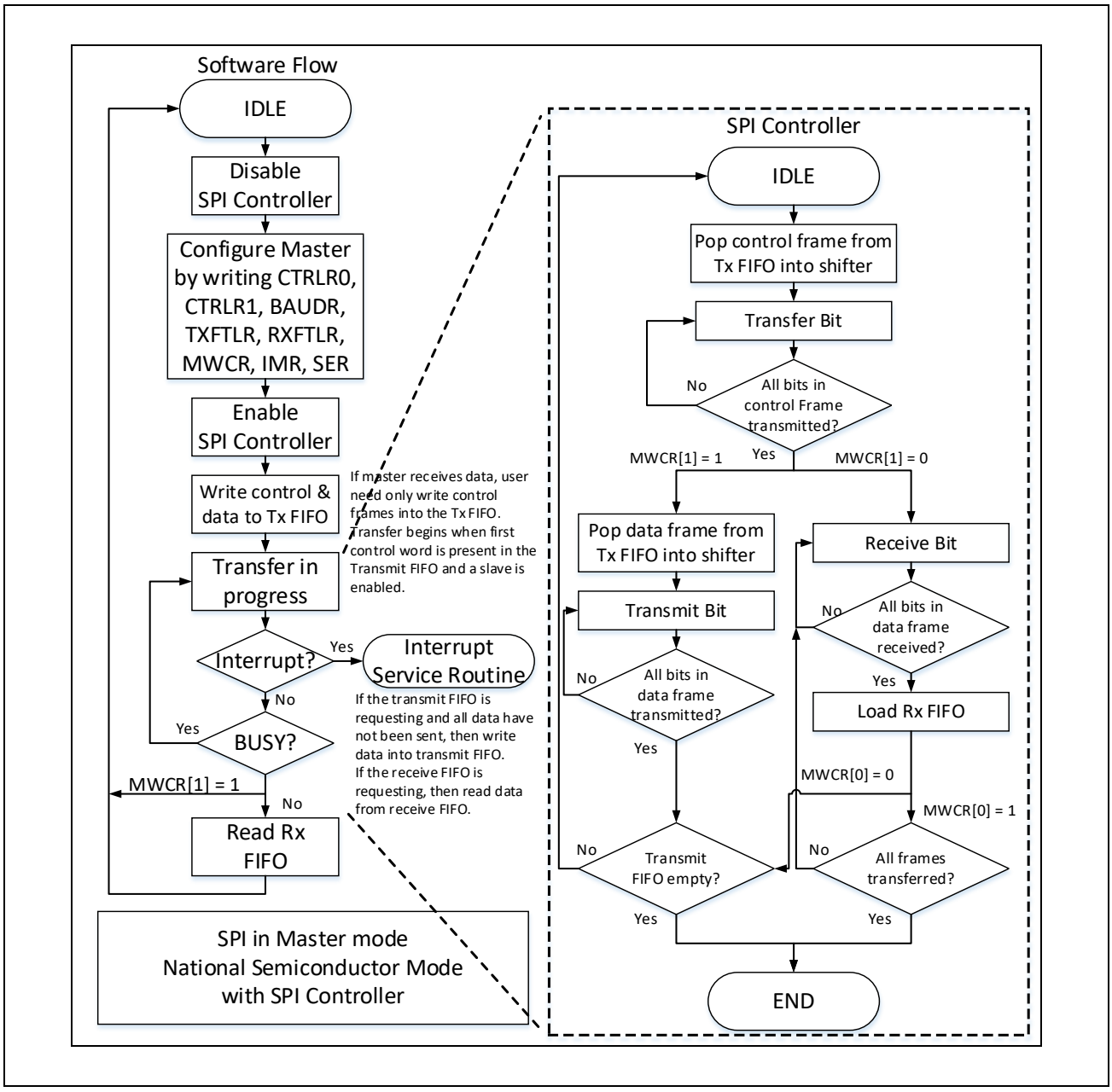

Figure 2.25 SPI Controller in Master Mode, National Semiconductor Mode

2.6.1.3 Programming Slave SPI in Motorola & Texas Mode

The sections Motorola Serial Peripheral Interface and Texas Instruments Synchronous Serial Protocol describe serial protocols, respectively. The sections also provide timing diagrams and information on how data are structured in the transmit and receive FIFOs before and after the serial transfer.

If the SPI slave is Receive Only Mode (bSpi_TMOD = 10), the transmit FIFO need not contain valid data because the data currently in the transmit shift register is resent each time the slave device is selected. You should mask the transmit FIFO empty interrupt when this mode is used.

If the SPI slave transmits data to the master, you must ensure that data exists in the transmit FIFO before a transfer is initiated by the serial master device. If the master initiates a transfer to the SPI slave when no data exists in the transmit FIFO, an error flag (bSpi_TXE) is set in the SPI status register, and the previously transmitted data frame is resent on SPI_MISO.

For continuous data transfers, you must ensure that the transmit FIFO buffer does not become empty before all the data have been transmitted. The transmit FIFO threshold level register (rSpi_TXFTLR) can be used to early interrupt (iSpi_TXE_Int) the processor, indicating that the transmit FIFO buffer is nearly empty.

When a DMA Controller is used for APB accesses, the DMA transmit data level register (rSpi_DMATDLR) can be used to early request the DMA Controller, indicating that the transmit FIFO is nearly empty. The FIFO can then be refilled with data to continue the serial transfer. The receive FIFO buffer should be read each time the receive FIFO generates a FIFO full interrupt request to prevent an overflow. The receive FIFO threshold level register (rSpi_RXFTLR) can be used to give early indication that the receive FIFO is nearly full.

When a DMA Controller is used for APB accesses, the DMA receive data level register (rSpi_DMARDLR) can be used to early request the DMA controller, indicating that the receive FIFO is nearly full.

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

1. If the SPI controller is enabled, disable it by writing 0 to bSpi_SSIENR.
2. Set up the SPI control registers for the transfer. These registers can be set in any order.
   - Write rSpi_CTRLR0
     For SPI transfers bSpi_SCPH and bSpi_SCPOL must be set identical to the master device.
   - Write rSpi_TXFTLR and rSpi_RXFTLR registers
     To set FIFO threshold levels.
   - Write rSpi_IMR register
     To set up interrupt masks.
3. Enable the SPI controller by writing 1 to the bSpi_SSIENR register.
4. If the transfer mode is Transmit and Receive Mode (bSpi_TMOD = 2’b00) or Transmit Only Mode (bSpi_TMOD = 2’b01), write data for transmission to the master into the transmit FIFO (Write rSpi_DR). If the transfer mode is Receive Only Mode (bSpi_TMOD = 2’b10), there is no need to write data into the transmit FIFO; the current value in the transmit shift register is retransmitted.
5. The SPI slave is now ready for the serial transfer. The transfer begins when the SPI slave is selected by a serial master device.
6. When the transfer is underway, the bSpi_BUSY status can be polled to return the transfer status.
   - If a transmit FIFO empty interrupt request is made, write the transmit FIFO (write rSpi_DR).
   - If a receive FIFO full interrupt request is made, read the receive FIFO (read rSpi_DR).

7. The transfer ends when the serial master removes the select input to the SPI slave.
   - When the transfer is completed, the bSpi_BUSY status is reset to 0.
8. If the transfer mode is not Transmit Only Mode (bSpi_TMOD != 2'b01), read the receive FIFO until empty.
9. Disable the SPI controller by writing 0 to bSpi_SSIENR.

The figure below shows a typical Software flow for a SPI slave Motorola & Texas mode serial transfer. The diagram also shows the hardware flow inside the serial- slave component.

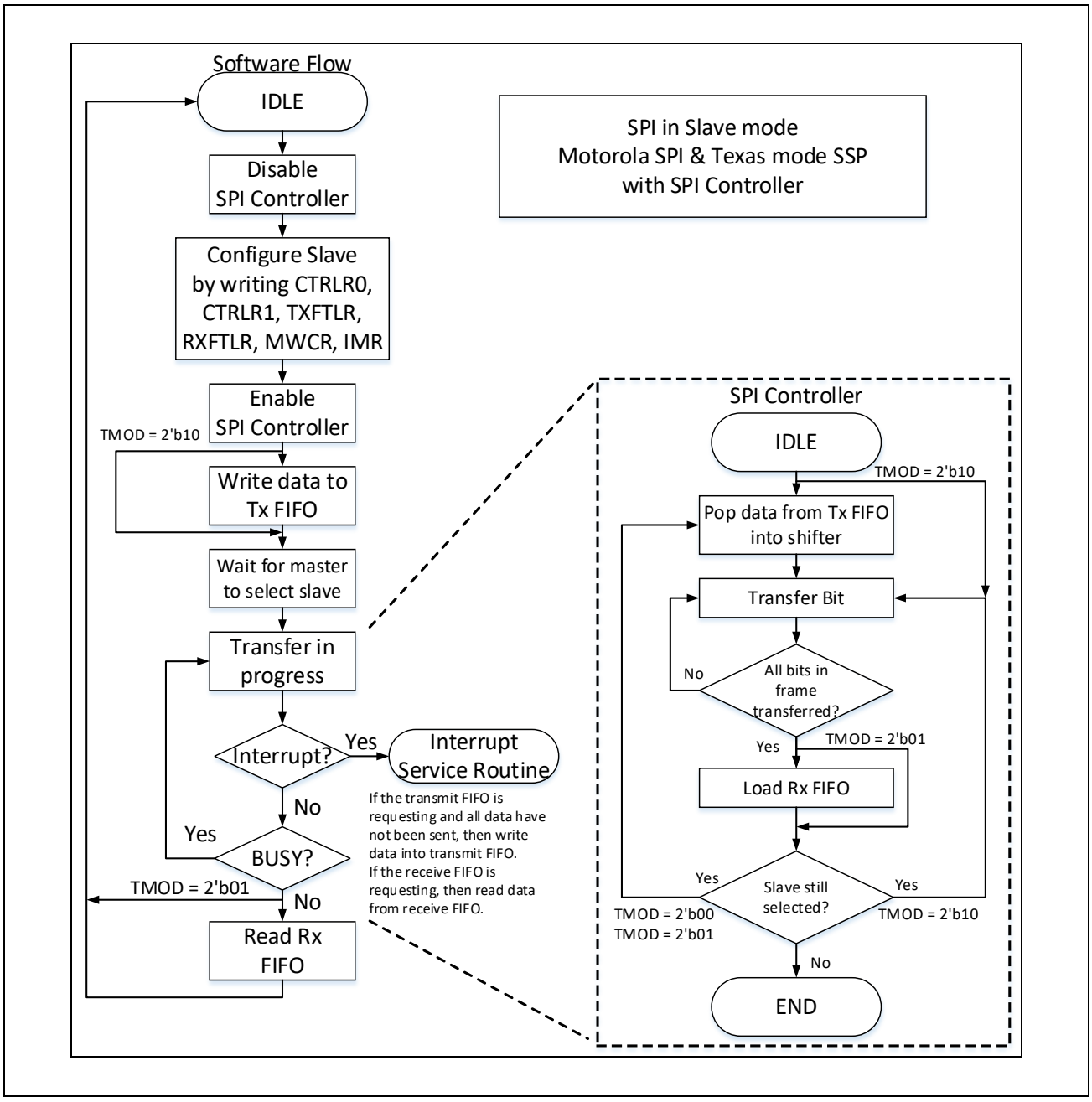

Figure 2.26 SPI Controller in Slave Mode, Motorola & Texas Mode

2.6.1.4 Programming Slave SPI in National Semiconductor Mode

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

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

See **Figure 2.26, SPI Controller in Slave Mode, Motorola & Texas Mode**.

Section 3 I2C

3.1 Overview

- 2 units
- Two speeds:
  - Standard mode (0 to 100 kb/s)
  - Fast mode (up to 400 kb/s)
- Separate 8×8bits transmit and 8x8bits receive FIFOs
- Master or slave I2C operation
- 7- or 10-bit addressing
- 7- or 10-bit combined format transfers
- Bulk transmit mode
- Transmit and receive buffers
- Interrupt or polled mode operation
- Programmable SDA hold time ($t_{HD;DAT}$)
- Serial reference clock
  - Programmable frequency up to 83.33 MHz
  - I2C1 and I2C2 share a same programmable integer divider

The figure below shows the interfaces of both I2C modules and its connections to other blocks.

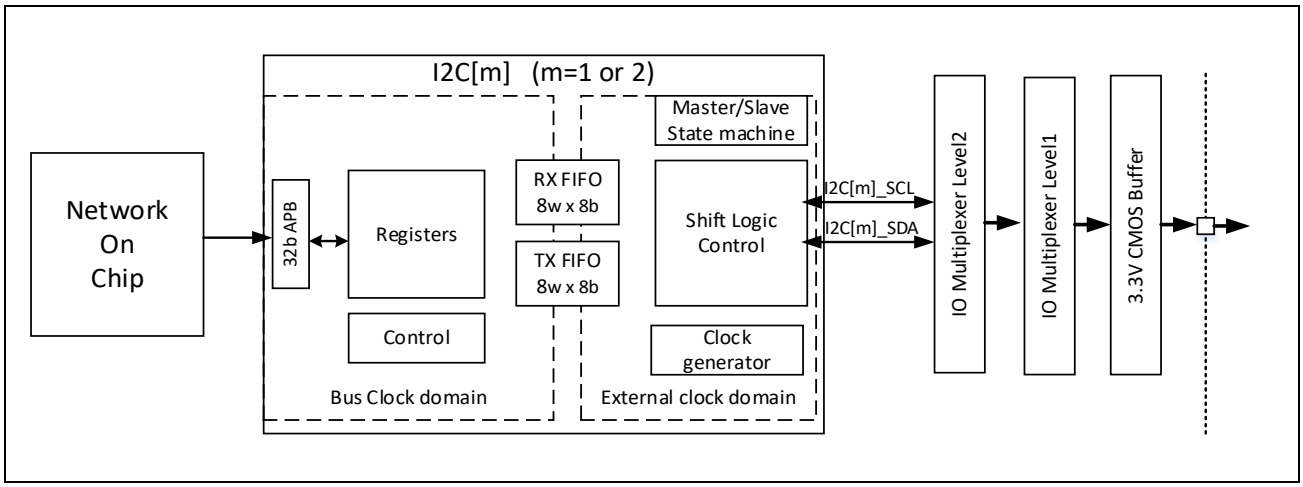

Figure 3.1 I2C Interfaces and Connections

3.2 Signal Interfaces

Signal Name	Input Output	Description
Clock		
I2C[m]_PCLK	Input	Internal bus clock (APB)
I2C[m]_SCLK	Input	Serial reference clock
Interrupt		
I2C[m]_Int	Output	Level sensitive interrupt output, Active High
External Signal		
I2C[m]_SCL	I/O	Serial clock (SCL)
I2C[m]_SDA	I/O	Serial data (SDA)

Note: m = 1 or 2.
Index removed style is mainly used in this chapter.
Ex) I2C_PCLK

3.3 Register Map

3.3.1 I2C1 Register Map

Table 3.1 I2C1 Register Map

Address	Register Symbol	Register Name
4006 3000h	IC_CON	I2C Control Register
4006 3004h	IC_TAR	I2C Target Address Register
4006 3008h	IC_SAR	I2C Slave Address Register
4006 3010h	IC_DATA_CMD	I2C Rx/Tx Data Buffer and Command Register
4006 3014h	IC_SS_SCL_HCNT	Standard mode I2C Clock SCL High Count Register
4006 3018h	IC_SS_SCL_LCNT	Standard mode I2C Clock SCL Low Count Register
4006 301Ch	IC_FS_SCL_HCNT	Fast mode I2C Clock SCL High Count Register
4006 3020h	IC_FS_SCL_LCNT	Fast mode I2C Clock SCL Low Count Register
4006 302Ch	IC_INTR_STAT	I2C Interrupt Status Register
4006 3030h	IC_INTR_MASK	I2C Interrupt Mask Register
4006 3034h	IC_RAW_INTR_STAT	I2C Raw Interrupt Status Register
4006 3038h	IC_RX_TL	I2C Receive FIFO Threshold Register
4006 303Ch	IC_TX_TL	I2C Transmit FIFO Threshold Register
4006 3040h	IC_CLR_INTR	Clear Combined and Individual Interrupt Register
4006 3044h	IC_CLR_RX_UNDER	Clear RX_UNDER Interrupt Register
4006 3048h	IC_CLR_RX_OVER	Clear RX_OVER Interrupt Register
4006 304Ch	IC_CLR_TX_OVER	Clear TX_OVER Interrupt Register
4006 3050h	IC_CLR_RD_REQ	Clear RD_REQ Interrupt Register
4006 3054h	IC_CLR_TX_ABRT	Clear TX_ABRT Interrupt Register
4006 3058h	IC_CLR_RX_DONE	Clear RX_DONE Interrupt Register
4006 305Ch	IC_CLR_ACTIVITY	Clear ACTIVITY Interrupt Register
4006 3060h	IC_CLR_STOP_DET	Clear STOP_DET Interrupt Register
4006 3064h	IC_CLR_START_DET	Clear START_DET Interrupt Register
4006 3068h	IC_CLR_GEN_CALL	Clear GEN_CALL Interrupt Register
4006 306Ch	IC_ENABLE	I2C Enable Register
4006 3070h	IC_STATUS	I2C Status Register
4006 3074h	IC_TXFLR	I2C Transmit FIFO Level Register
4006 3078h	IC_RXFLR	I2C Receive FIFO Level Register
4006 307Ch	IC_SDA_HOLD	I2C SDA Hold Time Length Register
4006 3080h	IC_TX_ABRT_SOURCE	I2C Transmit Abort Source Register
4006 3084h	IC_SLV_DATA_NACK_ONLY	Generate Slave Data NACK Register
4006 3094h	IC_SDA_SETUP	I2C SDA Setup Register
4006 3098h	IC_ACK_GENERAL_CALL	I2C ACK General Call Register
4006 309Ch	IC_ENABLE_STATUS	I2C Enable Status Register
4006 30A0h	IC_FS_SPKLEN	I2C Sm, Fm Spike Suppression Limit
4006 30A8h	IC_CLR_RESTART_DET	Clear RESTART_DET Interrupt Register
4006 30F4h	IC_COMP_PARAM_1	Component Parameter Register 1

3.3.2 I2C2 Register Map

Table 3.2 I2C2 Register Map

Address	Register Symbol	Register Name
4006 4000h	IC_CON	I2C Control Register
4006 4004h	IC_TAR	I2C Target Address Register
4006 4008h	IC_SAR	I2C Slave Address Register
4006 4010h	IC_DATA_CMD	I2C Rx/Tx Data Buffer and Command Register
4006 4014h	IC_SS_SCL_HCNT	Standard mode I2C Clock SCL High Count Register
4006 4018h	IC_SS_SCL_LCNT	Standard mode I2C Clock SCL Low Count Register
4006 401Ch	IC_FS_SCL_HCNT	Fast mode I2C Clock SCL High Count Register
4006 4020h	IC_FS_SCL_LCNT	Fast mode I2C Clock SCL Low Count Register
4006 402Ch	IC_INTR_STAT	I2C Interrupt Status Register
4006 4030h	IC_INTR_MASK	I2C Interrupt Mask Register
4006 4034h	IC_RAW_INTR_STAT	I2C Raw Interrupt Status Register
4006 4038h	IC_RX_TL	I2C Receive FIFO Threshold Register
4006 403Ch	IC_TX_TL	I2C Transmit FIFO Threshold Register
4006 4040h	IC_CLR_INTR	Clear Combined and Individual Interrupt Register
4006 4044h	IC_CLR_RX_UNDER	Clear RX_UNDER Interrupt Register
4006 4048h	IC_CLR_RX_OVER	Clear RX_OVER Interrupt Register
4006 404Ch	IC_CLR_TX_OVER	Clear TX_OVER Interrupt Register
4006 4050h	IC_CLR_RD_REQ	Clear RD_REQ Interrupt Register
4006 4054h	IC_CLR_TX_ABRT	Clear TX_ABRT Interrupt Register
4006 4058h	IC_CLR_RX_DONE	Clear RX_DONE Interrupt Register
4006 405Ch	IC_CLR_ACTIVITY	Clear ACTIVITY Interrupt Register
4006 4060h	IC_CLR_STOP_DET	Clear STOP_DET Interrupt Register
4006 4064h	IC_CLR_START_DET	Clear START_DET Interrupt Register
4006 4068h	IC_CLR_GEN_CALL	Clear GEN_CALL Interrupt Register
4006 406Ch	IC_ENABLE	I2C Enable Register
4006 4070h	IC_STATUS	I2C Status Register
4006 4074h	IC_TXFLR	I2C Transmit FIFO Level Register
4006 4078h	IC_RXFLR	I2C Receive FIFO Level Register
4006 407Ch	IC_SDA_HOLD	I2C SDA Hold Time Length Register
4006 4080h	IC_TX_ABRT_SOURCE	I2C Transmit Abort Source Register
4006 4084h	IC_SLV_DATA_NACK_ONLY	Generate Slave Data NACK Register
4006 4094h	IC_SDA_SETUP	I2C SDA Setup Register
4006 4098h	IC_ACK_GENERAL_CALL	I2C ACK General Call Register
4006 409Ch	IC_ENABLE_STATUS	I2C Enable Status Register
4006 40A0h	IC_FS_SPKLEN	I2C Sm, Fm Spike Suppression Limit
4006 40A8h	IC_CLR_RESTART_DET	Clear RESTART_DET Interrupt Register
4006 40F4h	IC_COMP_PARAM_1	Component Parameter Register 1

3.4 Register Description

3.4.1 IC_CON — I2C Control Register

Address: 4006 3000h (I2C1)
4006 4000h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	RX_FIFO_FULL_HLD_CTRL	TX_EMPTY_CTRL	STOP_DET_IFADDRESSED	IC_SLAVE_DISABLE	IC_RESTART_EN	IC_10BITADDR_MASTER_rd_only	IC_10BITADDR_SLAVE	SPEED		MASTER_MODE
Value after reset	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1

Table 3.3 IC_CON Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b10	Reserved		R
b9	RX_FIFO_FULL_HLD_CTRL	This bit controls whether the I2C controller should hold the bus when the Rx FIFO is physically full to its RX_BUFFER_DEPTH (depth = 8). 0: Disable bus hold 1: Enable bus hold	R/W
b8	TX_EMPTY_CTRL	This bit controls the generation of the TX_EMPTY interrupt, as described in the IC_RAW_INTR_STAT register.	R/W
b7	STOP_DET_IFADDRESSED	In slave mode: 1: Issues the STOP_DET interrupt only when it is addressed. 0: 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).	R/W
b6	IC_SLAVE_DISABLE	This bit controls whether I2C controller has its slave disabled. If this bit is set (slave is disabled), the I2C controller functions only as a master and does not perform any action that requires a slave. 0: Slave is enabled 1: Slave is disabled The slave disabled after reset is applied. ***Note*)** Software should ensure that if this bit is written with 0, then bit 0 should also be written with a 0.	R/W

Table 3.3 IC_CON Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b5	IC_RESTART_EN	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 I2C controller operations. 0: Disable 1: Enable When RESTART is disabled, the master is prohibited from performing the following functions: • Change direction within a transfer (split) • Send a START BYTE • Combined format transfers in 7-bit addressing modes • Read operation with a 10-bit address • Send multiple bytes per transfer By replacing RESTART condition followed by a STOP and a subsequent START condition, split operations are broken down into multiple I2C controller transfers. If the above operations are performed, it will result in setting bit 6 (TX_ABRT) of the IC_RAW_INTR_STAT register.	R/W
b4	IC_10BITADDR_MASTER_rd_only	Read only bit showing the bit addressing mode: the function of this bit is handled by bit 12 of IC_TAR register 0: 7-bit addressing 1: 10-bit addressing	R
b3	IC_10BITADDR_SLAVE	When acting as a slave, this bit controls whether the I2C controller responds to 7- or 10-bit addresses. 0: 7-bit addressing. The I2C controller 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 I2C controller responds to only 10-bit addressing transfers that match the full 10 bits of the IC_SAR register.	R/W
b2, b1	SPEED	These bits control at which speed the I2C controller operates; its setting is relevant only if one is operating the I2C controller in master mode. Hardware protects against illegal values being programmed by software. 1: Standard mode (≤100 kb/s) 2: Fast mode (≤400 kb/s)	R/W
b0	MASTER_MODE	This bit controls whether the I2C controller master is enabled. 0: Master disabled 1: Master enabled ***Note*)** Software should ensure that if this bit is written with “1” then bit 6 should also be written with a “1”.	R/W

3.4.2 IC_TAR — I2C Target Address Register

All bits can be dynamically updated as long as any set of the following conditions are true:

I2C controller is not enabled (IC_ENABLE[0] is set to 0);
or
I2C controller is enabled (IC_ENABLE[0]=1);
AND
I2C controller is not engaged in any Master (tx, rx) operation (IC_STATUS[5]=0);
AND
I2C controller is enabled to operate in Master mode (IC_CON[0]=1);
AND
there are NO entries in the TX FIFO (IC_STATUS[2]=1)

Address: 4006 3004h (I2C1)
4006 4004h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	IC_10BIT ADDR_M ASTER	SPECIA L	GC_OR _STAR T	IC_TAR									
Value after reset	X	X	X	1	0	0	0	0	0	1	0	1	0	1	0	1

Table 3.4 IC_TAR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b13	Reserved		R
b12	IC_10BITADDR_MAS TER	This bit controls whether the I2C controller starts its transfers in 7- or 10-bit addressing mode when acting as a master. 0: 7-bit addressing 1: 10-bit addressing	R/W
b11	SPECIAL	This bit indicates whether software performs a 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 GC_OR_START bit	R/W
b10	GC_OR_START	If bit 11 (SPECIAL) is set to 1, then this bit indicates whether a General Call or START byte command is to be performed by the I2C controller. 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 I2C controller remains in General Call mode until the SPECIAL bit value (bit 11) is cleared. 1: START BYTE	R/W
b9 to b0	IC_TAR	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.	R/W

3.4.3 IC_SAR — I2C Slave Address Register

Address: 4006 3008h (I2C1)
4006 4008h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	IC_SAR									
Value after reset	X	X	X	X	X	X	0	0	0	1	0	1	0	1	0	1

Table 3.5 IC_SAR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b10	Reserved		R
b9 to b0	IC_SAR	The IC_SAR holds the slave address when the I2C controller 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*)** This value 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.	R/W

3.4.4 IC_DATA_CMD — 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

Address: 4006 3010h (I2C1)
4006 4010h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	RESTART	STOP	CMD	DAT							
Value after reset	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0

Table 3.6 IC_DATA_CMD Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b11	Reserved		R
b10	RESTART	This bit controls whether a RESTART is issued before the byte is sent or received. 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.	W
b9	STOP	This bit controls whether a STOP is issued after the byte is sent or received. 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.	W
b8	CMD	This bit controls whether a read or a write is performed. This bit does not control the direction when the I2C controller acts as a slave. It controls only the direction when it acts as a master. 1: Read 0: Write 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 CPU data is to be transmitted and as DAT or IC_DATA_CMD[7:0]. 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. ***Note*)** It is possible that while attempting a master I2C read transfer on I2C controller, a RD_REQ interrupt may have occurred simultaneously due to a remote I2C master addressing the I2C controller. In this type of scenario, I2C controller ignores the IC_DATA_CMD write, generates a TX_ABRT interrupt, and waits to service the RD_REQ interrupt.	W

Table 3.6 IC_DATA_CMD Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b7 to b0	DAT	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 I2C controller. However, when you read this register, these bits return the value of data received on the I2C controller interface.	R/W

3.4.5 IC_SS_SCL_HCNT — Standard mode I2C Clock SCL High Count Register

Address: 4006 3014h (I2C1)
4006 4014h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	IC_SS_SCL_HCNT															
Value after reset	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	0

Table 3.7 IC_SS_SCL_HCNT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved		R
b15 to b0	IC_SS_SCL_HCNT	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 mode. 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. ***Note*)** This register must not be programmed to a value higher than 65525, because I2C controller uses a 16-bit counter to flag an I2C bus idle condition when this counter reaches a value of IC_SS_SCL_HCNT + 10.	R/W

3.4.6 IC_SS_SCL_LCNT — Standard mode I2C Clock SCL Low Count Register

Address: 4006 3018h (I2C1)
4006 4018h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	IC_SS_SCL_LCNT															
Value after reset	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0

Table 3.8 IC_SS_SCL_LCNT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved		R
b15 to b0	IC_SS_SCL_LCNT	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 mode. 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.	R/W

3.4.7 IC_FS_SCL_HCNT — Fast mode I2C Clock SCL High Count Register

Address: 4006 301Ch (I2C1)
4006 401Ch (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	IC_FS_SCL_HCNT															
Value after reset	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

Table 3.9 IC_FS_SCL_HCNT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved		R
b15 to b0	IC_FS_SCL_HCNT	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. 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.	R/W

3.4.8 IC_FS_SCL_LCNT — Fast mode I2C Clock SCL Low Count Register

Address: 4006 3020h (I2C1)
4006 4020h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	IC_FS_SCL_LCNT															
Value after reset	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1

Table 3.10 IC_FS_SCL_LCNT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved		R
b15 to b0	IC_FS_SCL_LCNT	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 mode. 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.	R/W

3.4.9 IC_INTR_STAT — 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.

Address: 4006 302Ch (I2C1)
4006 402Ch (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	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
Value after reset	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.11 IC_INTR_STAT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved	Refer to **Section 3.4.11, IC_RAW_INTR_STAT — I2C Raw Interrupt Status Register** for detailed descriptions of these bits.	R
b13	R_MASTER_ON_HOLD		R
b12	R_RESTART_DET		R
b11	R_GEN_CALL		R
b10	R_START_DET		R
b9	R_STOP_DET		R
b8	R_ACTIVITY		R
b7	R_RX_DONE		R
b6	R_TX_ABRT		R
b5	R_RD_REQ		R
b4	R_TX_EMPTY		R
b3	R_TX_OVER		R
b2	R_RX_FULL		R
b1	R_RX_OVER		R
b0	R_RX_UNDER		R

3.4.10 IC_INTR_MASK — I2C Interrupt Mask Register

These bits mask their corresponding interrupt status bits. A value of 0 prevents a bit from generating an interrupt.

Address: 4006 3030h (I2C1)
4006 4030h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	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
Value after reset	X	X	0	0	1	0	0	0	1	1	1	1	1	1	1	1

Table 3.12 IC_INTR_MASK Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved		R
b13	M_MASTER_ON_HOLD	Mask bit for the R_MASTER_ON_HOLD interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b12	M_RESTART_DET	Mask bit for the R_RESTART_DET interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b11	M_GEN_CALL	Mask bit for the R_GEN_CALL interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b10	M_START_DET	Mask bit for the R_START_DET interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b9	M_STOP_DET	Mask bit for the R_STOP_DET interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b8	M_ACTIVITY	Mask bit for the R_ACTIVITY interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b7	M_RX_DONE	Mask bit for the R_RX_DONE interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b6	M_TX_ABRT	Mask bit for the R_TX_ABRT interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b5	M_RD_REQ	Mask bit for the R_RD_REQ interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b4	M_TX_EMPTY	Mask bit for the R_TX_EMPTY interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W

Table 3.12 IC_INTR_MASK Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3	M_TX_OVER	Mask bit for the R_TX_OVER interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b2	M_RX_FULL	Mask bit for the R_RX_FULL interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b1	M_RX_OVER	Mask bit for the R_RX_OVER interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W
b0	M_RX_UNDER	Mask bit for the R_RX_UNDER interrupt in IC_INTR_STAT register. 0: Disable interrupt 1: Enable interrupt	R/W

3.4.11 IC_RAW_INTR_STAT — 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 I2C controller.

Address: 4006 3034h (I2C1)
4006 4034h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	MASTER_ON_HOLD	RESTART_DET	GEN_CALL	START_DET	STOP_DET	RAW_INTR_ACTIVITY	RX_DONE	TX_ABRT	RD_REQ	TX_EMPTY	TX_OVER	RX_FULL	RX_OVER	RX_UNDER
Value after reset	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.13 IC_RAW_INTR_STAT Register Contents (1/3)

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved		R
b13	MASTER_ON_HOLD	If this bit is set to 1b, it indicates that the master is holding the bus and TX FIFO is empty.	R
b12	RESTART_DET	If this bit is set to 1b, it indicates that a RESTART condition has occurred on the I2C interface when I2C controller is operating in Slave mode and the slave is being addressed. ***Note*)** Following are exceptions where the RESTART_DET interrupt will not get generated. In the case of Start byte transfer, where the RESTART comes before the Address field as per the I2C protocol defined format, the Slave is still not in the addressed mode and hence will not generate the RESTART_DET interrupt.	R
b11	GEN_CALL	Set only when a General Call address is received and it is acknowledged. It stays set until it is cleared either by disabling I2C controller or when the CPU reads bit 0 of the IC_CLR_GEN_CALL register. I2C controller stores the received data in the Rx buffer.	R
b10	START_DET	If this bit is set to 1b, it indicates that a START or RESTART condition has occurred on the I2C interface regardless of whether I2C controller is operating in slave or master mode.	R
b9	STOP_DET	The behavior of the STOP_DET interrupt status differs based on the STOP_DET_IFADDRESSED selection in the IC_CON register. When STOP_DET_IFADDRESSED = 0 If this bit is set to 1b, it indicates that a STOP condition has occurred on the I2C interface regardless of whether I2C controller is operating in slave or master mode. In slave mode, a STOP_DET interrupt is generated irrespective of whether the slave is addressed or not. When STOP_DET_IFADDRESSED = 1 In Master Mode (MASTER_MODE = 1'b1), if this bit is set to 1b, it indicates that a STOP condition has occurred on the I2C interface. In Slave Mode (MASTER_MODE = 1'b0), STOP_DET interrupt is generated only if the slave is addressed. ***Note*)** During a general call address, this slave does not issue a STOP_DET interrupt if STOP_DET_IFADDRESSED = 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).	R

Table 3.13 IC_RAW_INTR_STAT Register Contents (2/3)

Bit Position	Bit Name	Function	R/W
b8	RAW_INTR_ACTIVITY	This bit captures I2C controller activity and stays set until it is cleared. There are four ways to clear it: • Disabling the I2C controller • 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 I2C controller module is idle, this bit remains set until cleared, indicating that there was activity on the bus.	R
b7	RX_DONE	When the I2C controller 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.	R
b6	TX_ABRT	If this bit is set to 1b, it indicates that I2C controller, 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 I2C controller 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 IC_CLR_TX_ABRT register is read. Once this read is performed, the Tx FIFO is then ready to accept more data bytes from the APB interface. Both Tx FIFO and Rx FIFO will flush on Transmit Abort.	R
b5	RD_REQ	This bit is set to 1 when I2C controller is acting as a slave and another I2C master is attempting to read data from I2C controller. The I2C controller 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.	R
b4	TX_EMPTY	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_ENABLE_STATUS.IC_EN = 0, this bit is set to 0.	R
b3	TX_OVER	Set during transmit if the transmit buffer is filled to its depth of 8 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_ENABLE_STATUS.IC_EN goes to 0, this interrupt is cleared.	R
b2	RX_FULL	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.	R

Table 3.13 IC_RAW_INTR_STAT Register Contents (3/3)

Bit Position	Bit Name	Function	R/W
b1	RX_OVER	Set if the receive buffer is completely filled to its depth of 8 and an additional byte is received from an external I2C device. The I2C controller 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_ENABLE_STATUS.IC_EN goes to 0, this interrupt is cleared. ***Note*)** If IC_CON[9] bit (RX_FIFO_FULL_HLD_CTRL) is programmed to HIGH, then the RX_OVER interrupt will never occur, because the Rx FIFO will never overflow.	R
b0	RX_UNDER	Set to 1b 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_ENABLE_STATUS.IC_EN goes to 0, this interrupt is cleared.	R

3.4.12 IC_RX_TL — I2C Receive FIFO Threshold Register

Address: 4006 3038h (I2C1)
4006 4038h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	RX_TL							
Value after reset	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

Table 3.14 IC_RX_TL Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	RX_TL	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.	R/W

3.4.13 IC_TX_TL — I2C Transmit FIFO Threshold Register

Address: 4006 303Ch (I2C1)
4006 403Ch (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	TX_TL							
Value after reset	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

Table 3.15 IC_TX_TL Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	TX_TL	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.	R/W

3.4.14 IC_CLR_INTR — Clear Combined and Individual Interrupt Register

Address: 4006 3040h (I2C1)
4006 4040h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_INTR
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.16 IC_CLR_INTR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_INTR	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.	R

3.4.15 IC_CLR_RX_UNDER — Clear RX_UNDER Interrupt Register

Address: 4006 3044h (I2C1)
4006 4044h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_RX_UNDER
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.17 IC_CLR_RX_UNDER Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_RX_UNDER	Read this register to clear the RX_UNDER interrupt (bit 0) of the IC_RAW_INTR_STAT register.	R

3.4.16 IC_CLR_RX_OVER — Clear RX_OVER Interrupt Register

Address: 4006 3048h (I2C1)
4006 4048h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_RX_OVER
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.18 IC_CLR_RX_OVER Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_RX_OVER	Read this register to clear the RX_OVER interrupt (bit 1) of the IC_RAW_INTR_STAT register.	R

3.4.17 IC_CLR_TX_OVER — Clear TX_OVER Interrupt Register

Address: 4006 304Ch (I2C1)
4006 404Ch (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_TX_OVER
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.19 IC_CLR_TX_OVER Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_TX_OVER	Read this register to clear the TX_OVER interrupt (bit 3) of the IC_RAW_INTR_STAT register.	R

3.4.18 IC_CLR_RD_REQ — Clear RD_REQ Interrupt Register

Address: 4006 3050h (I2C1)
4006 4050h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_RD_REQ
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.20 IC_CLR_RD_REQ Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_RD_REQ	Read this register to clear the RD_REQ interrupt (bit 5) of the IC_RAW_INTR_STAT register.	R

3.4.19 IC_CLR_TX_ABRT — Clear TX_ABRT Interrupt Register

Address: 4006 3054h (I2C1)
4006 4054h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_TX_ABRT
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.21 IC_CLR_TX_ABRT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_TX_ABRT	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.	R

3.4.20 IC_CLR_RX_DONE — Clear RX_DONE Interrupt Register

Address: 4006 3058h (I2C1)
4006 4058h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_RX_DONE
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.22 IC_CLR_RX_DONE Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_RX_DONE	Read this register to clear the RX_DONE interrupt (bit 7) of the IC_RAW_INTR_STAT register.	R

3.4.21 IC_CLR_ACTIVITY — Clear ACTIVITY Interrupt Register

Address: 4006 305Ch (I2C1)
4006 405Ch (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_ACTIVITY
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.23 IC_CLR_ACTIVITY Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_ACTIVITY	Reading this register clears the ACTIVITY interrupt if the I2C controller 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.	R

3.4.22 IC_CLR_STOP_DET — Clear STOP_DET Interrupt Register

Address: 4006 3060h (I2C1)
4006 4060h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_STOP_DET
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.24 IC_CLR_STOP_DET Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_STOP_DET	Read this register to clear the STOP_DET interrupt (bit 9) of the IC_RAW_INTR_STAT register.	R

3.4.23 IC_CLR_START_DET — Clear START_DET Interrupt Register

Address: 4006 3064h (I2C1)
4006 4064h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_START_DET
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.25 IC_CLR_START_DET Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_START_DET	Read this register to clear the START_DET interrupt (bit 10) of the IC_RAW_INTR_STAT register.	R

3.4.24 IC_CLR_GEN_CALL — Clear GEN_CALL Interrupt Register

Address: 4006 3068h (I2C1)
4006 4068h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_GEN_CALL
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.26 IC_CLR_GEN_CALL Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_GEN_CALL	Read this register to clear the GEN_CALL interrupt (bit 11) of the IC_RAW_INTR_STAT register.	R

3.4.25 IC_ENABLE — I2C Enable Register

Address: 4006 306Ch (I2C1)
4006 406Ch (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	ABORT	ENABLE
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0

Table 3.27 IC_ENABLE Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved		R
b1	ABORT	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.	R/W
b0	ENABLE	Controls whether the I2C controller is enabled. 0: Disables I2C controller (TX and RX FIFOs are held in an erased state) 1: Enables I2C controller Software can disable I2C controller while it is active. However, it is important that care be taken to ensure that I2C controller is disabled properly. When I2C controller 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 I2C controller 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 I2C controller stops the current transfer at the end of the current byte and does not acknowledge the transfer.	R/W

3.4.26 IC_STATUS — 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 controller is disabled by writing 0 in bit 0 of the IC_ENABLE register:

- Bits 1 and 2 are set to 1
- Bits 3 and 4 are set to 0

When the master or slave state machines goes to idle and IC_ENABLE_STATUS.IC_EN = 0:

- Bits 5 and 6 are set to 0

Address: 4006 3070h (I2C1)
4006 4070h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	SLV_ACTIVITY	MST_ACTIVITY	RFF	RFNE	TFE	TFNF	IC_STATUS_ACTIVITY
Value after reset	X	X	X	X	X	X	X	X	X	0	0	0	0	1	1	0

Table 3.28 IC_STATUS Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b7	Reserved		R
b6	SLV_ACTIVITY	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 I2C controller is not Active 1: Slave FSM is not in IDLE state so the Slave part of I2C controller is Active	R
b5	MST_ACTIVITY	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 I2C controller is not Active 1: Master FSM is not in IDLE state so the Master part of I2C controller is Active ***Note*)** IC_STATUS[0] - that is, IC_STATUS_ACTIVITY bit - is the OR of SLV_ACTIVITY and MST_ACTIVITY bits.	R
b4	RFF	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	R
b3	RFNE	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	R

Table 3.28 IC_STATUS Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b2	TFE	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	R
b1	TFNF	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	R
b0	IC_STATUS_ACTIVITY	I2C Activity Status. 0: I2C controller is not Active 1: I2C controller is Active	R

3.4.27 IC_TXFLR — 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 controller 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.

Address: 4006 3074h (I2C1)
4006 4074h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	TXFLR			
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0

Table 3.29 IC_TXFLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b4	Reserved		R
b3 to b0	TXFLR	Transmit FIFO Level. Contains the number of valid data entries in the transmit FIFO.	R

3.4.28 IC_RXFLR — 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 controller 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.

Address: 4006 3078h (I2C1)
4006 4078h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	RXFLR			
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0

Table 3.30 IC_RXFLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b4	Reserved		R
b3 to b0	RXFLR	Receive FIFO Level. Contains the number of valid data entries in the receive FIFO.	R

3.4.29 IC_SDA_HOLD — 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 I2C_SCLK 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 I2C_SCLK cycles.

Address: 4006 307Ch (I2C1)
4006 407Ch (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	IC_SDA_RX_HOLD							
Value after reset	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	IC_SDA_TX_HOLD															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 3.31 IC_SDA_HOLD Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b24	Reserved		R
b23 to b16	IC_SDA_RX_HOLD	Sets the required SDA hold time in units of I2C_SCLK period, when I2C controller acts as a receiver.	R/W
b15 to b0	IC_SDA_TX_HOLD	Sets the required SDA hold time in units of I2C_SCLK period, when I2C controller acts as a transmitter.	R/W

3.4.30 IC_TX_ABRT_SOURCE — 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.

Address: 4006 3080h (I2C1)
4006 4080h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	TX_FLUSH_CNT									—	—	—	—	—	—	ABRT_USER_ABRT
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	ABRT_SLVRD_INTX	ABRT_SLV_ARBLOST	ABRT_SLVFLUSH_TXFIFO	ARB_LOST	ABRT_MASTER_DIS	ABRT_10B_RD_NORSTRT	ABRT_SBYTE_NORSTRT	—	ABRT_SBYTE_ACKDET	—	ABRT_GCALL_READ	ABRT_GCALL_NOACK	ABRT_TXDATA_NOACK	ABRT_10ADDR2_NOACK	ABRT_10ADDR1_NOACK	ABRT_7B_ADDR_NOACK
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.32 IC_TX_ABRT_SOURCE Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b23	TX_FLUSH_CNT	This field indicates the number of Tx FIFO Data Commands which are flushed due to TX_ABRT interrupt. It is cleared whenever I2C controller is disabled. Role of I2C controller: Master-Transmitter or Slave-Transmitter	R
b22 to b17	Reserved		R
b16	ABRT_USER_ABRT	This is a master-mode-only bit. 1: Master has detected the transfer abort (IC_ENABLE[1]). Role of I2C controller: Master-Transmitter	R
b15	ABRT_SLVRD_INTX	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 I2C controller: Slave-Transmitter	R
b14	ABRT_SLV_ARBLOST	1: Slave 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 I2C controller no longer own the bus. Role of I2C controller: Slave-Transmitter	R
b13	ABRT_SLVFLUSH_TXFIFO	1: 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 I2C controller: Slave-Transmitter	R
b12	ARB_LOST	1: Master has lost arbitration, or if IC_TX_ABRT_SOURCE[14] is also set, then the slave transmitter has lost arbitration. Role of I2C controller: Master-Transmitter or Slave-Transmitter	R
b11	ABRT_MASTER_DIS	1: User tries to initiate a Master operation with the Master mode disabled. Role of I2C controller: Master-Transmitter or Master-Receiver	R

Table 3.32 IC_TX_ABRT_SOURCE Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b10	ABRT_10B_RD_NORSTRT	1: 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 I2C controller: Master-Receiver	R
b9	ABRT_SBYTE_NORSTRT	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. 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 I2C controller: Master	R
b8	Reserved		R
b7	ABRT_SBYTE_ACKDET	1: Master has sent a START Byte and the START Byte was acknowledged (wrong behavior). Role of I2C controller: Master	R
b6	Reserved		R
b5	ABRT_GCALL_READ	1: I2C controller in master mode sent a General Call but the user programmed the byte following the General Call to be a read from the bus. Role of I2C controller: Master-Transmitter	R
b4	ABRT_GCALL_NOACK	1: I2C controller in master mode sent a General Call and no slave on the bus acknowledged the General Call. Role of I2C controller: Master-Transmitter	R
b3	ABRT_TXDATA_NOACK	1: This is a master-mode only bit. Master has received an acknowledgement for the address, but when it sent data byte(s) following the address, it did not receive an acknowledge from the remote slave(s). Role of I2C controller: Master-Transmitter	R
b2	ABRT_10ADDR2_NOACK	1: Master is in 10-bit address mode and the second address byte of the 10-bit address was not acknowledged by any slave. Role of I2C controller: Master-Transmitter or Master-Receiver	R
b1	ABRT_10ADDR1_NOACK	1: Master is in 10-bit address mode and the first 10-bit address byte was not acknowledged by any slave. Role of I2C controller: Master-Transmitter or Master-Receiver	R
b0	ABRT_7B_ADDR_NOACK	1: Master is in 7-bit addressing mode and the address sent was not acknowledged by any slave. Role of I2C controller: Master-Transmitter or Master-Receiver	R

3.4.31 IC_SLV_DATA_NACK_ONLY — Generate Slave Data NACK Register

The register is used to generate a NACK for the data part of a transfer when I2C controller is acting as a slave-receiver.

A write can occur on this register if both of the following conditions are met:

- I2C controller is disabled (IC_ENABLE[0] = 0)
- Slave part is inactive (IC_STATUS[6] = 0)

Address: 4006 3084h (I2C1)
4006 4084h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	NACK
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.33 IC_SLV_DATA_NACK_ONLY Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	NACK	Generate NACK. This NACK generation only occurs when I2C controller 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	R/W

3.4.32 IC_SDA_SETUP — I2C SDA Setup Register

This register controls the amount of time delay (in terms of number of I2C_SCLK periods) introduced in the rising edge of SCL, relative to SDA changing, when I2C controller services a read request in a slave-transmitter operation. The relevant I2C requirement is $t_{SU;DAT}$ 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.

Address: 4006 3094h (I2C1)
4006 4094h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	SDA_SETUP							
Value after reset	X	X	X	X	X	X	X	X	0	1	1	0	0	1	0	0

Table 3.34 IC_SDA_SETUP Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	SDA_SETUP	SDA Setup. It is recommended that if the required delay is 1000 ns, then for an I2C_SCLK 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.	R/W

3.4.33 IC_ACK_GENERAL_CALL — I2C ACK General Call Register

The register controls whether I2C controller responds with an ACK or NACK when it receives an I2C General Call address.

NOTE

This register is applicable only when the I2C controller is in slave mode.

Address: 4006 3098h (I2C1)
4006 4098h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	ACK_GEN_CALL
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 3.35 IC_ACK_GENERAL_CALL Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	ACK_GEN_CALL	ACK General Call When set to 1, I2C controller responds with a ACK when it receives a General Call. Otherwise, I2C controller responds with a NACK.	R/W

3.4.34 IC_ENABLE_STATUS — I2C Enable Status Register

The register is used to report the I2C controller hardware status when the IC_ENABLE[0] register is set from 1 to 0; that is, when I2C controller 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 written with “0”, a delay occurs for bit 0 to be read as “0” because disabling the I2C controller depends on I2C bus activities.

Address: 4006 309Ch (I2C1)
4006 409Ch (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	SLV_RX_DATA_LOST	SLV_DISABLED_WHILE_BUSY	IC_EN
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0

Table 3.36 IC_ENABLE_STATUS Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b3	Reserved		R
b2	SLV_RX_DATA_LOST	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, the I2C controller 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 I2C controller 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, the I2C controller is deemed to have been disabled without being actively involved in the data phase of a Slave-Receiver transfer. • The CPU can safely read this bit when IC_EN (bit 0) is read as 0.	R

Table 3.36 IC_ENABLE_STATUS Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b1	SLV_DISABLED_WHILE_BUSY	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) I2C controller 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, I2C controller 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 I2C controller (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 I2C controller 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, I2C controller is deemed to have been disabled when there is master activity, or when the I2C bus is idle. • The CPU can safely read this bit when IC_EN (bit 0) is read as 0.	R
b0	IC_EN	I2C interface Status. When read as 1, I2C controller is deemed to be in an enabled state. When read as 0, I2C controller 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).	R

3.4.35 IC_FS_SPKLEN — I2C Sm, Fm Spike Suppression Limit

This register is used to store the duration, measured in I2C_SCLK cycles, of the longest spike that is filtered out by the spike suppression logic when the component is operating in Standard mode or Fast mode. The relevant I2C requirement is t_{SP} as detailed in the I2C Bus Specification. This register must be programmed with a minimum value of 1.

Address: 4006 30A0h (I2C1)
4006 40A0h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	IC_FS_SPKLEN							
Value after reset	X	X	X	X	X	X	X	X	0	0	0	0	0	1	0	1

Table 3.37 IC_FS_SPKLEN Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved		R
b7 to b0	IC_FS_SPKLEN	This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in I2C_SCLK 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.	R/W

3.4.36 IC_CLR_RESTART_DET — Clear RESTART_DET Interrupt Register

Address: 4006 30A8h (I2C1)
4006 40A8h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	CLR_RESTART_DET
Value after reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0

Table 3.38 IC_CLR_RESTART_DET Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved		R
b0	CLR_RESTART_DET	Read this register to clear the RESTART_DET interrupt (bit 12) of IC_RAW_INTR_STAT register.	R

3.4.37 IC_COMP_PARAM_1 — Component Parameter Register 1

This is a constant read-only register that contains encoded information about the I2C controller parameter settings.

Address: 4006 30F4h (I2C1)
4006 40F4h (I2C2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	TX_BUFFER_DEPTH							
Value after reset	X	X	X	X	X	X	X	X	0	0	0	0	0	1	1	1

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	RX_BUFFER_DEPTH								ADD_ENCODED_PARAMS	HAS_DMA	INTR_IO	HC_COUNT_VALUES	MAX_SPEED_MODE		APB_DATA_WIDTH	
Value after reset	0	0	0	0	0	1	1	1	1	0	1	0	1	0	1	0

Table 3.39 IC_COMP_PARAM_1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b24	Reserved		R
b23 to b16	TX_BUFFER_DEPTH	The value of this register is derived from the depth of transmit buffer. 0x7: The buffer depth is 8.	R
b15 to b8	RX_BUFFER_DEPTH	The value of this register is derived from the depth of receive buffer. 0x7: The buffer depth is 8.	R
b7	ADD_ENCODED_PARAMS	The value of this register shows if encoded information has been added. 1: The capability of reading these encoded parameters via software has been included.	R
b6	HAS_DMA	The value of this register shows if the DMA handshaking interface signals is available. 0: Not available	R
b5	INTR_IO	The value of this register shows if the interrupts are individual or combined to a single one. 1: Combined	R
b4	HC_COUNT_VALUES	The value of this register shows if the *CNT registers are writable or read-only. 0: Writable	R
b3, b2	MAX_SPEED_MODE	The value of this register shows the maximum I2C mode supported by the I2C controller. 2: Fast mode	R
b1, b0	APB_DATA_WIDTH	The value of this register shows the width of the APB data bus to which the I2C controller is attached. 2: 32 bits	R

3.5 Operation Modes

It is important to note that the I2C controller should only be set to operate as an I2C Master, or I2C Slave, but not both simultaneously. This is achieved by ensuring that bit 6 (IC_SLAVE_DISABLE) and 0 (MASTER_MODE) of the IC_CON register are never set to 0 and 1, respectively.

3.5.1 Slave Mode Operation

3.5.1.1 Initial Configuration

To use the I2C controller as a slave, perform the following steps:

(1) Disable the I2C controller 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 I2C controller 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 I2C controller 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 I2C controller by writing a “1” in bit 0 of the IC_ENABLE register.

CAUTION

- It is recommended that the 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 I2C Slave.
- This scenario can also be avoided by configuring the I2C controller with IC_SLAVE_DISABLE = 1 and 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 I2C_SCLK cycles after reset de-assertion.

3.5.1.2 Slave Transmitter Operation for a Single Byte

When another I2C master device on the bus addresses the I2C controller and requests data, the I2C controller acts as a slave transmitter and the following steps occur:

(5) The other I2C master device initiates an I2C transfer with an address that matches the slave address in the IC_SAR register of the I2C controller.

(6) The I2C controller acknowledges the sent address and recognizes the direction of the transfer to indicate that it is acting as a slave transmitter.

(7) The I2C controller 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.

- 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.
- Software must then act to satisfy the I2C transfer.
- The timing interval used should be in the order of 10 times the fastest SCL clock period the I2C controller can handle. For example, for 400 kb/s, the timing interval is 25 µs.

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 I2C bus.

(8) If there is any data remaining in the Tx FIFO before receiving the read request, then the I2C controller 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 Tx FIFO is forced into a flushed/reset state whenever a TX_ABRT event occurs, it is necessary for software to release the I2C controller from this state by reading the IC_CLR_TX_ABRT register before attempting to write into the Tx FIFO. See IC_RAW_INTR_STAT register 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.

- 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.
- There is no further action required from software.
- The timing interval used should be similar to that described in the previous step for the IC_RAW_INTR_STAT[5] register.

(9) Software writes to the IC_DATA_CMD register with the data to be written (by writing a “0” in bit 8).

(10) Software must clear the RD_REQ and TX_ABRT interrupts (bits 5 and 6, respectively) of the IC_RAW_INTR_STAT register.

(11) The I2C controller releases the SCL and transmits the byte.

(12) The master may hold the I2C bus by issuing a RESTART condition or release the bus by issuing a STOP condition.

3.5.1.3 Slave Receiver Operation for a Single Byte

When another I2C master device on the bus addresses the I2C controller and is sending data, the I2C controller acts as a slave receiver and the following steps occur:

(1) The other I2C master device initiates an I2C transfer with an address that matches the I2C slave address in the IC_SAR register.

(2) The I2C controller acknowledges the sent address and recognizes the direction of the transfer to indicate that the I2C controller is acting as a slave receiver.

(3) I2C controller receives the transmitted byte and places it in the receive buffer.

(4) I2C controller 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_RX_TL to a value larger than 0, then it is recommended that a timing routine (similarly to “Slave Transmitter Operation for a Single Byte”) 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 I2C bus by issuing a RESTART condition, or release the bus by issuing a STOP condition.

3.5.1.4 Slave Transfer Operation for Bulk Transfer

In the standard I2C 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. I2C controller 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 I2C controller 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 I2C controller holds the I2C 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_MASK 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 (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”.

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, 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 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 I2C controller 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 I2C controller 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.

3.5.2 Master Mode Operation

3.5.2.1 Initial Configuration

To use the I2C controller as a master perform the following steps:

(1) Disable the I2C controller 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).

(3) Write to the IC_TAR register the address of the I2C device to be addressed. It also indicates whether a General Call or a START BYTE command is going to be performed by I2C controller. 7-bit or 10-bit addressing is controlled by the IC_10BITADDR_MASTER bit field (bit 12).

(4) Enable the I2C controller by writing a 1 to bit 0 of the IC_ENABLE register.

(5) 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 I2C controller is enabled, the data and commands are lost as the buffers are kept cleared when I2C controller is not enabled.

3.5.2.2 Dynamic IC_TAR or IC_10BITADDR_MASTER Update

The I2C controller supports dynamic updating of the IC_TAR (bits 9:0) and IC_10BITADDR_MASTER (bit 12) bit fields of the IC_TAR register. 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:

- I2C controller is not enabled (IC_ENABLE[0] = 0);
  OR
  I2C controller is enabled (IC_ENABLE[0] = 1); AND
  I2C controller is not engaged in any Master (tx, rx) operation (IC_STATUS[5] = 0); AND
  I2C controller 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.

- I2C controller is enabled (IC_ENABLE[0]=1); AND
  I2C controller 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

I2C controller 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.

3.5.2.3 Master Transmit and Master Receive

The I2C controller 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 I2C Rx/Tx Data Buffer and Command Register (IC_DATA_CMD). The CMD bit [8] should be written to 0 for I2C 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 I2C controller master continues to initiate transfers as long as there are commands present in the transmit FIFO. If the transmit FIFO becomes empty the master 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.

3.5.3 Disabling the I2C controller

The IC_ENABLE_STATUS register is used to allow software to unambiguously determine when the I2C controller has been completely disabled in response to bit 0 of the IC_ENABLE register being set from 1 to 0.

NOTE

The 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 I2C Master while processing a command without the STOP bit set, the I2C Master continues to remain active, holding the SCL line low until a new command is received in the Tx FIFO.

3.5.3.1 Procedure

(1) Define a timer interval (ti2c_poll) equal to the 10 times the signaling period for the highest I2C transfer speed used in the system and supported by I2C controller. For example, if the highest I2C transfer mode is 400 kb/s, then this ti2c_poll is 25 µs.

(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 I2C master transactions to be started by software, but allows any pending transfers to be completed.
This step can be ignored if I2C controller is programmed to operate as an I2C 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 ti2c_poll and proceed to the previous step. Otherwise, exit with a relevant success code.

3.5.4 Aborting the I2C Transfer

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.

3.5.4.1 Procedure

(1) Stop filling the Tx FIFO (IC_DATA_CMD) with new commands.

(2) Set bit 1 of the IC_ENABLE register (ABORT) to 1.

(3) Wait for the TX_ABRT interrupt.

(4) Read the IC_TX_ABRT_SOURCE register to identify the source as ABRT_USER_ABRT.

3.6 Programming the I2C Controller

3.6.1 Spike Suppression

The I2C controller contains programmable spike suppression logic that match requirements imposed by the I2C Bus Specification for Standard mode/Fast mode (t_{SP}).

This logic is based on counters that monitor the input signals (SCL and SDA), checking if they remain stable for a predetermined amount of I2C_SCLK cycles before they are sampled internally. There is one separate counter for each signal (SCL and SDA). The number of I2C_SCLK cycles can be programmed by the user and should be calculated taking into account the frequency of I2C_SCLK 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 I2C Bus Specification calls a maximum spike length of 50ns for the Standard mode and Fast mode:

IC_FS_SPKLEN register holds the maximum spike length for Standard mode and Fast mode.

The register is 8 bits width and accessible through the APB interface for read and write purposes; however, it can be written to only when the I2C controller is disabled. The minimum value that can be programmed into the register is 1; attempting to program a value smaller than 1 results in the value 1 being written.

As an example, for a 10 ns I2C_SCLK period the count limit value to be used is 5 (50 ns spike suppression). Because the minimum value that can be programmed into the IC_FS_SPKLEN registers is 1, the spike length specification can be exceeded by a long pulse of I2C_SCLK. Consider the simple example of a 10 MHz (100 ns period) I2C_SCLK; in this case, the minimum spike length that can be programmed is 100 ns, which means that spikes up to this length are suppressed.

3.6.2 I2C_SCLK Frequency Configuration

When the I2C controller is configured as a master, the *CNT registers must be set before any I2C 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

NOTE

It is not necessary to program any of the *CNT registers if the I2C controller is enabled to operate only as an I2C slave, since these registers are used only to determine the SCL timing requirements for operation as an I2C master.

3.6.2.1 Minimum High and Low Counts

When the I2C controller 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.

Details regarding the I2C controller high and low counts are as follows:

- The minimum value of IC_FS_SPKLEN + 7 for the *_LCNT registers is due to the time required for the I2C controller to drive SDA after a negative edge of SCL.
- The minimum value of IC_FS_SPKLEN + 5 for the *_HCNT registers is due to the time required for the I2C controller to sample SDA during the high period of SCL.
- The I2C controller 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 I2C controller adds IC_FS_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 IC_FS_SPKLEN + 2 I2C_SCLK cycles.
  - This filtering includes metastability removal and the programmable spike suppression on SDA and SCL edges.
  - Whenever SCL is driven 1 to 0 by the I2C controller -- that is, completing the SCL high time -- an internal logic latency of three I2C_SCLK cycles is incurred. Consequently, the minimum SCL low time of which the I2C controller is capable is nine (9) I2C_SCLK periods (7 + 1 + 1), while the minimum SCL high time is thirteen (13) I2C_SCLK periods (6 + 1 + 3 + 3).

NOTE

The total high time and low time of SCL generated by the I2C controller master is also influenced by the rise time and fall time of the SCL line, as shown in the illustration and equations in Figure below. 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 I2C controller.

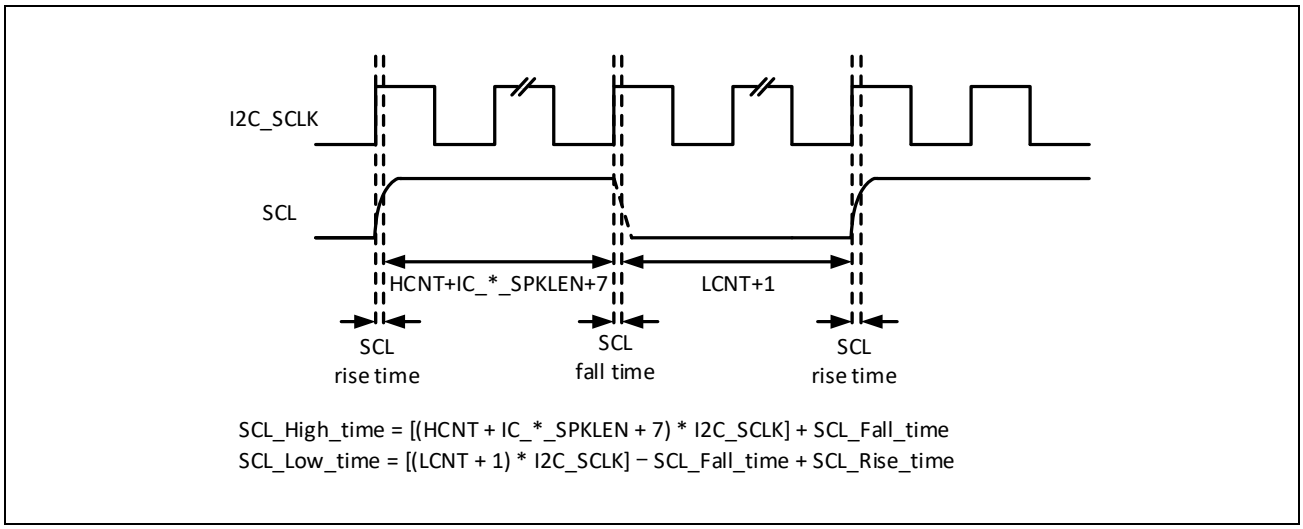

Figure 3.2 SCL High and Low Time

The table below lists the minimum I2C_SCLK values for all modes with high and low count values. SCL_Rise_time and SCL_Fall_time are not considered.

Table 3.40 Minimum High and Low Counts

Speed Modes	Minimum I2C_SCLK Freq (MHz)	Minimum IC_FS_SPKLEN	IC_*_SCL_LCNT	SCL Low Time (µs)	IC_*_SCL_HCNT	SCL High Time (µs)
Standard mode	2.7	1	12	4.7	6	5.2
Fast mode	12	1	15	1.33	6	1.16

Note: I2C_SCLK must be faster than or equal to I2C_PCLK.

3.6.3 SDA Hold Time

The I2C protocol specification requires 300 ns of hold time on the SDA signal ($t_{HD;DAT}$) in Standard mode and Fast mode.

Board delays on the SCL and SDA signals can mean that the hold time requirement is met at the I2C master, but not at the I2C slave (or vice versa). As each application encounters differing board delays, the I2C controller 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).

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 I2C controller is disabled (IC_ENABLE[0] = 0).

3.6.3.1 SDA Hold Timings in Receiver

When I2C controller acts as a receiver, according to the I2C 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 I2C controller applies to the incoming SDA line. Each value in the IC_SDA_RX_HOLD register represents a unit of one I2C_SCLK 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.

The Figure below shows the I2C protocol as receiver with IC_SDA_RX_HOLD programmed to greater than or equal to 3. If IC_SDA_RX_HOLD is greater than 3, I2C controller does not hold SDA beyond 3 I2C_SCLK cycles, because SCL goes LOW internally.

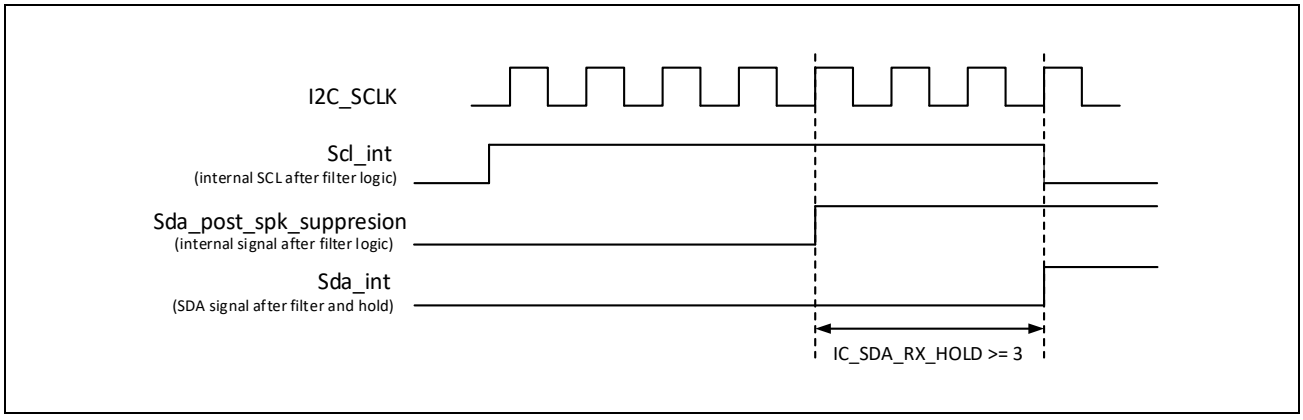

Figure 3.3 SDA Hold Timings in the Receiver

The maximum values of IC_SDA_RX_HOLD that can be programmed in the register for the respective speed modes are derived from the following equations:

Standard mode

Maximum IC_SDA_RX_HOLD = IC_SS_SCL_HCNT − IC_FS_SPKLEN − 3

Fast mode

Maximum IC_SDA_RX_HOLD = IC_FS_SCL_HCNT − IC_FS_SPKLEN − 3

The above maximum value is applicable in Master mode. In Slave mode, make sure the IC_SDA_RX_HOLD does not exceed the maximum SCL fall time (t_f in Standard mode and Fast mode).

3.6.3.2 SDA Hold Timings in Transmitter

The IC_SDA_TX_HOLD register can be used to alter the timing of the generated SDA signal by the I2C controller. Each value in the IC_SDA_TX_HOLD register represents a unit of one I2C_SCLK period.

When the I2C controller is operating in Master Mode, the minimum $t_{HD;DAT}$ timing is one I2C_SCLK period. Therefore, even when IC_SDA_TX_HOLD has a value of zero, the I2C controller will drive SDA one I2C_SCLK cycle after driving SCL to logic 0. For all other values of IC_SDA_TX_HOLD, the following is true:

- Drive on SDA occurs IC_SDA_TX_HOLD I2C_SCLK cycles after driving SCL to logic 0

When the I2C controller is operating in Slave Mode, the minimum $t_{HD;DAT}$ timing is IC_FS_SPKLEN + 7 I2C_SCLK periods. This delay allows for synchronization and spike suppression on the SCL sample. Therefore, even when IC_SDA_TX_HOLD has a value less than IC_FS_SPKLEN + 7, the I2C controller drives SDA IC_FS_SPKLEN + 7 I2C_SCLK cycles after SCL has transitioned to logic 0. For all other values of IC_SDA_TX_HOLD, the following is true:

- Drive on SDA occurs IC_SDA_TX_HOLD I2C_SCLK cycles after SCL has transitioned to logic 0.

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 I2C_SCLK cycles.

Section 4 Basic GPIO

4.1 Overview

RZ/N1 provides 3 Basic GPIO (BGPIO) modules.

The BGPIO supports the following features:

- Up to six ports which are separately configurable:
  - BGPIO1 port A, BGPIO1 port B, BGPIO2 port A, BGPIO2 port B, BGPIO3 port A, BGPIO3 port B
- Separate data registers and data direction registers for each signal
- Independently controllable signal bits
- Configurable interrupt mode for BGPIO1 port A, BGPIO2 port A and BGPIO3 port A only (32 interrupts by port)
  - This can be programmed to accept external signals as interrupt sources on any of the bits of the signal
  - The type of interrupt is programmable with one of the following settings:
    a. Active high and level
    b. Active low and level
    c. Rising edge
    d. Falling edge
- All interrupts from BGPIO1, 2, 3 port A are routed by a programmable wrapper to extract 8 interrupts directly routed to Cortex®-A7 and Cortex®-M3
- Trigger synchronous operation with dedicated wrapper circuit allowing real time operation controlled by interrupts. The status of the GPIO signal is updated in synchronization with an interrupt from an on-chip peripheral function. The trigger synchronous control mode can be enable/disable via register setting.

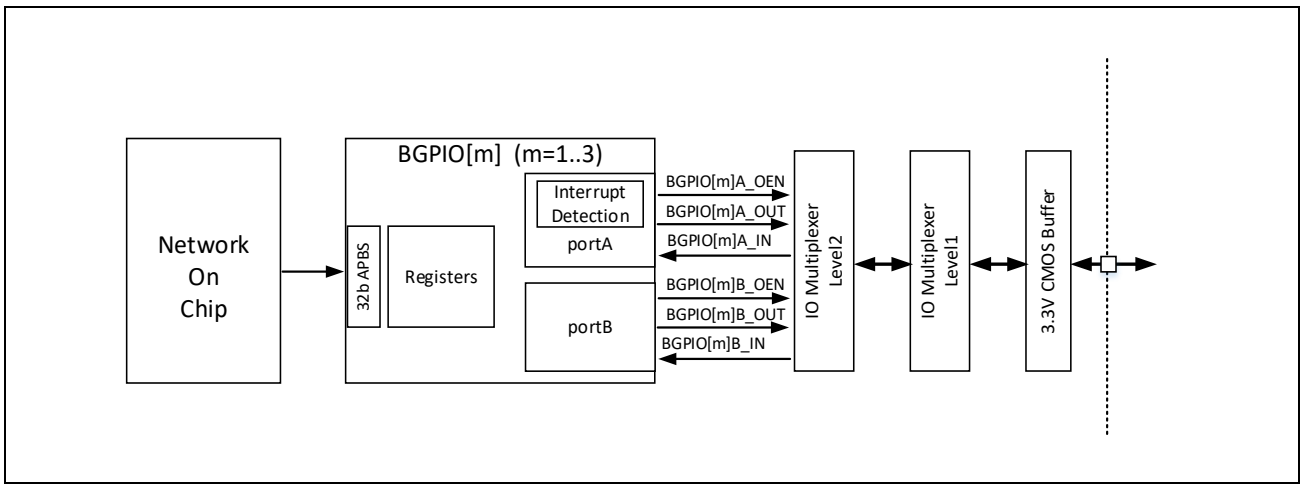

Figure 4.1 BGPIO Summary Synoptic

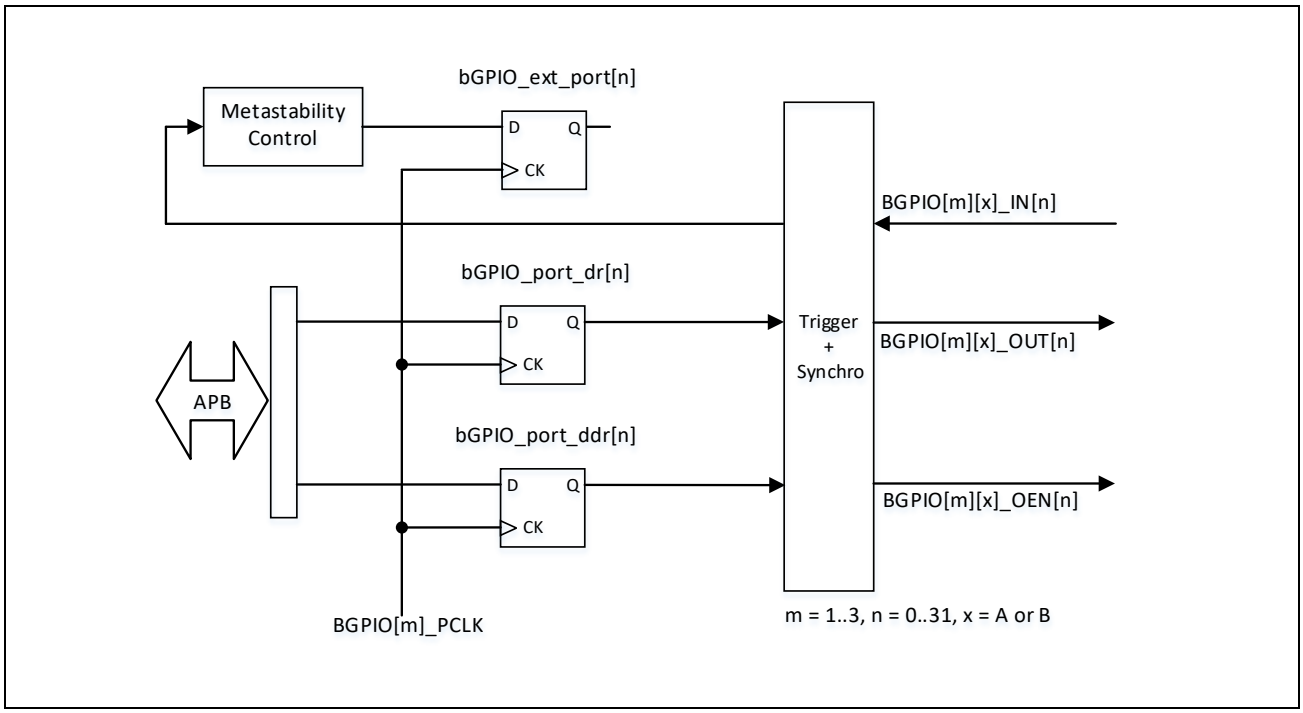

Figure 4.2 BGPIO Synoptic (Port Management)

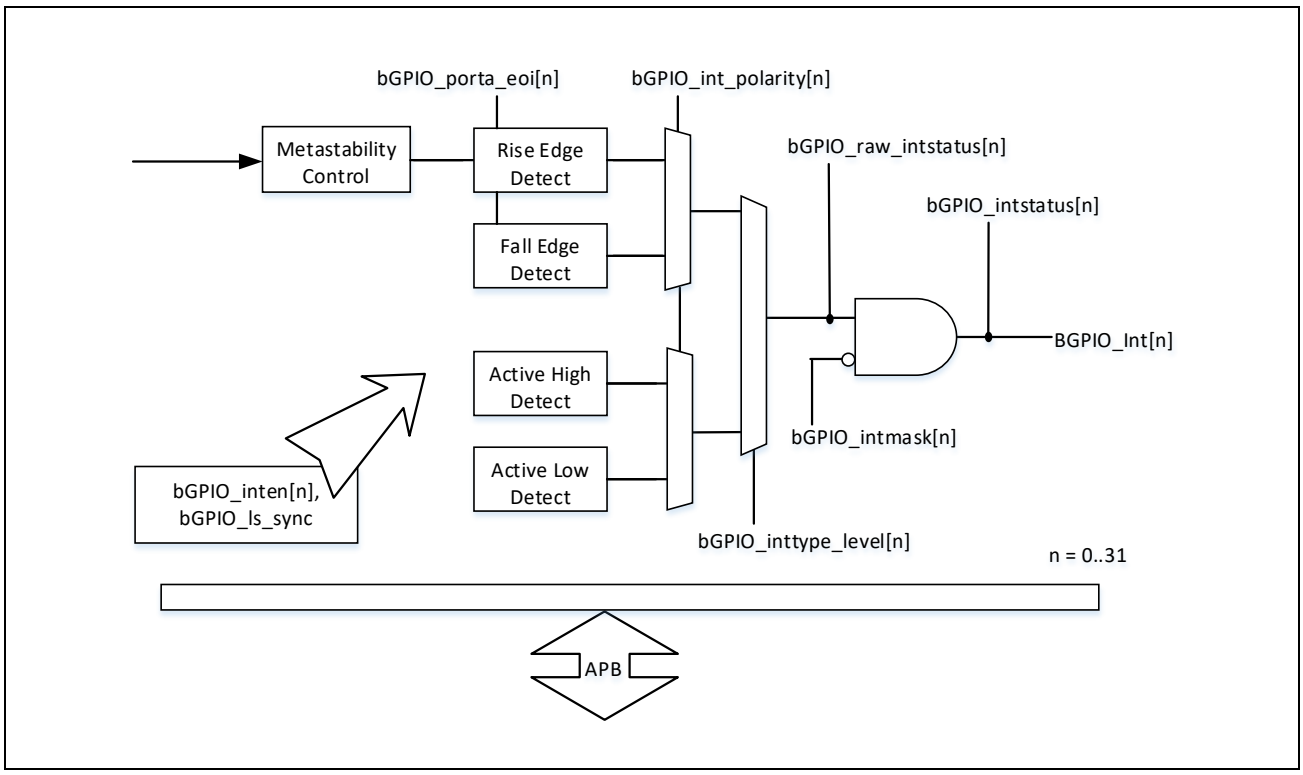

Figure 4.3 BGPIO Synoptic (Interrupt Management)

4.2 Signal Interfaces

Table 4.1 BGPIO Signal Interface

Signal Name	Input Output	Description
Clock		
BGPIO[m]_PCLK	Input	Internal bus clock (APB)
Interrupt		
BGPIO[m]_Int[31:0]	Output	Level sensitive interrupt output of port A, Active High
External GPIO Pin Interface Signal		
BGPIO1A_OEN[31:0] BGPIO1B_OEN[31:0] BGPIO2A_OEN[31:0] BGPIO2B_OEN[31:0] BGPIO3A_OEN[31:0] BGPIO3B_OEN[9:0]	Output	GPIO output enable
BGPIO1A_IN[31:0] BGPIO1B_IN[31:0] BGPIO2A_IN[31:0] BGPIO2B_IN[31:0] BGPIO3A_IN[31:0] BGPIO3B_IN[9:0]	Input	GPIO input
BGPIO1A_OUT[31:0] BGPIO1B_OUT[31:0] BGPIO2A_OUT[31:0] BGPIO2B_OUT[31:0] BGPIO3A_OUT[31:0] BGPIO3B_OUT[9:0]	Output	GPIO output
GPIO Trigger Control Signal		
CFG_GPIOT_PTEN1A[31:0] CFG_GPIOT_PTEN1B[31:0] CFG_GPIOT_PTEN2A[31:0] CFG_GPIOT_PTEN2B[31:0] CFG_GPIOT_PTEN3A[31:0] CFG_GPIOT_PTEN3B[9:0]	Input	GPIO trigger Enable
GPIO_TRIGGER[3:0]	Input	GPIO trigger signal

Note: m = 1..3
Output enable, input, output are routed by IO Multiplexing logic.

4.3 Register Map

4.3.1 Register Map BGPIO1

Table 4.2 Basic GPIO1 Register Map

Address	Register Symbol	Register Name
5000 B000h	rGPIO_swporta_dr	GPIO Port A Data Output Register
5000 B004h	rGPIO_swporta_ddr	GPIO Port A Data Direction Register
5000 B00Ch	rGPIO_swportb_dr	GPIO Port B Data Output Register
5000 B010h	rGPIO_swportb_ddr	GPIO Port B Data Direction Register
5000 B030h	rGPIO_inten	GPIO Port A Interrupt Enable Register
5000 B034h	rGPIO_intmask	GPIO Port A Interrupt Mask Register
5000 B038h	rGPIO_inttype_level	GPIO Port A Interrupt Level Register
5000 B03Ch	rGPIO_int_polarity	GPIO Port A Interrupt Polarity Register
5000 B040h	rGPIO_intstatus	GPIO Port A Interrupt Status
5000 B044h	rGPIO_raw_intstatus	GPIO Port A Raw Interrupt Status (Premasking)
5000 B04Ch	rGPIO_porta_eoi	GPIO Port A Clear Interrupt Register
5000 B050h	rGPIO_ext_porta	GPIO Port A Data Input Register
5000 B054h	rGPIO_ext_portb	GPIO Port B Data Input Register
5000 B060h	rGPIO_ls_sync	GPIO Port A Level-Sensitive Synchronization Enable Register

4.3.2 Register Map BGPIO2

Table 4.3 Basic GPIO2 Register Map

Address	Register Symbol	Register Name
5000 C000h	rGPIO_swporta_dr	GPIO Port A Data Output Register
5000 C004h	rGPIO_swporta_ddr	GPIO Port A Data Direction Register
5000 C00Ch	rGPIO_swportb_dr	GPIO Port B Data Output Register
5000 C010h	rGPIO_swportb_ddr	GPIO Port B Data Direction Register
5000 C030h	rGPIO_inten	GPIO Port A Interrupt Enable Register
5000 C034h	rGPIO_intmask	GPIO Port A Interrupt Mask Register
5000 C038h	rGPIO_inttype_level	GPIO Port A Interrupt Level Register
5000 C03Ch	rGPIO_int_polarity	GPIO Port A Interrupt Polarity Register
5000 C040h	rGPIO_intstatus	GPIO Port A Interrupt Status
5000 C044h	rGPIO_raw_intstatus	GPIO Port A Raw Interrupt Status (Premasking)
5000 C04Ch	rGPIO_porta_eoi	GPIO Port A Clear Interrupt Register
5000 C050h	rGPIO_ext_porta	GPIO Port A Data Input Register
5000 C054h	rGPIO_ext_portb	GPIO Port B Data Input Register
5000 C060h	rGPIO_ls_sync	GPIO Port A Level-Sensitive Synchronization Enable Register

4.3.3 Register Map BGPIO3

Table 4.4 Basic GPIO3 Register Map

Address	Register Symbol	Register Name
5000 D000h	rGPIO_swporta_dr	GPIO Port A Data Output Register
5000 D004h	rGPIO_swporta_ddr	GPIO Port A Data Direction Register
5000 D00Ch	rGPIO_swportb_dr	GPIO Port B Data Output Register
5000 D010h	rGPIO_swportb_ddr	GPIO Port B Data Direction Register
5000 D030h	rGPIO_inten	GPIO Port A Interrupt Enable Register
5000 D034h	rGPIO_intmask	GPIO Port A Interrupt Mask Register
5000 D038h	rGPIO_inttype_level	GPIO Port A Interrupt Level Register
5000 D03Ch	rGPIO_int_polarity	GPIO Port A Interrupt Polarity Register
5000 D040h	rGPIO_intstatus	GPIO Port A Interrupt Status
5000 D044h	rGPIO_raw_intstatus	GPIO Port A Raw Interrupt Status (Premasking)
5000 D04Ch	rGPIO_porta_eoi	GPIO Port A Clear Interrupt Register
5000 D050h	rGPIO_ext_porta	GPIO Port A Data Input Register
5000 D054h	rGPIO_ext_portb	GPIO Port B Data Input Register
5000 D060h	rGPIO_ls_sync	GPIO Port A Level-Sensitive Synchronization Enable Register

4.4 Register Description

4.4.1 rGPIO_swporta_dr — GPIO Port A Data Output Register

Address: 5000 B000h (BGPIO1)
5000 C000h (BGPIO2)
5000 D000h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_port_dr															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_port_dr															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.5 rGPIO_swporta_dr Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_port_dr	For each n bit, with n = 0..31, values written to this register are output on the external pins BGPIO[m]A[n] if the corresponding data direction bits for port are set to output mode The value read back is equal to the last value written to this register.	R/W

4.4.2 rGPIO_swporta_ddr — GPIO Port A Data Direction Register

Address: 5000 B004h (BGPIO1)
5000 C004h (BGPIO2)
5000 D004h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_port_ddr															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_port_ddr															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.6 rGPIO_swporta_ddr Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_port_ddr	For each n bit, with n = 0..31, values written to this register independently control the direction of the corresponding data bit in external pins BGPIO[m]A[n] 1'b0: Input (default after reset) 1'b1: Output	R/W

4.4.3 rGPIO_swportb_dr — GPIO Port B Data Output Register

Address: 5000 B00Ch (BGPIO1)
5000 C00Ch (BGPIO2)
5000 D00Ch (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_port_dr															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_port_dr															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.7 rGPIO_swportb_dr Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_port_dr	For each n bit, with n = 0..31, values written to this register are output on the external pins BGPIO[m]B[n] if the corresponding data direction bits for port are set to output mode The value read back is equal to the last value written to this register.	R/W

4.4.4 rGPIO_swportb_ddr — GPIO Port B Data Direction Register

Address: 5000 B010h (BGPIO1)
5000 C010h (BGPIO2)
5000 D010h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_port_ddr															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_port_ddr															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.8 rGPIO_swportb_ddr Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_port_ddr	For each n bit, with n = 0..31, values written to this register independently control the direction of the corresponding data bit in external pins BGPIO[m]B[n] 1'b0: Input (default after reset) 1'b1: Output	R/W

4.4.5 rGPIO_inten — GPIO Port A Interrupt Enable Register

Address: 5000 B030h (BGPIO1)
5000 C030h (BGPIO2)
5000 D030h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_inten															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_inten															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.9 rGPIO_inten Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_inten	Allows each n bit of port A, with n = 0..31, to be configured for interrupts. By default, the generation of interrupts is disabled. Whenever a 1 is written to a bit of this register, it configures the corresponding bit on port A to become an interrupt, otherwise, the port A operates as a normal GPIO signal. Interrupts are disabled on the corresponding bits of port A if the corresponding data direction register is set to Output. 1'b0: Configure port A bit as normal GPIO signal (default) 1'b1: Configure port A bit as interrupt	R/W

4.4.6 rGPIO_intmask — GPIO Port A Interrupt Mask Register

Address: 5000 B034h (BGPIO1)
5000 C034h (BGPIO2)
5000 D034h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_intmask															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_intmask															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.10 rGPIO_intmask Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_intmask	For each n bit, with n=0..31, controls whether an interrupt on port A can create an interrupt for the interrupt controller by not masking it. By default, all interrupts bits are unmasked. Whenever a 1 is written to a bit in this register, it masks the interrupt generation capability for this signal otherwise interrupts are allowed through. The unmasked status can be read as well as the resultant status after masking. 1'b0: Interrupt bits are unmasked (default) 1'b1: Mask interrupt	R/W

4.4.7 rGPIO_inttype_level — GPIO Port A Interrupt Level Register

Address: 5000 B038h (BGPIO1)
5000 C038h (BGPIO2)
5000 D038h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_inttype_levell															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_inttype_levell															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.11 rGPIO_inttype_level Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_inttype_levell	For each n bit, with n = 0..31, controls the type of interrupt that can occur on port A. Whenever a 0 is written to a bit of this register, it configures the interrupt type to be level-sensitive, otherwise, it is edge-sensitive. 1'b0: Level-sensitive (default) 1'b1: Edge-sensitive	R/W

4.4.8 rGPIO_int_polarity — GPIO Port A Interrupt Polarity Register

Address: 5000 B03Ch (BGPIO1)
5000 C03Ch (BGPIO2)
5000 D03Ch (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_int_polarity															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_int_polarity															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.12 rGPIO_int_polarity Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_int_polarity	For each n bit, with n = 0..31, controls the polarity of edge or level sensitivity that can occur on input of port A. Whenever a 0 is written to a bit of this register, it configures the interrupt type to falling-edge or active-low sensitive, otherwise, it is rising-edge or active-high sensitive. 1'b0: Active-low (default) 1'b1: Active-high	R/W

4.4.9 rGPIO_intstatus — GPIO Port A Interrupt Status

Address: 5000 B040h (BGPIO1)
5000 C040h (BGPIO2)
5000 D040h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_intstatus															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_intstatus															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.13 rGPIO_intstatus Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_intstatus	For each n bit, with n = 0..31, interrupt status of port A. 1'b0: No Interrupt 1'b1: Interrupt generated	R

4.4.10 rGPIO_raw_intstatus — GPIO Port A Raw Interrupt Status (Premasking)

Address: 5000 B044h (BGPIO1)
5000 C044h (BGPIO2)
5000 D044h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_raw_intstatus															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_raw_intstatus															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.14 rGPIO_raw_intstatus Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_raw_intstatus	For each n bit, with n = 0..31, raw interrupt of status of port A (premasking bits). 1'b0: No interrupt requested 1'b1: Interrupt requested	R

4.4.11 rGPIO_porta_eoi — GPIO Port A Clear Interrupt Register

Address: 5000 B04Ch (BGPIO1)
5000 C04Ch (BGPIO2)
5000 D04Ch (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_porta_eoi															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_porta_eoi															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.15 rGPIO_porta_eoi Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_porta_eoi	For each n bit, with n = 0..31, controls the clearing of edge type interrupts from port A. When a 1 is written into a corresponding bit of this register, the interrupt is cleared. All interrupts are cleared when port A is not configured for interrupts. 1'b0: No interrupt clear (default) 1'b1: Clear interrupt	W

4.4.12 rGPIO_ext_porta — GPIO Port A Data Input Register

Address: 5000 B050h (BGPIO1)
5000 C050h (BGPIO2)
5000 D050h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_ext_port															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_ext_port															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.16 rGPIO_ext_porta Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_ext_port	For each n bit, with n = 0..31, when port is configured as Input, then reading this location reads the values on the external pins BGPIO[m]A[n] When the data direction of port is set as Output, reading this location reads the data output register for port except if the BGPIO is in trigger mode because of the flip-flops present on the path in that case.	R

4.4.13 rGPIO_ext_portb — GPIO Port B Data Input Register

Address: 5000 B054h (BGPIO1)
5000 C054h (BGPIO2)
5000 D054h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bGPIO_ext_port															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bGPIO_ext_port															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.17 rGPIO_ext_portb Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bGPIO_ext_port	For each n bit, with n = 0..31, when port is configured as Input, then reading this location reads the values on the external pins BGPIO[m]B[n] When the data direction of port is set as Output, reading this location reads the data output register for port except if the BGPIO is in trigger mode because of the flip-flops present on the path in that case.	R

4.4.14 rGPIO_ls_sync — GPIO Port A Level-Sensitive Synchronization Enable Register

Address: 5000 B060h (BGPIO1)
5000 C060h (BGPIO2)
5000 D060h (BGPIO3)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bGPIO_ls_sync
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.18 rGPIO_ls_sync Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Read as 0.	R
b0	bGPIO_ls_sync	Writing a 1 to this register results in all level-sensitive interrupts being synchronized to clock. 1'b0: No synchronization to BGPIO_PCLK (default) 1'b1: Synchronize to BGPIO_PCLK ***Caution*)** In order to prevent glitches on the interrupt lines to the interrupt controller, the firmware must configure this bit with 32'h1 value.	R/W

4.5 Operation

4.5.1 Main Functions Blocks Description

4.5.1.1 Data & Control Flow

The BGPIO controls the external output data and direction of external I/O. It also can read back the data on external inputs using memory mapped registers.

Each bit in each port A and port B is individually controllable. The data and control flow for a signal are shown in **Figure 4.2, BGPIO Synoptic (Port Management)**.

- Port direction control

  rGPIO_swporta_ddr.bGPIO_port_ddr[n] and rGPIO_swportb_ddr.bGPIO_port_ddr[n]

- Port output data

  rGPIO_swporta_dr.bGPIO_port_dr[n] and rGPIO_swportb_dr.bGPIO_port_dr[n]

- Port input data (read only)

  rGPIO_ext_porta.bGPIO_ext_port[n] and rGPIO_ext_portb.bGPIO_ext_port[n]

CAUTION

Valid BGPIO ports and registers depend on the device, RZ/N1D, N1S or N1L
Bit9 to 0 of the register related to the BGPIO3 port B are available only for RZ/N1D-400.

4.5.1.2 Interruption (Only Port A)

The port A can be programmed to accept external signals as interrupt sources on any of the bits of the signal. The type of interrupt is programmable with one of the following settings:

- Active high and level
- Active low and level
- Rising edge
- Falling edge

For each n bit of port A, with n = 0..31, the interrupts can be masked by programming the bGPIO_intmask[n] register. The interrupt status can be read before masking (called raw status) and after masking.

Whenever the port A is configured for interrupts, the data direction must be set to Input and the mode must be set to Software for interrupts to be latched. If the data direction register is reprogrammed to Output mode, then any pending interrupts are not lost. However, no new interrupts are generated.

Figure 4.3, BGPIO Synoptic (Interrupt Management) shows how the interrupts are generated and how the data flows.

For edge detected interrupts, the interrupt can be cleared by writing a 1 to the bGPIO_porta_eoi[n] bit for the corresponding bit to disable the interrupt. This write also clears the interrupt status and raw status registers. It is recommended that the interrupt source is cleared prior to writing to the bGPIO_porta_eoi[n] register. Writing to the bGPIO_porta_eoi[n] bit has no effect on level sensitive interrupts.

If level sensitive interrupts cause an interrupt, then software can poll the bGPIO_raw_intstatus[n] register until the interrupt source disappears, or it can write to the bGPIO_intmask[n] register to mask the interrupt.

If a software reads the bGPIO_intstatus[n] register to find multiple pending interrupt requests, then it is up to the processor to prioritize these pending interrupt requests.

There are no restrictions on the number of edge detected interrupts that can be cleared simultaneously by writing multiple 1's to the bGPIO_porta_eoi[n] register.

A case may arise where a software writes 1 in order to clear an existing interrupt during the same clock cycle in which a new interrupt is detected. In such a case, writing to the interrupt clear register clears only the first interrupt. The second interrupt is not lost, since setting an interrupt has a higher priority than clearing it.

4.5.1.3 Programmable Interrupts Routed on Cortex-A7 and M3

All interrupts from BGPIO1, 2, 3 port A are routed by GPIO interrupt multiplexer in system control to extract 8 interrupts GPIO_Int[7:0] directly routed to Cortex-A7 and M3. The multiplexer is set by rGPIOs_Level2_Gpio_Int_[n] of Config Sys2.

For each interrupt GPIO_Int[n] with n = 0..7:

- Selects an interrupt source from 3x32 possible interrupt sources (port A only)
  - BGPIO1_Int[31:0] or BGPIO2_Int[31:0] or BGPIO3_Int[31:0]

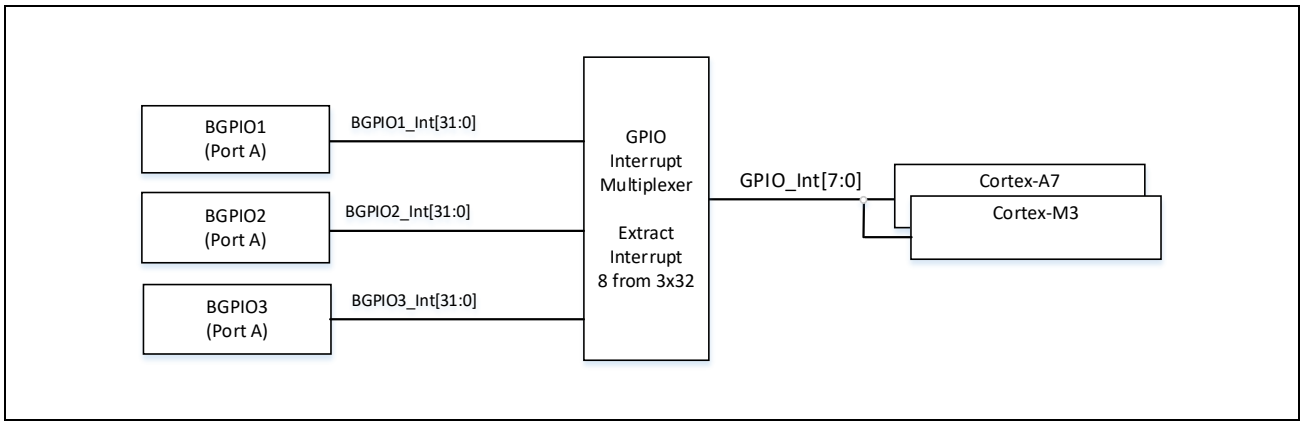

Figure 4.4 Programmable Interrupts Routed on Cortex-A7 and M3

4.5.1.4 Trigger Synchronous Operation

Trigger Synchronous operation managed with dedicated wrapper circuit allows real time operation by interrupt sources configured in system controller. The GPIO data signal (input, output) of the GPIO pin is updated in synchronization with an interrupt source from an on-chip peripheral function. The trigger synchronous control mode can be enable/disable via register setting. Note the GPIO output enable is always directly connected (no synchronization feature).

See **Figure 4.5, Synchronization Principle and Capture on Event**.

- Trigger synchronous control mode disable (Transparent mode):
  - The GPIO data signals are directly driven per BGPIO signals, no capture and latch, internal input/output signals and output enable signals are directly connected
  - The bit[n] in System Control CFG_GPIOT_PTEN_mj register should be set to “0” in this case
    a. With m: 1, 2, 3 → Depending on reference port BGPIO1, BGPIO2, BGPIO3
    b. With j: A, B → Depending on if current port is A or B
    c. With n = 0..31 → Bit number of current port addressing
- Trigger synchronous control mode enable:
  - The BGPIO internal input signals and output signals are updated in synchronization with an interrupt (GPIO_TRIGGER[3:0] signals) from an on-chip peripheral function selected inside System Controller.
  - The rising edge detection of GPIO_TRIGGER[3:0] signal starts a capture on output and input signals synchronized with BGPIO_PCLK
  - The BGPIO internal input signals and output signals are latched and not changed from last capture event if the rising edge of GPIO_TRIGGER[3:0] signal is not detected
  - The bit[n] in System Control CFG_GPIOT_PTEN_mj register should be set to “1” in this case
  - The capture and latch functions are not available on output enable signals (directly connected)

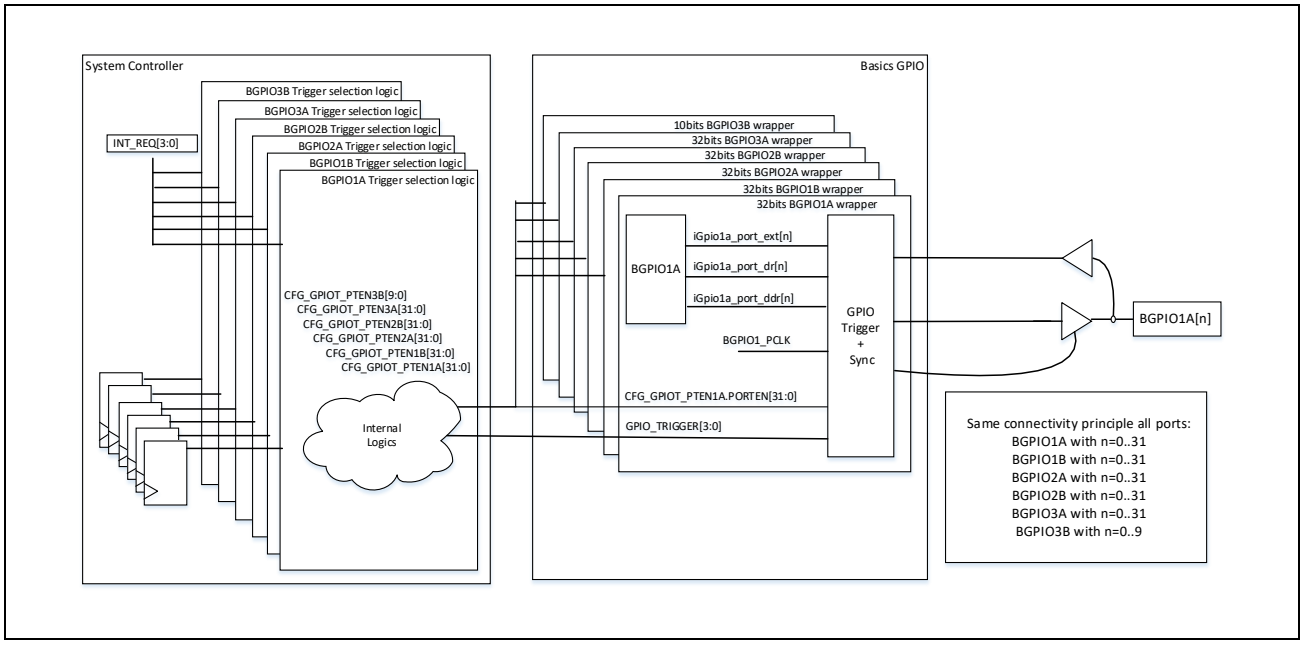

Figure 4.5 Synchronization Principle and Capture on Event

Table 4.19 Trigger Synchronous Operation Allocation Line

BGPIO Signal Name	GPIO Trigger Signal Name	PTEN Signal Name
BGPIO1A		
BGPIO1A[7:0]	GPIO_TRIGGER[0]	CFG_GPIOT_PTEN1A.PORTEN [7:0]
BGPIO1A[15:8]	GPIO_TRIGGER[1]	CFG_GPIOT_PTEN1A.PORTEN [15:8]
BGPIO1A[23:16]	GPIO_TRIGGER[2]	CFG_GPIOT_PTEN1A.PORTEN [23:16]
BGPIO1A[31:24]	GPIO_TRIGGER[3]	CFG_GPIOT_PTEN1A.PORTEN [31:24]
BGPIO1B		
BGPIO1B[7:0]	GPIO_TRIGGER[0]	CFG_GPIOT_PTEN1B.PORTEN [7:0]
BGPIO1B[15:8]	GPIO_TRIGGER[1]	CFG_GPIOT_PTEN1B.PORTEN [15:8]
BGPIO1B[23:16]	GPIO_TRIGGER[2]	CFG_GPIOT_PTEN1B.PORTEN [23:16]
BGPIO1B[31:24]	GPIO_TRIGGER[3]	CFG_GPIOT_PTEN1B.PORTEN [31:24]
BGPIO2A		
BGPIO2A[7:0]	GPIO_TRIGGER[0]	CFG_GPIOT_PTEN2A.PORTEN [7:0]
BGPIO2A[15:8]	GPIO_TRIGGER[1]	CFG_GPIOT_PTEN2A.PORTEN [15:8]
BGPIO2A[23:16]	GPIO_TRIGGER[2]	CFG_GPIOT_PTEN2A.PORTEN [23:16]
BGPIO2A[31:24]	GPIO_TRIGGER[3]	CFG_GPIOT_PTEN2A.PORTEN [31:24]
BGPIO2B		
BGPIO2B[7:0]	GPIO_TRIGGER[0]	CFG_GPIOT_PTEN2B.PORTEN [7:0]
BGPIO2B[15:8]	GPIO_TRIGGER[1]	CFG_GPIOT_PTEN2B.PORTEN [15:8]
BGPIO2B[23:16]	GPIO_TRIGGER[2]	CFG_GPIOT_PTEN2B.PORTEN [23:16]
BGPIO2B[31:24]	GPIO_TRIGGER[3]	CFG_GPIOT_PTEN2B.PORTEN [31:24]
BGPIO3A		
BGPIO3A[7:0]	GPIO_TRIGGER[0]	CFG_GPIOT_PTEN3A.PORTEN [7:0]
BGPIO3A[15:8]	GPIO_TRIGGER[1]	CFG_GPIOT_PTEN3A.PORTEN [15:8]
BGPIO3A[23:16]	GPIO_TRIGGER[2]	CFG_GPIOT_PTEN3A.PORTEN [23:16]
BGPIO3A[31:24]	GPIO_TRIGGER[3]	CFG_GPIOT_PTEN3A.PORTEN [31:24]
BGPIO3B		
BGPIO3B[7:0]	GPIO_TRIGGER[0]	CFG_GPIOT_PTEN3B.PORTEN [7:0]
BGPIO3B[9:8]	GPIO_TRIGGER[1]	CFG_GPIOT_PTEN3B.PORTEN [10:8]

Table 4.20 Trigger Synchronous Operation Basics Function

GPIO Trigger Mode	GPIO Trigger	Status of BGPIO Signals
CFG_GPIOT_PTEN[m]A CFG_GPIOT_PTEN[m]B	GPIO_TRIGGER[3:0]	BGPIO[m]A BGPIO[m]B
1’b0	1’bx	Transparent mode
1’b1	On rising edge detection	Trigger synchronous control mode, Event capture
1’b1	No rising detection	Trigger synchronous control mode, no change from last event capture

Note: m = 1..3

4.6 Usage Notes

4.6.1 Programming Consideration

Programming the BGPIO registers for interrupt capability, edge sensitive or level sensitive interrupts, and interrupt polarity should be completed prior to enable the interrupts on BGPIO1 port A, BGPIO2 port A and BGPIO3 port A in order to prevent glitches on the interrupt lines to the interrupt controller.

CAUTION

- In order to prevent glitches, programming the bGPIO_ls_sync bit with “32’h1” value in rGPIO_ls_sync register.
- Writing to the interrupt clear register clears an edge-detected interrupt and has no effect on a level-sensitive interrupt.

Section 5 Timer Block

The Peripheral Group1 (PG1) Subsystem includes 2 timer blocks which have 8 Sub-timers.

5.1 Overview

- 2 units
- 6 programmable timers 16 bits, Sub-timer (0..5)
- 2 programmable timers 32 bits, Sub-timer (6..7)
- Each Sub-timer generates single interrupt
- Pre-scaler selectable between 2 time bases
  - 25 MHz or 1 MHz
- 2 Function modes
  - Auto-reload mode:
    An interrupt is activated at pre-set value time. A counter is automatically cleared and restarts.
  - Single-shot mode:
    An interrupt is activated at pre-set value time. A counter is stopped and disabled.
- DMA Coupling
  - DMAC flow controller mode is used
  - Start a DMA transaction at interrupt timing
  - Available only on Sub-timer (6..7)

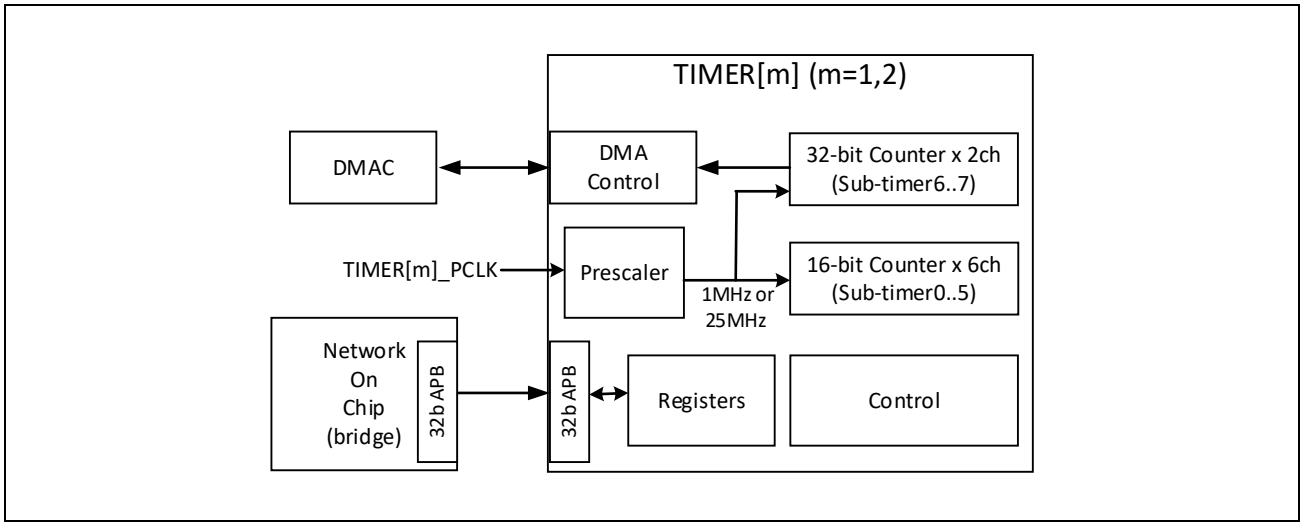

Figure 5.1 TIMER Interfaces and Connections

5.2 Signal Interfaces

Signal Name	Input Output	Description
Clock		
TIMER[m]_PCLK	Input	Internal bus clock (APB) and prescaler input clock, 25 MHz
Interrupt		
TIMER[m]_Int[7:0]	Output	Level sensitive interrupt output for each Sub-timer, Active High

Note: m =1 or 2
Index removed description style is used in this chapter.
Ex) TIMER_PCLK

5.3 Register Map

5.3.1 TIMER1 Register Map

Table 5.1 TIMER1 Register Map

Address	Register Symbol	Register Name
5100 1000h + 20h × n	rTimerLoadCount_[n] (n = 0..7)	Preset Value of Sub-timer[n]
5100 1004h + 20h × n	rTimerCurrentCount_[n] (n = 0..7)	Current Value of Sub-timer[n]
5100 1008h + 20h × n	rTimerControl_[n] (n = 0..7)	Control Mode of Sub-timer[n]
5100 100Ch + 20h × n	rTimerClearInt_[n] (n = 0..7)	Clears the Interruption of Sub-timer[n]
5100 1010h + 20h × n	rTimerStatusInt0_[n] (n = 0..7)	Interruption Status before Masking of Sub-timer[n]
5100 1014h + 20h × n	rTimerStatusInt1_[n] (n = 0..7)	Interruption Status after Masking of Sub-timer[n]
5100 1100h	rTimerAllClearInt	Clear All Interrupt
5100 1104h	rTimerAllStatusInt0	All Interrupts Status before Masking
5100 1108h	rTimerAllStatusInt1	All Interrupts Status after Masking
5100 110Ch	rTimer_DMA_Pending	TIMER DMA Requests Status
5100 1110h	rTimer_DMA_PendingOvf	TIMER DMA Overflow Status
5100 1114h	rTimer_DMA_PendingClrOvf	TIMER DMA Overflow Clear

5.3.2 TIMER2 Register Map

Table 5.2 TIMER2 Register Map

Address	Register Symbol	Register Name
5100 2000h + 20h × n	rTimerLoadCount_[n] (n = 0..7)	Preset Value of Sub-timer[n]
5100 2004h + 20h × n	rTimerCurrentCount_[n] (n = 0..7)	Current Value of Sub-timer[n]
5100 2008h + 20h × n	rTimerControl_[n] (n = 0..7)	Control Mode of Sub-timer[n]
5100 200Ch + 20h × n	rTimerClearInt_[n] (n = 0..7)	Clears the Interruption of Sub-timer[n]
5100 2010h + 20h × n	rTimerStatusInt0_[n] (n = 0..7)	Interruption Status before Masking of Sub-timer[n]
5100 2014h + 20h × n	rTimerStatusInt1_[n] (n = 0..7)	Interruption Status after Masking of Sub-timer[n]
5100 2100h	rTimerAllClearInt	Clear All Interrupt
5100 2104h	rTimerAllStatusInt0	All Interrupts Status before Masking
5100 2108h	rTimerAllStatusInt1	All Interrupts Status after Masking
5100 210Ch	rTimer_DMA_Pending	TIMER DMA Requests Status
5100 2110h	rTimer_DMA_PendingOvf	TIMER DMA Overflow Status
5100 2114h	rTimer_DMA_PendingClrOvf	TIMER DMA Overflow Clear

5.4 Register Description

5.4.1 rTimerLoadCount_[n] — Preset Value of Sub-timer[n] (n = 0..5)

Address: 5100 1000h + 20h × n (TIMER1)
5100 2000h + 20h × n (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bTimerLoadCount															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.3 rTimerLoadCount_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used.	R
b15 to b0	bTimerLoadCount	Preset value The timer counts up from zero until preset value. ***Caution*)** • The preset value must be different from 0. If preset value = 0, the timer doesn't run. • When the timer runs (bTimerEnable set "1"), if the CPU writes the new preset value, it is immediately recognized. • When timer mode is auto-reload, the periodic cycle is (bTimerLoadCount + 1). See **Section 5.6, Usage Notes** if the firmware modifies bTimerLoadCount when timer is running.	R/W

5.4.2 rTimerLoadCount_[n] — Preset Value of Sub-timer[n] (n = 6..7)

Address: 5100 1000h + 20h × n (TIMER1)
5100 2000h + 20h × n (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bTimerLoadCount															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bTimerLoadCount															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.4 rTimerLoadCount_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bTimerLoadCount	Preset value The timer counts up from zero until preset value. ***Caution*)** • The preset value must be different from 0. If preset value = 0, the timer doesn't run. • When the timer runs (bTimerEnable set "1"), if the CPU writes the new preset value, it is immediately recognized. • When timer mode is auto-reload, the periodic cycle is (bTimerLoadCount + 1). See **Section 5.6, Usage Notes** if the firmware modifies bTimerLoadCount when timer is running.	R/W

5.4.3 rTimerCurrentCount_[n] — Current Value of Sub-timer[n] (n = 0..5)

Address: 5100 1004h + 20h × n (TIMER1)
5100 2004h + 20h × n (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bTimerCurrentCount															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.5 rTimerCurrentCount_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used.	R
b15 to b0	bTimerCurrentCount	Current value of timer When the current value of one timer equals preset value then: An interruption is triggered. The timer is cleared if bTimerEnable is set to 1.	R

5.4.4 rTimerCurrentCount_[n] — Current Value of Sub-timer[n] (n = 6..7)

Address: 5100 1004h + 20h × n (TIMER1)
5100 2004h + 20h × n (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bTimerCurrentCount															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bTimerCurrentCount															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.6 rTimerCurrentCount_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bTimerCurrentCount	Current value of timer When the current value of one timer equals preset value then: An interruption is triggered. The timer is cleared if bTimerEnable is set to 1.	R

5.4.5 rTimerControl_[n] — Control Mode of Sub-timer[n] (n = 0..7)

Address: 5100 1008h + 20h × n (TIMER1)
5100 2008h + 20h × n (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bTimer DMAEnable	bTimer MaskInt	bTimer Enable	bTimer Mode	bTimer Prescaler
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.7 rTimerControl_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Not used.	R
b4	bTimerDMAEnable	DMA channel Enable 1: DMA channel enable 0: DMA channel disable • All current DMA request are forced in idle state, after DMA acknowledge reception for current access. ***Note*)** • Only implemented on rTimerControl_6 and 7. • Not used and Reserved on rTimerControl_0..5.	R/W
b3	bTimerMaskInt	Interruption mask of Sub-timer 0: Masked 1: Not Masked	R/W
b2	bTimerEnable	0: Disable timer and keep a current value • Stops the timer, it does not increment bTimerCurrentCount keeps the same value. 1: Reset and start timer • Sample status of bTimerMode and bTimerPrescaler • Reset the prescaler counter • Reset bTimerCurrentCount • Start the timer in increment mode ***Caution*)** To rearm timer, the user must disable timer by writing “0” on bTimerEnable and enable timer by writing “1” on bTimerEnable.	R/W
b1	bTimerMode	This bit allows to select the operation mode of the timer, according to the encoding below: 0: Single-shot • The timer is incremented up to preset value. • At preset value time, an interruption is activated, the timer is stopped and disabled. 1: Auto-reload • The timer is incremented up to preset value. • At preset value time, an interruption is activated, the timer is automatically cleared and restarts. bTimerMode is sampled when the firmware set “1” on bTimerEnable (rising edge).	R/W
b0	bTimerPrescaler	This bit controls the prescaler configuration, according to encoding below: 1: Time base 1 MHz (1 µs) 0: Time base 25 MHz bTimerPrescaler is sampled when the firmware set “1” on bTimerEnable (rising edge).	R/W

5.4.6 rTimerClearInt_[n] — Clears the Interruption of Sub-timer[n] (n = 0..7)

Address: 5100 100Ch + 20h × n (TIMER1)
5100 200Ch + 20h × n (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bTimer ClearInt
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.8 rTimerClearInt_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Not used.	R
b0	bTimerClearInt	Reading from this register returns zero (0) and clears the interruption of timer.	R

5.4.7 rTimerStatusInt0_[n] — Interruption Status before Masking of Sub-timer[n] (n = 0..7)

Address: 5100 1010h + 20h × n (TIMER1)
5100 2010h + 20h × n (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bTimerStatusInt0
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.9 rTimerStatusInt0_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Not used.	R
b0	bTimerStatusInt0	An interruption is generated on rising edge of TIMER_Int[n] Interruption status before masking of Sub-timer. 0: CPU interruption is not active on Sub-timer 1: Interruption bit has been set on Sub-timer Reading from this register does not clear any active interrupts.	R

5.4.8 rTimerStatusInt1_[n] — Interruption Status after Masking of Sub-timer[n] (n = 0..7)

Address: 5100 1014h + 20h × n (TIMER1)
5100 2014h + 20h × n (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bTimerStatusInt1
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.10 rTimerStatusInt1_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b1	Reserved	Not used.	R
b0	bTimerStatusInt1	An interruption is generated on rising edge of TIMER_Int[n] Interruption status after masking of Sub-timer. 0: CPU interruption is not active on Sub-timer or interrupt is masked 1: CPU interruption bit has been set on Sub-timer and not masked Reading from this register does not clear any active interrupts.	R

5.4.9 rTimerAllClearInt — Clear All Interrupt

Address: 5100 1100h (TIMER1)
5100 2100h (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bTimerAllClearInt							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.11 rTimerAllClearInt Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Not used.	R
b7 to b0	bTimerAllClearInt	Reading this register returns all zeroes (0) and clears all active interruptions.	R

5.4.10 rTimerAllStatusInt0 — All Interrupts Status before Masking

Address: 5100 1104h (TIMER1)
5100 2104h (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bTimerAllStatusInt0							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.12 rTimerAllStatusInt0 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Not used.	R
b7 to b0	bTimerAllStatusInt0	Interruption status before masking of all Sub-timers. With following bit allocation: Bit7: Dedicated for Sub-timer7 ... Bit0: Dedicated for Sub-timer0 For each bit 0: The corresponding Sub-timer CPU interruption is not active 1: The potential corresponding Sub-timer CPU interruption has been set, depending on bTimerMaskInt bit Reading from this register does not clear any active interrupts.	R

5.4.11 rTimerAllStatusInt1 — All Interrupts Status after Masking

Address: 5100 1108h (TIMER1)
5100 2108h (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bTimerAllStatusInt1							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.13 rTimerAllStatusInt1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Not used.	R
b7 to b0	bTimerAllStatusInt1	Interruption status after masking of all Sub-timers. With following bit allocation: Bit7: Dedicated for Sub-timer7 ... Bit0: Dedicated for Sub-timer0 For each bit 0: The corresponding Sub-timer CPU interruption is not active 1: The corresponding Sub-timer CPU interruption has been set Reading from this register does not clear any active interrupts.	R

5.4.12 rTimer_DMA_Pending — TIMER DMA Requests Status

DMA feature is only implemented on Sub-timer 6 and 7.

TIMER DMA requests are started and run until detection end of DMA transfer.

Address: 5100 110Ch (TIMER1)
5100 210Ch (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bTimer_DMA_Running_7	bTimer_DMA_Running_6	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.14 rTimer_DMA_Pending Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Not used.	R
b7	bTimer_DMA_Running_7	DMA running status of Sub-timer7 Sub-timer7 counts up from a programmed value and generate an interrupt when the count reaches the preset value. When a rising edge of interrupt is detected, a TIMER DMA requests are started and run until detection end of DMA transfer. 1: DMA requests are running 0: No TIMER DMA request on same channel ***Note*)** • The bit will be automatically cleared when end of DMA transfer is detected on this channel. • If contention exists where this bit receives both a request to set and a request to clear on the same cycle, regardless of the source of either, this bTimer_DMA_Running_7 bit will be clear and the request to set will be ignored. In this case a DMA request process is not restarted. • In this case the overflow bTimer_DMA_RunningOvf_7 bit in the rTimer_DMA_PendingOvf register will be set regardless of whether this bit was previously set or not.	R
b6	bTimer_DMA_Running_6	Same as bTimer_DMA_Running_7	R
b5 to b0	Reserved	Not used.	R

5.4.13 rTimer_DMA_PendingOvf — TIMER DMA Overflow Status

DMA feature is only implemented on Sub-timer 6 and 7.

Indicates that a new TIMER DMA requests was generated with a previous request was already running.

Address: 5100 1110h (TIMER1)
5100 2110h (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bTimer_DMA_RunningOvf_7	bTimer_DMA_RunningOvf_6	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.15 rTimer_DMA_PendingOvf Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Not used.	R
b7	bTimer_DMA_RunningOvf_7	Overflow with TIMER DMA is running on DMA requests of Sub-timer7. Indicates that a new TIMER DMA requests was generated with a previous request was already running on Sub-timer7. 1: TIMER DMA overflow on DMA requests 0: No TIMER DMA overflow ***Note***) • An overflow condition does not stop TIMER DMA transfer from being processed. It simply is an indication that a DMA request was missed. • This means that the first DMA transfer block will be processed normally until the DMA finish occurred but the second DMA transfer block triggered is not processed.	R
b6	bTimer_DMA_RunningOvf_6	Same as bTimer_DMA_RunningOvf_7	R
b5 to b0	Reserved	Not used.	R

5.4.14 rTimer_DMA_PendingClrOvf — TIMER DMA Overflow Clear

DMA feature is only implemented on Sub-timer 6 and 7.

Address: 5100 1114h (TIMER1)
5100 2114h (TIMER2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bTimer_DMA_RunningClrOvf_7	bTimer_DMA_RunningClrOvf_6	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.16 rTimer_DMA_PendingClrOvf Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Not used.	R
b7	bTimer_DMA_RunningClrOvf_7	Clear Overflow with TIMER DMA is running on DMA requests of Sub-timer7. 1: Clear Overflow bit 0: No Action ***Note*)** • If software tries to write 1 to bTimer_DMA_RunningClrOvf_7 bit on the same clock cycle that hardware tries to set the overflow bTimer_DMA_RunningOvf_7 bit in the rTimer_DMA_PendingOvf register, then hardware has priority and the bTimer_DMA_RunningOvf_7 bit will be set. • When read, this bit returns “0”.	W
b6	bTimer_DMA_RunningClrOvf_6	Same as bTimer_DMA_RunningClrOvf_7	W
b5 to b0	Reserved	Not used.	R

5.5 Operation

5.5.1 Prescaler Counter

Each sub-timer has an independent clock input that it is connected to 25 MHz or 1 MHz. The prescaler counter builds a clock in order to increment the timer.

2 clocks can be selected by a signal bTimerPrescaler:

25 MHz:

- Each sub-timer uses directly TIMER_PCLK (25 MHz)

1 MHz (using prescaler counter):

- The divider by 25 builds a new clock from TIMER_PCLK (25 MHz).
- The divider is a counter in up mode (counting from 0 to 24)
- Set bTimerEnable to “1”, initialize the prescaler counter on reset value (0) and after that starting a new cycle (0, 1, 2 ... 22, 23, 24)
- A rise edge on bTimerEnable resets the prescaler counter to “0”

5.5.2 Counter 16 or 32 Bits

Sub-timers count up from 0 until a programmed value and generate an interruption when the count reaches the preset value. On each Sub-timer, a single interruption is generated, which is active whenever any of the individual timer interrupts is active. All interruption status registers can be accessed at any time.

The preset value for each Sub-timer is loaded into a dedicated rTimerLoadCount_[n] register.

Two events can clear the timer:

- Enable timer after reset or disabled
- Timer counts up to preset value

Two functions mode are implemented:

- Auto-reload mode:
  - The timer is incremented up to preset value.
  - At preset value time an interruption is activated, the timer is automatically cleared and restarts.
- Single-shot mode:
  - The timer is incremented up to preset value.
  - At preset value time an interruption is activated, the timer is stopped and disabled.

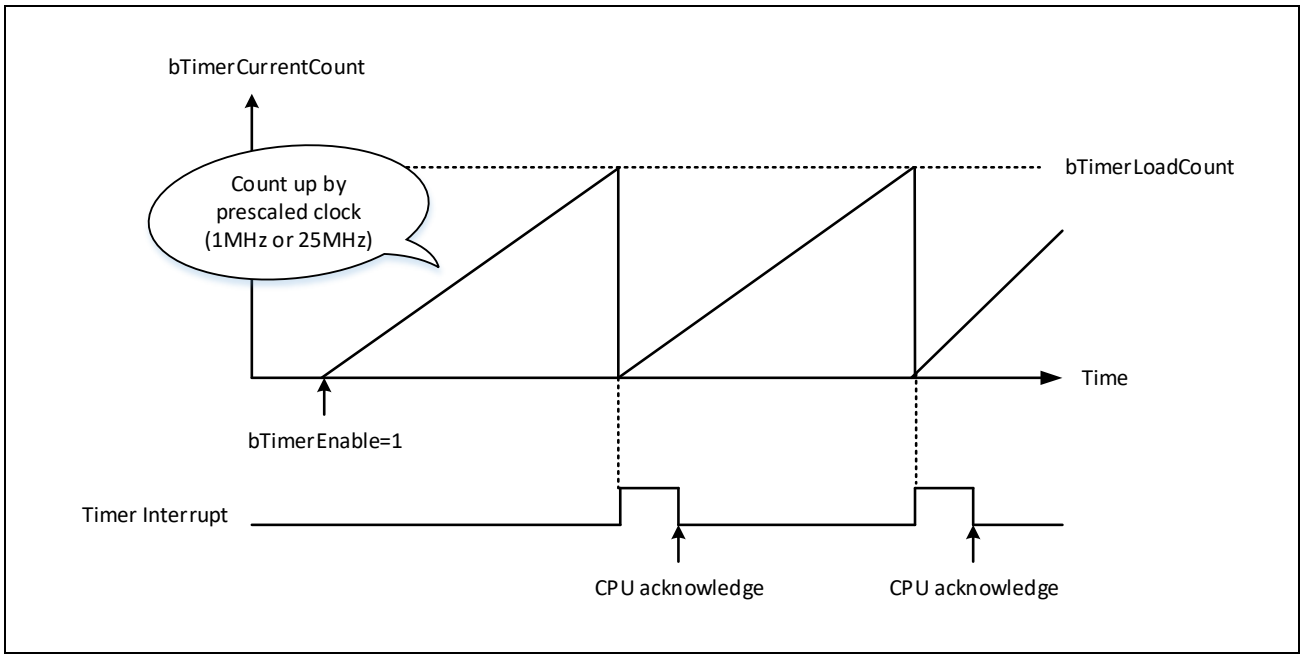

Figure 5.2 Timer Mode (bTimerMode) is auto-reload

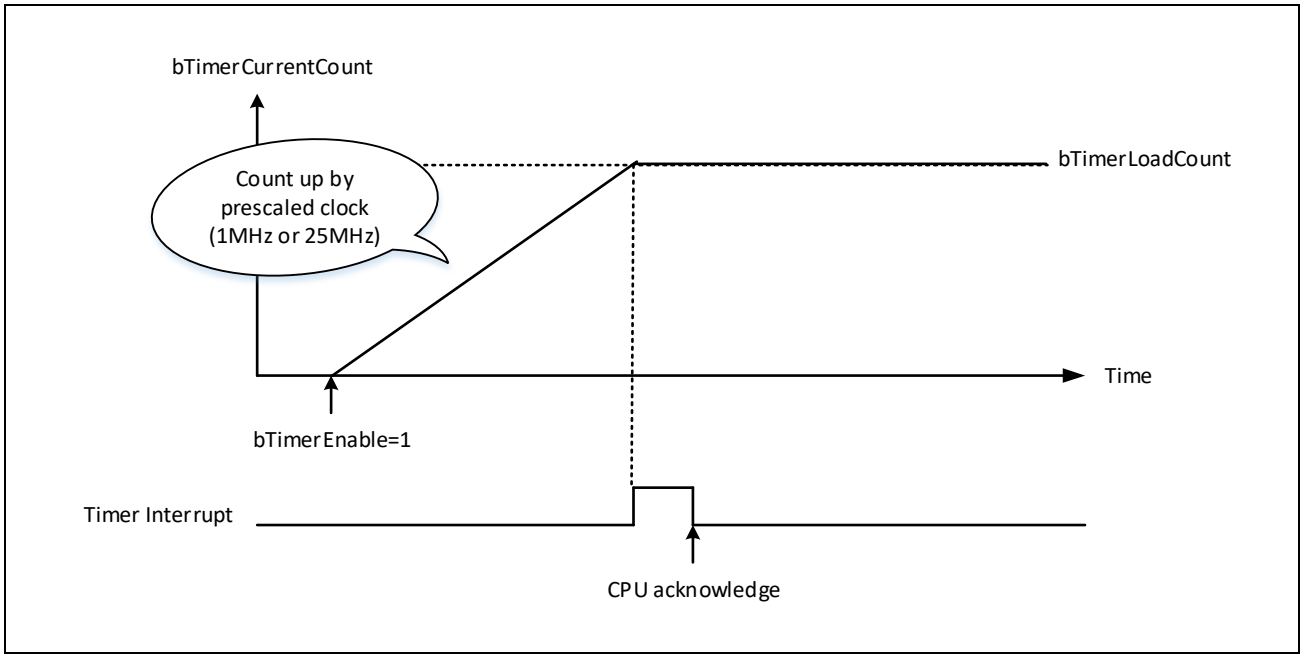

Figure 5.3 Timer Mode (bTimerMode) is single-shot

It can disable or enable a timer.

[Disable]

- Stops timer increment, bTimerCurrentCount keeps the same value.

[Enable]

- Clears the prescaler counter.
- Clears the current counter bTimerCurrentCount.
- Restart timer increment.

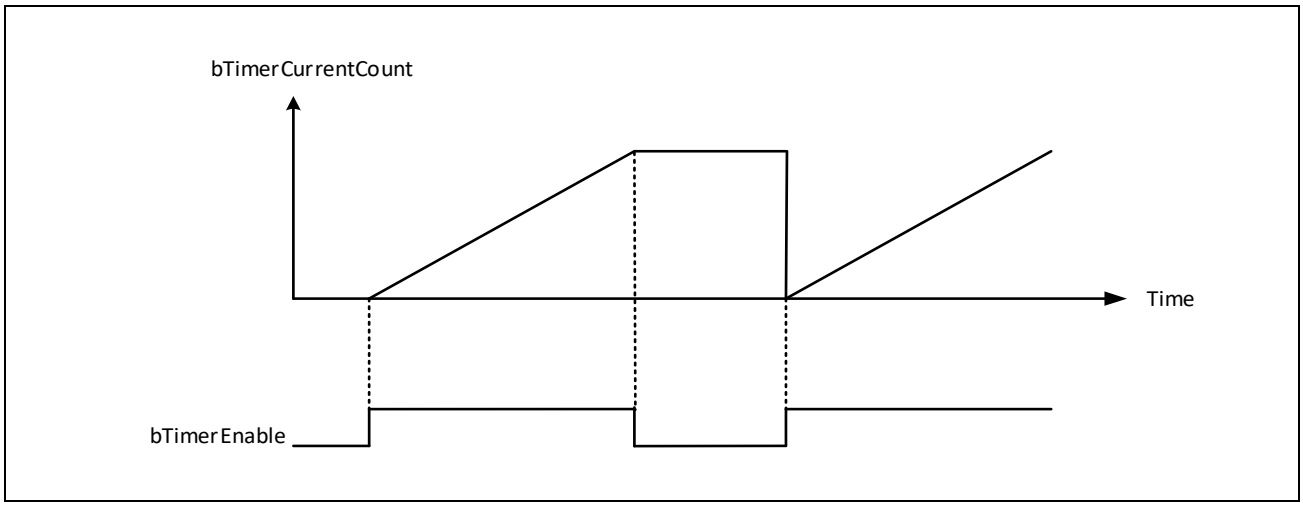

Figure 5.4 bTimerEnable Off and On

5.5.3 Interruption

Timers count up to a programmed value and generate an interruption when the count reaches the preset value. On each Sub-timer, single interruption is generated, which is active whenever any of the individual timer interrupts is active.

An interruption can be acknowledged only on CPU read in bTimerClearInt bit.

An example for bTimerLoadCount = 5 case is shown as below.

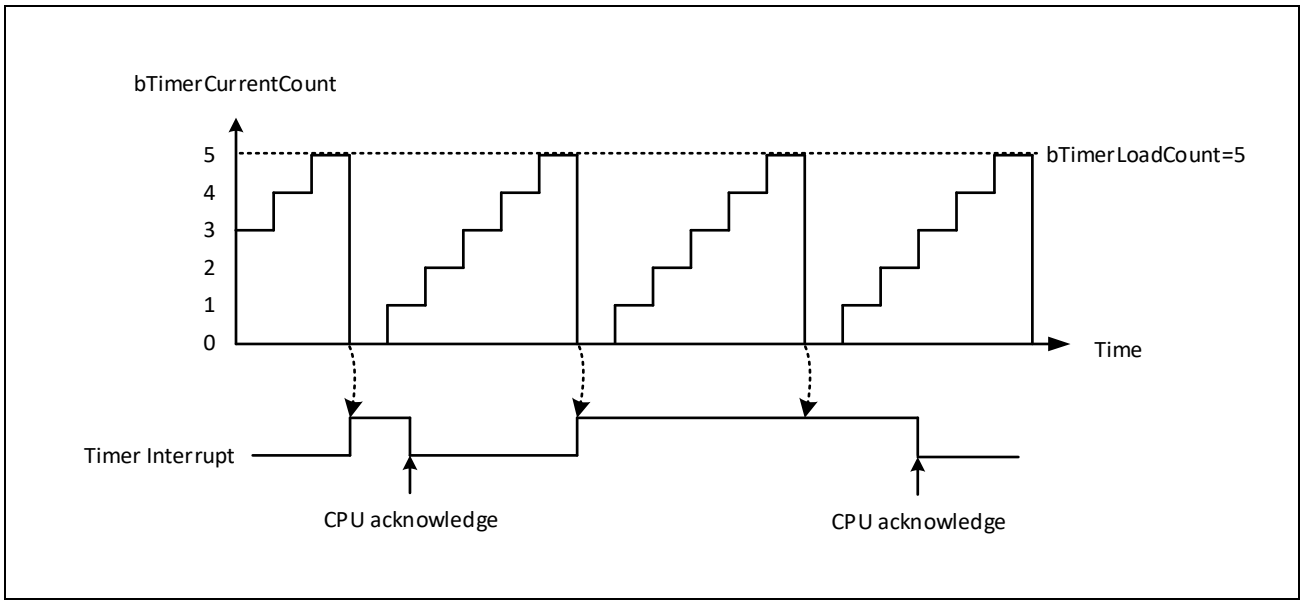

Figure 5.5 Timer Interruption

5.5.4 DMA Control

The TIMER has optional DMA capability. It has a handshaking interface to a DMA Controller to request and control transfers. In this mode, DMA controller must be configured in DMAC flow controller mode. The DMA always transfers data using DMA burst transactions if possible, for efficiency.

TIMER has two DMA channel to transfer a memory block from source to destination.

For each timer block, the TIMER DMA flow control is managed by these followings DMA bits:

- bTimerDMAEnable bit in rTimerControl_[n] register (n = 6 or 7)
  - Enable or disable DMA channel of Sub-timer[n] (n = 6 or 7)
- Interrupt of Sub-timer[n] (n = 6 or 7)
  - Sub-timer[n] counts up from a programmed value and generate an interrupt when the count reaches the preset value.
  - When a rising edge of interrupt is detected, a TIMER DMA request is started and run until detection end of DMA transfer.
- bTimer_DMA_Running_[n] (n = 6 or 7) bit in rTimer_DMA_Pending register
  - TIMER DMA request is started and run until detection end of DMA transfer.

To enable the DMA Controller interface on the TIMER and enable the handshaking interface:

- DMA controller must be configured.
  - Source and destination addresses
  - Burst size on source and destination
  - Block size to transfer
  - DMAC flow controller mode
  - Channel allocation
  - Only one block
  - Interrupt setting
- Set bTimerDMAEnable bit in rTimerControl_[n] register (n = 6 or 7) to enable TIMER DMA channel.
  - When a rising edge of Timer interrupt is detected on Sub-timer[n], a TIMER DMA request is started and run until detection end of DMA transfer.
- bTimer_DMA_Running_[n] (n = 6 or 7) bit in rTimer_DMA_Pending register gives a current status on DMA transaction in running.

CAUTION

DMA Controller must be configured in DMAC flow controller mode, because TIMER do not know the size of block transferred.

5.6 Usage Notes

CAUTION

In order to avoid potential synchronization problems when initializing, loading, and enabling a TIMER module, you should follow the basic procedure below.

(1) Initialize the Sub-timer through the rTimerControl_[n]

- Disable the Sub-timer by writing a “0” of bTimerEnable.
- Masking of interruption of Sub-timer by writing a “0” of bTimerMaskInt

(2) Reset a potential interruption of Sub-timer through the rTimerClearInt_[n]

- Cleans the interruption by reading a “0” of bTimerClearInt

(3) Initialize the Sub-timer value through the rTimerLoadCount_[n]

- Write a value on 16/32 bits of bTimerLoadCount.
- Warning, 0 is strictly forbidden, the Sub-timer does not run.

(4) Initialize the Sub-timer through the rTimerControl_[n] register

- Select timer mode
- Select prescaler mode
- No Masking interruption of Sub-timer by writing a “1” to bTimerMaskInt.
- Enable the Sub-timer by writing a “1” to bTimerEnable.

CAUTION

Warning if the firmware modifies bTimerLoadCount when Sub-timer is running:

- For example, when Sub-timer is enabled with bTimerCurrentCount = “6” and bTimerLoadCount = “10”.
- If the firmware loads a new value to bTimerLoadCount = “5”, then preset is detected and an interruption is triggered.
- It means that when Sub-timer is enabled and the firmware loads a new value into bTimerLoadCount which is less than bTimerCurrentCount and different from zero, preset is detected and an interruption is triggered.
- After this, the behavior of Sub-timer depends on bTimerMode (auto-reload mode or single-shot mode).

Section 6 CAN

6.1 Overview

RZ/N1 has two instances of CAN controller with “Sync frame” transmission mechanism.

For CAN 2.0:

- Supports full CAN 2.0 – both 2.0A (equivalent to CAN 1.2) and 2.0B
- Supports both 11-bit and 29-bit identifiers
- Supports bit rates from less than 125 Kbps to more than 1 Mbps
- 64 bytes Receive FIFO
- Acceptance filtering
- Software-driven bit-rate detection (offering hot plug in support)
- Single-shot transmission option, listen only mode, reception of “own” messages
- Arbitration lost interrupt with data of bit position
- Read/write error counters
- Last error register
- Programmable error limit warning
- Broadly compatible with Philips SJA1000 in its PeliCAN mode

Additional features:

Transmission periodic “Sync frame” for CANopen®.

- Programmable time base (in bit period unit) to configure:
  - Period of “Sync frame” emission (the emission of the “Sync frame” can be deactivated)
  - Mask frame time
  - Passive error detection system

Specification 2.0A (which is equivalent to CAN 1.2) covers standard message formats (11-bit identifier). Specification 2.0B covers both standard and extended message formats (both 11-bit and 29-bit identifiers).

The CAN controller is broadly compatible with a Philips SJA1000 working in its PeliCAN mode but with some exceptions (detailed in Difference between CAN Controllers and Reference Philips SJA1000 Devices chapter).

This specification should be read in conjunction with the BOSCH CAN specification version 2.0 (hereafter referred to as the CAN 2.0 specification).

CAUTION

- The CAN 2.0 specification uses the convention that, on the CAN bus, Recessive bits are logic “1” while Dominant bits are logic “0”.
- This convention is also used in this specification.

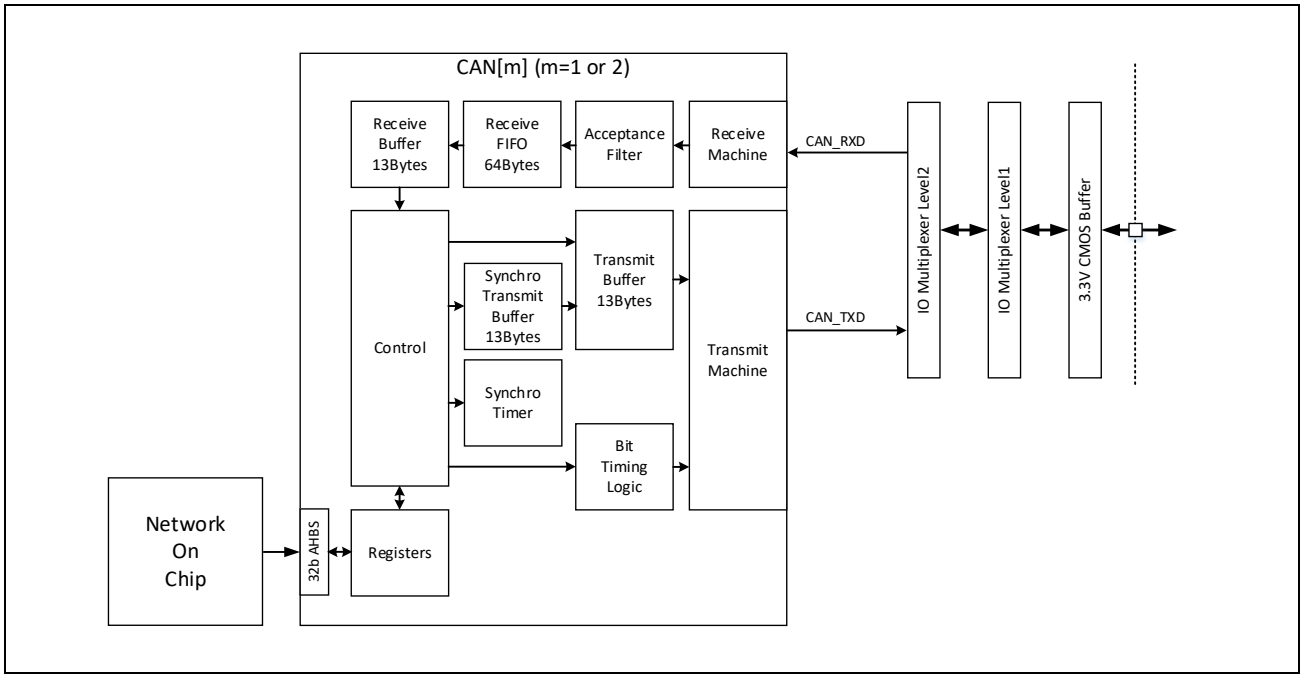

Figure 6.1 CAN Synoptic

6.2 Signal Interfaces

Signal Name	Input Output	Description
Clock		
CAN[m]_HCLK	Input	Internal bus clock (AHB), 48 MHz fixed
Interrupt		
CAN[m]_Int	Output	Level sensitive interrupt output, Active High
External Signal		
CAN[m]_RXD	Input	Receive data
CAN[m]_TXD	Output	Transmit data

Note: m = 1 or 2.
Index removed style is used in this chapter.
Ex.) CAN_HCLK

6.3 Register Map

6.3.1 Register Map (CAN1)

Table 6.1 CAN1 Register Map

Address	Register Symbol	Register Name
5210 4000h	rCan_MOD	Configuration Mode Register
5210 4004h	rCan_CMR	Command Register
5210 4008h	rCan_SR	Controller Status Register
5210 400Ch	rCan_IR	Interrupt Register
5210 4010h	rCan_IER	Interrupt Event Register
5210 4018h	rCan_BTR0	Bus Timing Register 0
5210 401Ch	rCan_BTR1	Bus Timing Register 1
5210 4020h	rCan_OCR	Output Control Register
5210 402Ch	rCan_ALC	Arbitration Lost Capture Register
5210 4030h	rCan_ECC	Error Code Capture Register
5210 4034h	rCan_EWLR	Error Warning Limit Register
5210 4038h	rCan_RXERR	Receive Error Counter Register
5210 403Ch	rCan_TXERR	Transmit Error Counter Register
5210 4040h to 5210 4070h	rCan_WrTransmitBuffer	Write Transmit Buffer Register
	rCan_RdReceiveBuffer	Read Receive Buffer Register
5210 4040h + 4h × n	rCan_ACR[n] (n = 0..3)	Acceptance Code Filter [n] Register
5210 4050h + 4h × n	rCan_AMR[n] (n = 0..3)	Acceptance Mask Filter [n] Register
5210 4074h	rCan_RMC	Receive Message Counter Register
5210 4078h	rCan_RBSA	Receive Buffer Start Address Register
5210 4080h to 5210 417Ch	rCan_ReceiveFifo	Receive FIFO Register
5210 4180h to 5210 41B0h	rCan_RdTransmitBuffer	Read Transmit Buffer Register
5210 4440h to 5210 4470h	rCan_SyncTransmitBuffer	Sync Frame Transmit Buffer Register
5210 4480h	rCan_SyncPeriod	Time Window Sync Frame Transmission Register
5210 4488h	rCan_SyncStatusInt	Sync Frame Interrupt Status Register
5210 448Ch	rCan_SyncMaskInt	Sync Frame Mask Interrupt Register
5210 4490h	rCan_SyncClearInt	Sync Frame Clear Interrupt Register
5210 4494h	rCan_SyncStatus	Sync Frame Status Configuration Register
5210 4498h	rCan_SyncClearSetRunStop	Sync Frame Generation Register
5210 44A0h	rCan_SyncPassiveError	Sync Passive Error Detection Register

6.3.2 Register Map (CAN2)

Table 6.2 CAN2 Register Map

Address	Register Symbol	Register Name
5210 5000h	rCan_MOD	Configuration Mode Register
5210 5004h	rCan_CMR	Command Register
5210 5008h	rCan_SR	Controller Status Register
5210 500Ch	rCan_IR	Interrupt Register
5210 5010h	rCan_IER	Interrupt Event Register
5210 5018h	rCan_BTR0	Bus Timing Register 0
5210 501Ch	rCan_BTR1	Bus Timing Register 1
5210 5020h	rCan_OCR	Output Control Register
5210 502Ch	rCan_ALC	Arbitration Lost Capture Register
5210 5030h	rCan_ECC	Error Code Capture Register
5210 5034h	rCan_EWLR	Error Warning Limit Register
5210 5038h	rCan_RXERR	Receive Error Counter Register
5210 503Ch	rCan_TXERR	Transmit Error Counter Register
5210 5040h to 5210 5070h	rCan_WrTransmitBuffer	Write Transmit Buffer Register
	rCan_RdReceiveBuffer	Read Receive Buffer Register
5210 5040h + 4h × n	rCan_ACR[n] (n = 0..3)	Acceptance Code Filter [n] Register
5210 5050h + 4h × n	rCan_AMR[n] (n = 0..3)	Acceptance Mask Filter [n] Register
5210 5074h	rCan_RMC	Receive Message Counter Register
5210 5078h	rCan_RBSA	Receive Buffer Start Address Register
5210 5080h to 5210 517Ch	rCan_ReceiveFifo	Receive FIFO Register
5210 5180h to 5210 51B0h	rCan_RdTransmitBuffer	Read Transmit Buffer Register
5210 5440h to 5210 5470h	rCan_SyncTransmitBuffer	Sync Frame Transmit Buffer Register
5210 5480h	rCan_SyncPeriod	Time Window Sync Frame Transmission Register
5210 5488h	rCan_SyncStatusInt	Sync Frame Interrupt Status Register
5210 548Ch	rCan_SyncMaskInt	Sync Frame Mask Interrupt Register
5210 5490h	rCan_SyncClearInt	Sync Frame Clear Interrupt Register
5210 5494h	rCan_SyncStatus	Sync Frame Status Configuration Register
5210 5498h	rCan_SyncClearSetRunStop	Sync Frame Generation Register
5210 54A0h	rCan_SyncPassiveError	Sync Passive Error Detection Register

6.4 Register Description

6.4.1 rCan_MOD — Configuration Mode Register

Used to configure the behavior of the CAN controller in the following modes:

- Sleep Mode, Acceptance Filter Mode, Self Test Mode, Listen Only Mode, Reset Mode

Address: 5210 4000h (CAN1)
5210 5000h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bCan_SM	bCan_AFM	bCan_STM	bCan_LOM	bCan_RM
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 6.3 rCan_MOD Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Read as 0.	R
b4	bCan_SM	Sleep Mode Can only be written in Operation Mode. See **Section 6.5.6, Sleep Mode**. When there is no bus activity and no interrupts are pending, the CAN controller can be put into Sleep Mode. This function is implemented for firmware compatibility and has no impact on power saving. 1: Sleep Mode The controller enters its Sleep Mode provided no CAN interrupt is pending and there is no bus activity. If there is bus activity or an interrupt is pending, the Wake-Up procedure is executed. 0: Wake-up (normal operation) If sleeping, the CAN controller wakes up. [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	R/W
b3	bCan_AFM	Acceptance Filter Mode See **Section 6.5.7, Acceptance Filtering**. 1: Single Filter Receive data filtered using one 4 bytes filter. 0: Dual Filter Receive data filtered using two shorter filters.	R/W
b2	bCan_STM	Self Test Mode See **Section 6.5.5, Self Reception**. 1: Self Test enabled In this mode, a full node test is possible without any other active node on the bus using the Self Reception request command. The CAN controller will perform a successful transmission, even if no acknowledge is received. 0: Normal operation An acknowledge is required for successful transmission.	R/W

Table 6.3 rCan_MOD Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b1	bCan_LOM	Listen Only Mode See **Section 6.5.5, Self Reception**. 1: Listen Only enabled In this mode, the CAN controller does not send an acknowledge to the CAN bus, even when a message is received successfully. 0: Normal operation The error counters are stopped at the current value.	R/W
b0	bCan_RM	Reset Mode See **Section 6.5.14, Reset Mode**. 1: Reset Mode selected Any message currently being transmitted or received is aborted and Reset Mode is entered. 0: Normal operation The controller returns to Operating Mode on the “1-to-0” transition of this bit. [Condition of “Set 1”] • Switch to “Bus Off” (Set “1” in bCan_BS)	R/W

6.4.2 rCan_CMR — Command Register

Setting one or more bits within the Command Register initiates an action within the transfer layer of the CAN controller.

The potential action on CAN controller are:

- Self Reception Request
- Clear Data Overrun
- Release Receive Buffer
- Abort Transmission
- Transmission Request

CAUTION

- This register is write only. When read, all bits return “0”.
- Setting the command bits bCan_AT and bCan_TR simultaneously results in a single-shot transmission of the transmit message without re-transmission in the event of an error or loss of arbitration.
- Setting the command bits bCan_AT and bCan_SRR simultaneously results in a single-shot transmission of the transmit message using the Self Reception feature, again without re-transmission in the event of an error or arbitration loss.
- If bCan_SRR and bCan_TR are set simultaneously, the bCan_SRR bit is ignored.
- A Transmission Request made in a previous command cannot be cancelled by setting the bCan_TR bit (Transmission Request) to “0”. The requested transmission can only be cancelled by setting the bCan_AT bit (Abort Transmission) to “1”.

Address: 5210 4004h (CAN1)
5210 5004h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bCan_SRR	bCan_CDO	bCan_RRB	bCan_AT	bCan_TR
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.4 rCan_CMR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Read as 0.	R
b4	bCan_SRR	Self Reception Request 1: Set when a message is to be transmitted and received simultaneously [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	W

Table 6.4 rCan_CMR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3	bCan_CDO	Clear Data Overrun 1: Set to clear the data overrun condition signaled by the bCan_DOS bit (Data Overrun Status). No further Data Overrun Interrupt will be generated while the bCan_DOS bit (Data Overrun Status) remains set [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	W
b2	bCan_RRB	Release Receive Buffer 1: Set to release the “Receive Buffer” [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	W
b1	bCan_AT	Abort Transmission 1: Set to cancel the next transmission request, provided this is not already in progress [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	W
b0	bCan_TR	Transmission Request 1: Set when a message is to be transmitted [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	W

6.4.3 rCan_SR — Controller Status Register

CAUTION

- If both the Receive Status (bCan_RS) and the Transmit Status (bCan_TS) bits are “0”, the CAN bus is idle.
- If both bits (bCan_RS and bCan_TS) are “1”, the controller is waiting to become idle again.
- After hardware reset, idle state is entered once the Bus Free sequence (11 consecutive recessive bits) has been detected.
- After a “Bus Off” event, 128 Bus Free sequences must be received before idle state is entered.

Address: 5210 4008h (CAN1)
5210 5008h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_BS	bCan_ES	bCan_TS	bCan_RS	bCan_TCS	bCan_TBS	bCan_DOS	bCan_RBS
Value after reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0

Table 6.5 rCan_SR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7	bCan_BS	Bus Status 1: The CAN controller is in “Bus Off” state and is not involved in bus activities 0: The CAN controller is involved in bus activities	R
b6	bCan_ES	Error Status 1: At least one of the error counters has reached or exceeded the CPU warning limit defined by the Error Warning Limit register (bCan_EWLR bit in rCan_EWLR register) 0: Both error counters are below the warning limit	R
b5	bCan_TS	Transmit Status 1: The CAN controller is in the process of transmitting a message 0: No message is being transmitted [Condition of “Set 1”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	R
b4	bCan_RS	Receive Status 1: The CAN controller is in the process of receiving a message 0: Nothing is currently being received [Condition of “Set 1”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	R
b3	bCan_TCS	Transmission Complete Status 1: The last requested transmission has been successfully completed 0: The last requested transmission has not been completed yet	R

Table 6.5 rCan_SR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b2	bCan_TBS	Transmit Buffer Status 1: Transmit Buffer Released The CPU may write a message to the “Transmit Buffer”. 0: Transmit Buffer Locked The CPU cannot access the “Transmit Buffer” because a message is either waiting for transmission or is in the process of being transmitted. [Condition of “Set 1”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	R
b1	bCan_DOS	Data Overrun Status 1: Data Overrun A message has been lost because there was not enough space for that message in the Receive FIFO. 0: No data overrun has occurred since the last Clear Data Overrun command was given. (bCan_CDO bit in rCan_CMR register). The overrun condition is only indicated if the entire message was received. No overrun condition is shown if the message did not complete due to an error. [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	R
b0	bCan_RBS	Receive Buffer Status 1: Receive Buffer Not Empty One or more complete messages are available to be read from the “Receive FIFO” via the “Receive Buffer”. 0: Receive Buffer Empty No message currently available to be read. [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	R

6.4.4 rCan_IR — Interrupt Register

CAUTION

The Interrupt Register is read-only. Also, after the register has been read by the CPU, all interrupt bits except the Receive Interrupt bit are reset.

Address: 5210 400Ch (CAN1)
5210 500Ch (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_BEI	bCan_ALI	bCan_EPI	bCan_WUI	bCan_DOI	bCan_EI	bCan_TI	bCan_RI
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.6 rCan_IR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7	bCan_BEI	Bus Error Interrupt 1: Set when the CAN controller detects an error on the CAN bus, provided the bCan_BEIE bit is set within the Interrupt Enable Register (rCan_IER) 0: No interrupt [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS) • Read of this register	R
b6	bCan_ALI	Arbitration Lost Interrupt 1: Set when the CAN controller loses arbitration and becomes a receiver, provided the bCan_ALIE bit is set within the Interrupt Enable Register (rCan_IER) 0: No interrupt [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS) • Read of this register	R
b5	bCan_EPI	Error Passive Interrupt 1: Set when the CAN controller re-enters error active state after being in error passive state or when at least one error counter exceeds the protocol-defined level of 127, provided the bCan_EPIE bit is set within the Interrupt Enable Register (rCan_IER) 0: No interrupt [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set "1" in bCan_BS) • Read of this register	R

Table 6.6 rCan_IR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b4	bCan_WUI	Wake-Up Interrupt 1: Set when bus activity is detected while the CAN controller is sleeping, provided the bCan_WUIE bit is set within the Interrupt Enable Register (rCan_IER). A wake-up interrupt is also generated if the CPU tries to set the Sleep Mode (bCan_SM) bit while the CAN controller is involved in bus activities or a CAN interrupt is pending 0: No interrupt [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS) • Read of this register	R
b3	bCan_DOI	Data Overrun Interrupt 1: Set on a “0-to-1” transition of the Data Overrun Status bit (bCan_DOS), provided the bCan_DOIE bit is set within the Interrupt Enable Register (rCan_IER) 0: No interrupt [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS) • Read of this register	R
b2	bCan_EI	Error Warning Interrupt 1: Set on every change (set or clear) of either the Bus Status or Error Status bits (bCan_BS, bCan_ES), provided the bCan_EIE bit is set within the Interrupt Enable (rCan_IER) register 0: No interrupt [Condition of “Cleared to 0”] • Read of this register ***Note*)** If the Reset Mode was entered due to a “Bus Off” condition, the Error Warning Interrupt will be set (if enabled).	R
b1	bCan_TI	Transmit Interrupt 1: Set whenever the Transmit Buffer Status (bCan_TBS) changes from “0-to-1” (released), provided the bCan_TIE bit is set within the Interrupt Enable Register (rCan_IER) 0: No interrupt [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS) • Read of this register	R
b0	bCan_RI	Receive Interrupt 1: Set whenever the Receive Buffer contains one or more messages, provided the bCan_RIE bit is set within the Interrupt Enable Register (rCan_IER). Cleared when the release Receive Buffer command (bCan_RRB) is issued, provided there is no further data to read in the Receive Buffer 0: No interrupt [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS) The bCan_RI bit (when enabled) mirrors the Receive Buffer Status bit (bCan_RBS), which is why it is not automatically cleared when the Interrupt Register is read.	R

6.4.5 rCan_IER — Interrupt Event Register

Address: 5210 4010h (CAN1)
5210 5010h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_BEIE	bCan_ALIE	bCan_EPIE	bCan_WUIE	bCan_DOIE	bCan_EIE	bCan_TIE	bCan_RIE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.7 rCan_IER Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7	bCan_BEIE	Bus Error Interrupt Enable 1: Enabled, an interrupt will be generated when a bus error has been detected 0: Masked, the interrupt is disabled	R/W
b6	bCan_ALIE	Arbitration Lost Interrupt Enable 1: Enabled, an interrupt will be generated when the CAN controller loses arbitration 0: Masked, the interrupt is disabled	R/W
b5	bCan_EPIE	Error Passive Interrupt Enable 1: Enabled, an interrupt will be generated when the error status of the CAN controller changes from error active to error passive or vice versa 0: Masked, the interrupt is disabled	R/W
b4	bCan_WUIE	Wake-Up Interrupt Enable 1: Enabled, an interrupt will be generated when the sleeping CAN controller wakes up 0: Masked, the interrupt is disabled	R/W
b3	bCan_DOIE	Data Overrun Interrupt Enable 1: Enabled, an interrupt will be generated when the Data Overrun Status bit (bCan_DOS) is set 0: Masked, the interrupt is disabled	R/W
b2	bCan_EIE	Error Warning Interrupt Enable 1: Enabled, an interrupt will be generated when the bus status or error status bits (bCan_BS, bCan_ES) change 0: Masked, the interrupt is disabled	R/W
b1	bCan_TIE	Transmit Interrupt Enable 1: Enabled, an interrupt will be generated when a message has been successfully transmitted or the “Transmit Buffer” is accessible again 0: Masked, the interrupt is disabled	R/W
b0	bCan_RIE	Receive Interrupt Enable 1: Enabled, an interrupt will be generated when the Receive Buffer Status (bCan_RBS) goes from “0” to “1” (“full”) 0: Masked, the interrupt is disabled The Receive Interrupt Enable bit has direct influence on the Receive Interrupt bit and the external interrupt output CAN_Int. If bCan_RIE is cleared, CAN_Int will immediately become inactive (low) if no other interrupt is pending.	R/W

6.4.6 rCan_BTR0 — Bus Timing Register 0

CAUTION

Bus Timing Register 0 defines the values of the Synchronization Jump Width (SJW) and the Baud Rate Prescaler (BRP). This register can only be written in Reset Mode, in Operating Mode, it is read only.

Address: 5210 4018h (CAN1)
5210 5018h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_SJW		bCan_BRP					
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.8 rCan_BTR0 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7, b6	bCan_SJW	Synchronization Jump Size The Synchronization Jump Width defines the maximum number of time quanta by which a bit period may be shortened or lengthened in attempting to re-synchronize on the relevant signal edge (recessive to dominant) of the current transmission. SJW = Tq × (2 × bCan_SJW[1] + bCan_SJW[0] + 1)	R/W
b5 to b0	bCan_BRP	Baud Rate Prescaler The Baud Rate Prescaler defines the “time quantum” Tq of the clock for CAN as a multiple of the CAN_HCLK period. The time quantum of the clock for CAN is given by: Tq = 2 × Tperiod (CAN_HCLK) × (32 × bCan_BRP[5] + 16 × bCan_BRP[4] + 8 × bCan_BRP[3] + 4 × bCan_BRP[2] + 2 × bCan_BRP[1] + bCan_BRP[0] + 1) Where Tperiod(CAN_HCLK) = time period of the CAN_HCLK See **Section 6.5.13, Bit Period and Bus Timing Parameters**.	R/W

6.4.7 rCan_BTR1 — Bus Timing Register 1

CAUTION

Bus Timing Register 1 defines the length of the bit period, the location of the sample point and the number of samples to be taken at each sample point.
This register can only be written in Reset Mode, in Operating Mode, it is read only.

Address: 5210 401Ch (CAN1)
5210 501Ch (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_SAM	bCan_TSEG2			bCan_TSEG1			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.9 rCan_BTR1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7	bCan_SAM	Sample Mode 1: The bus will be sampled three times. This is recommended for low/medium speed buses (class A or B). 0: The bus will be sampled once. This is recommended for high speed buses (SAE class C).	R/W
b6 to b4	bCan_TSEG2	bCan_TSEG1 and bCan_TSEG2 define the length of the bit period by giving the number of time quanta up to and after the point(s) at which incoming data will be sampled. See **Section 6.5.13, Bit Period and Bus Timing Parameters**. See **Figure 6.3, CAN: General Structure of a Bit Period, Tsyncseg, Tseg1, Tseg2**. Tsyncseg = 1 × Tq Tseg2 = Tq × (4 × bCan_TSEG2[2] + 2 × bCan_TSEG2[1] + bCan_TSEG2[0] + 1)	R/W
b3 to b0	bCan_TSEG1	bCan_TSEG1 and bCan_TSEG2 define the length of the bit period by giving the number of time quanta up to and after the point(s) at which incoming data will be sampled. See **Section 6.5.13, Bit Period and Bus Timing Parameters**. See **Figure 6.3, CAN: General Structure of a Bit Period, Tsyncseg, Tseg1, Tseg2**.. Tsyncseg = 1 × Tq Tseg1 = Tq × (8 × bCan_TSEG1[3] + 4 × bCan_TSEG1[2] + 2 × bCan_TSEG1[1] + bCan_TSEG1[0] + 1)	R/W

6.4.8 rCan_OCR — Output Control Register

CAUTION

This register can only be written in Reset Mode, in Operating Mode, it is read only.

Address: 5210 4020h (CAN1)
5210 5020h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bCan_OCMODE	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.10 rCan_OCR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved	Read as 0.	R
b1, b0	bCan_OCMODE	The Output Control Register allows the selection of output driver configurations. Only one mode “Normal Output” is implemented. The additional driver configurations available in the SJA1000 through this register are not supported by the CAN controller. In Normal Output Mode: The bit sequence is transmitted to CAN_TXD. 2’b00: Reserved 2’b01: Reserved 2’b10: Normal mode 2’b11: Reserved	R/W

6.4.9 rCan_ALC — Arbitration Lost Capture Register

Address: 5210 402Ch (CAN1)
5210 502Ch (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bCan_ALC				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.11 rCan_ALC Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Read as 0.	R
b4 to b0	bCan_ALC	When bus arbitration lost, an Arbitration Lost Interrupt (bCan_ALI) is generated (to enable this interrupt, set “1” in bCan_ALIE bit) and the current bit position of the frame is captured into this register. The contents of this register are then maintained until the register has been read by the CPU. The capture mechanism is then activated again. 5’b0_0000: Arbitration lost in 1st bit of identifier (ID.28) 5’b0_0001: Arbitration lost in 2nd bit of identifier (ID.27) … … … 5’b0_1001: Arbitration lost in 10th bit of identifier (ID.19) 5’b0_1010: Arbitration lost in 11th bit of identifier (ID.18) 5’b0_1011: Arbitration lost in SRTR bit, (RTR bit in Standard Frame Format messages) 5’b0_1100: Arbitration lost in IDE bit For extended Frame Format messages only 5’b0_1101: Arbitration lost in 12th bit of identifier (ID.17) 5’b0_1110: Arbitration lost in 13th bit of identifier (ID.16) 5’b0_1111: Arbitration lost in 14th bit of identifier (ID.15) 5’b1_0000: Arbitration lost in 15th bit of identifier (ID.14) 5’b1_0001: Arbitration lost in 16th bit of identifier (ID.13) … … … 5’b1_1101: Arbitration lost in 28th bit of identifier (ID.1) 5’b1_1110: Arbitration lost in 29th bit of identifier (ID.0) 5’b1_1111: Arbitration lost in RTR bit. See **Section 6.5.9, Bus Arbitration**.	R

6.4.10 rCan_ECC — Error Code Capture Register

Address: 5210 4030h (CAN1)
5210 5030h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_ECC_Code		bCan_ECC_Direction	bCan_ECC_Segment				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.12 rCan_ECC Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7, b6	bCan_ECC_Code	When a bus error occurs, a Bus Error Interrupt (bCan_BEI) is generated (to enable this interrupt, set “1” in bCan_BEIE bit) and the current bit position of the frame is captured into this Error Code Capture Register. The contents of this register are then maintained until the register has been read by CPU. The capture mechanism is then activated again. Error Code: 2’b00: Bit error 2’b01: Form error 2’b10: Stuff error 2’b11: Some other type of error See **Section 6.5.10, Error Handling**.	R
b5	bCan_ECC_Direction	Error Direction: 1: Error occurred during reception 0: Error occurred during transmission See **Section 6.5.10, Error Handling**.	R

Table 6.12 rCan_ECC Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b4 to b0	bCan_ECC_Segment	Error Segment Code: 5'b0_0000: Not used 5'b0_0001: Not used 5'b0_0010: ID.21 to ID.28 5'b0_0011: Start of frame 5'b0_0100: SRTR bit 5'b0_0101: IDE bit 5'b0_0110: ID.18 to ID.20 5'b0_0111: ID.13 to ID.17 5'b0_1000: CRC Sequence 5'b0_1001: Reserved bit0 5'b0_1010: Data Field 5'b0_1011: Data Length Code 5'b0_1100: RTR bit 5'b0_1101: Reserved bit1 5'b0_1110: ID.0 to ID.4 5'b0_1111: ID.5 to ID.12 5'b1_0000: Not used 5'b1_0001: Active Error Flag 5'b1_0010: Intermission 5'b1_0011: Tolerate Dominant bits 5'b1_0100: Not Used 5'b1_0101: Not used 5'b1_0110: Passive Error Flag 5'b1_0111: Error Delimiter 5'b1_1000: CRC Delimiter 5'b1_1001: Acknowledge 5'b1_1010: End of frame 5'b1_1011: Acknowledge Delimiter 5'b1_1100: Overload Flag 5'b1_1101: Not used 5'b1_1110: Not used 5'b1_1111: Not used See **Section 6.5.10, Error Handling**.	R

6.4.11 rCan_EWLR — Error Warning Limit Register

CAUTION

This register can only be written in Reset Mode, in Operating Mode, it is read only.

Address: 5210 4034h (CAN1)
5210 5034h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_EWLR							
Value after reset	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Table 6.13 rCan_EWLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_EWLR	Error Warning Limit The number of errors in either reception or transmission at which a warning should be generated. When either the Transmit Error Counter (bCan_TXERR bit) or the Receive Error Counter (bCan_RXERR bit) passes this value, the Error Status (bCan_ES) bit in the Status Register (rCan_SR) is set and an Error Warning Interrupt (bCan_EI) is generated (to enable this interrupt, set “1” in bCan_EIE). See **Section 6.5.10, Error Handling**. You should also note that changes made within Reset Mode are only put into effect on the return to Operating Mode.	R/W

6.4.12 rCan_RXERR — Receive Error Counter Register

CAUTION

This register can only be written in Reset Mode, in Operating Mode, it is read only.

Address: 5210 4038h (CAN1)
5210 5038h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_RXERR							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.14 rCan_RXERR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_RXERR	Receive Error Counter This counter is incremented when errors are experienced in the Receive bit stream and decremented when messages are received without error, in line with the rules given in the CAN 2.0 specification. Together with the associated Transmit Error Counter (bCan_TXERR), it provides an indication of the quality of transmission being experienced on the CAN bus. Two levels of the counter trigger specific events. • When the counter reaches the level set in the Error Warning Limit register (bCan_EWLR), an Error Warning Interrupt (bCan_EI) is generated (to enable this interrupt, set “1” in bCan_EIE) unless this has previously been triggered by the Transmit Error Counter (bCan_TXERR bit). • When the counter goes over 127, the device is put into “Error Passive” state in accordance with the CAN 2.0 specification (unless previously triggered by the Transmit Error Counter) and a last Active error is sent and an Error Passive Interrupt (bCan_EPI) is also generated (to enable this interrupt, set “1” in bCan_EPIE bit). When a “Bus Off” event occurs, the counter is automatically set to “0” (Set “1” in bCan_BS). See **Section 6.5.10, Error Handling**. You should note, however, that writing to this register has no effect when the CAN controller is in “Bus Off” state and that any change made within Reset Mode will in any case only come into effect on return to Operating Mode. ***Note*)** If the Reset Mode was entered due to a “Bus Off” condition (Set “1” in bCan_BS), the Receive Error Counter will be cleared and the Transmit Error Counter will be initialized to 127 and used to count-down the CAN-defined bus-off recovery time consisting of 128 occurrences of 11 consecutive recessive bits.	R/W

6.4.13 rCan_TXERR — Transmit Error Counter Register

CAUTION

This register can only be written in Reset Mode, in Operating Mode, it is read only.

Address: 5210 403Ch (CAN1)
5210 503Ch (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_TXERR							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.15 rCan_TXERR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_TXERR	Transmit Error Counter This counter is incremented when Transmission errors are experienced and decremented when messages are transmitted without error, in line with the rules given in the CAN 2.0 specification. Together with the associated Receive Error Counter (bCan_RXERR), it provides an indication of the quality of transmission being experienced on the CAN bus. Three levels of the counter trigger specific events. • When the counter reaches the level set in the Error Warning Limit register (bCan_EWLR), an Error Warning Interrupt (bCan_EI) is generated (to enable this interrupt, set “1” in bCan_EIE) unless this has previously been triggered by the Receive Error Counter (bCan_RXERR bit). • When the counter goes over 127, the device is put into “Error Passive” state in accordance with the CAN 2.0 specification (unless previously triggered by the Receive Error Counter), a last Active error is sent and an Error Passive Interrupt (bCan_EPI) is generated (to enable this interrupt, set “1” in bCan_EPIE bit) • When the counter goes over 255, the device is put into “Bus Off” state in accordance with the CAN 2.0 specification and is automatically put into Reset mode (except during start-up when there is only one node on the CAN bus). An Error Warning Interrupt (bCan_EI) is also generated (to enable this interrupt, set “1” in bCan_EPIE bit). After a “Bus Off” event, the register is initialized to 127 in order to count the minimum protocol-defined time before the CAN controller can take part in further transmission on the CAN bus (128 occurrences of the “Bus-Free” sequence of 11 consecutive recessive bits). Reading the Transmit Error Counter during this time will give the status of the “Bus Off” recovery. See **Section 6.5.10, Error Handling**.	R/W

Table 6.15 rCan_TXERR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
		***Note*)** • If the Reset Mode is re-entered before the “Bus Off” recovery has been completed (bCan_TXERR greater than 0), “Bus Off” will stay active with bCan_TXERR frozen until the CAN controller is taken back into Operating Mode. • While in “Bus Off” state, writing a value in the range from 0 to 254 to bCan_TXERR clears the “Bus Off” flag. The CAN controller will then wait for just one Bus Free sequence after the Reset Mode has been cleared. • Writing 255 to bCan_TXERR in Reset Mode initiates a CPU-driven “Bus Off” event. No error or bus status change happens in response to the new bCan_TXERR value until the CAN controller is taken back into Operating Mode when a “Bus Off” event will be performed exactly as if it had been forced by a bus error. This means Reset Mode is entered again, the Transmit Error Counter is initialized to 127, the Receive Error counter is cleared and the relevant Status and Interrupt register bits are set. Clearing Reset Mode now will perform the protocol-defined Bus Off recovery sequence (waiting for 128 occurrences of the Bus Free signal). • If the Reset Mode was entered due to a “Bus Off” condition (Set “1” in bCan_BS), the Receive Error Counter will be cleared and the Transmit Error Counter will be initialized to 127 and used to count-down the CAN-defined bus-off recovery time consisting of 128 occurrences of 11 consecutive recessive bits.	

6.4.14 rCan_WrTransmitBuffer — Write Transmit Buffer Register

CAUTION

This register is write only and it can only be written in Operating Mode.

Address: 5210 4040h - 5210 4070h (CAN1)
5210 5040h - 5210 5070h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_WrTransmitBuffer							
Value after reset	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X

Table 6.16 rCan_WrTransmitBuffer Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_WrTransmitBuffer	Write Transmit buffer Allows the transmission of frame data to the CAN Bus. The Transmit Buffer has a length of 13 bytes. It accommodates one transmit message of up to eight data bytes. See **Section 6.5.3, Transmission**. Write-only access to the Transmit Buffer is provided in Operating Mode using CAN offsets 12'h040 - 12'h070. Read access to the Transmit Buffer is possible using CAN offsets 12'h180 - 12'h1B0. The global layout of the Transmit Buffer is described in **Section 6.5.11, Transmit Buffer Layout**. It is important to distinguish between "Standard Frame Format" (SFF) messages and the "Extended Frame Format" (EFF) messages.	W

6.4.15 rCan_RdReceiveBuffer — Read Receive Buffer Register

Address: 5210 4040h - 5210 4070h (CAN1)
5210 5040h - 5210 5070h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_RdReceiveBuffer							
Value after reset	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X

Table 6.17 rCan_RdReceiveBuffer Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_RdReceiveBuffer	Read Receive buffer The Receive Buffer provides the window through which the CPU accesses the Receive FIFO. Like the Transmit Buffer, the Receive Buffer has a length of 13 bytes (enough to accommodate one Receive message of up to eight data bytes). See **Section 6.5.4, Reception**. Read-only access to the Receive Buffer is provided in Operating Mode using CAN offsets 12'h040 - 12'h070. The global layout of the Receive Buffer is described in **Section 6.5.12, Receive Buffer Layout**. It is important to distinguish between "Standard Frame Format" (SFF) messages and the "Extended Frame Format" (EFF) messages.	R

6.4.16 rCan_ACR[n] — Acceptance Code Filter [n] Register (n = 0..3)

CAUTION

This register can only be written in Reset Mode.

Address: 5210 4040h + 4h × n (CAN1)
5210 5040h + 4h × n (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_ACR							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.18 rCan_ACR[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_ACR	Acceptance Code Filter These 8-bit registers hold the bit patterns used by the Acceptance Filter in filtering received data in conjunction with the masks provided by rCan_AMR[n]. The way in which these bit patterns are applied depends on whether a single filter or dual filters are being used and on whether the data is in Standard Frame Format (SFF) or Extended Frame Format (EFF). See **Section 6.5.7, Acceptance Filtering**.	R/W

6.4.17 rCan_AMR[n] — Acceptance Mask Filter [n] Register (n = 0..3)

CAUTION

This register can only be written in Reset Mode.

Address: 5210 4050h + 4h × n (CAN1)
5210 5050h + 4h × n (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_AMR							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.19 rCan_AMR[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_AMR	Acceptance Mask filter These 8-bit registers hold the mask patterns applied by the Acceptance Filter in filtering the data received. “0”s in these registers identify the bits of the incoming data bytes that are required to match the bit values in the corresponding Acceptance Code Registers rCan_ACR[n]. “1”s mark individual bits as “don’t care”. The bits of the incoming data picked out by these masks depends on whether a single filter or dual filters are being used and on whether the data is in Standard Frame Format (SFF) or Extended Frame Format (EFF). See **Section 6.5.7, Acceptance Filtering**.	R/W

6.4.18 rCan_RMC — Receive Message Counter Register

Address: 5210 4074h (CAN1)
5210 5074h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bCan_RMC				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.20 rCan_RMC Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Read as 0.	R
b4 to b0	bCan_RMC	Receive Message Counter The Receive Message Counter register holds the number of messages currently available in the Receive FIFO. It is automatically incremented by each Receive event and decremented by each Release Receive Buffer command. See **Section 6.5.4, Reception**. [Condition of “Cleared to 0”] • Software Reset (Set “1” in bCan_RM) • Switch to “Bus Off” (Set “1” in bCan_BS)	R

6.4.19 rCan_RBSA — Receive Buffer Start Address Register

CAUTION

This register can only be written in Reset Mode.

Address: 5210 4078h (CAN1)
5210 5078h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	bCan_RBSA					
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.21 rCan_RBSA Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b6	Reserved	Read as 0.	R
b5 to b0	bCan_RBSA	Receive Buffer Start Address The Receive Buffer Start Address register holds the current location of the RX FIFO Read Pointer within the 64-byte Receive FIFO as a value between 0 and 63. Location 0 maps to CAN offset 12'h080. Location 63 maps to CAN offset 12'h17C. See **Section 6.5.4, Reception**. This register is left unchanged by a software reset (which also does not change the FIFO contents). However, a software reset sets the RX FIFO Write Pointer to the value of the RX FIFO Read Pointer, so the data accessed by the Receive Buffer following a software reset will be overwritten by the next message to be held in the Receive FIFO.	R

6.4.20 rCan_ReceiveFifo — Receive FIFO Register

CAUTION

This register can only be written in Reset Mode.

Address: 5210 4080h - 5210 417Ch (CAN1)
5210 5080h - 5210 517Ch (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_ReceiveFifo							
Value after reset	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X

Table 6.22 rCan_ReceiveFifo Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_ReceiveFifo	Receive FIFO, 64Bytes Data received by the controller is first filtered by the “Acceptance Filter” then passed to the “Receive FIFO”. The “Receive FIFO” is 64 bytes depth, allowing space for up to five full “Extended Frame Format” messages, and is used in a circular fashion The Receive Buffer Start Address register (rCan_RBSA) holds the current location of the RX FIFO Read Pointer within the 64-byte Receive FIFO as a value between 0 and 63. Location 0 maps to CAN offset 12’h080 Location 63 maps to CAN offset 12’h17C. See **Section 6.5.4, Reception**.	R/W

6.4.21 rCan_RdTransmitBuffer — Read Transmit Buffer Register

Address: 5210 4180h - 5210 41B0h (CAN1)
5210 5180h - 5210 51B0h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_RdTransmitBuffer							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.23 rCan_RdTransmitBuffer Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_RdTransmitBuffer	Read Transmit buffer Transmit buffer can be read back. See **Section 6.5.3, Transmission**. Write-only access to the Transmit Buffer is provided in Operating Mode using CAN offsets 12'h040 - 12'h070. Read access to the Transmit Buffer is possible using CAN offsets 12'h180 - 12'h1B0. The global layout of the Transmit Buffer is described in **Section 6.5.11, Transmit Buffer Layout**.	R

6.4.22 rCan_SyncTransmitBuffer — Sync Frame Transmit Buffer Register

Address: 5210 4440h - 5210 4470h (CAN1)
5210 5440h - 5210 5470h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bCan_SyncTransmitBuffer							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.24 rCan_SyncTransmitBuffer Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bCan_SyncTransmitBuffer	Sync Frame Transmit Buffer This buffer is written by CPU with the contents of “Sync frame”. It allows the transmission of “Sync frame” on the CAN Bus in a time window dedicated. This time window dedicated for “Sync frame” allows the CAN controller to perform the following actions: • Send the last frame written by the CPU in rCan_WrTransmitBuffer • Manage the potential retries on this frame (arbitration loss) • Send a “Sync frame” initialized in rCan_SyncTransmitBuffer This time window is managed by bCan_SyncMaskFrame bit in rCan_SyncStatus register. The size of time window is controlled by a timer bCan_SyncMaskFrameTime bit in rCan_SyncPeriod register. See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. The Transmit Buffer has a length of 13 bytes. It accommodates one transmit message of up to eight data bytes. The global layout of the Transmit Buffer is described in **Section 6.5.11, Transmit Buffer Layout**.	R/W

6.4.23 rCan_SyncPeriod — Time Window Sync Frame Transmission Register

Time parameters of “Sync frame” in bit period unit, allows the control of:

- Period of “Sync frame”
- Time window dedicated for the transmission of “Sync frame” on the CAN Bus.

Address: 5210 4480h (CAN1)
5210 5480h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	bCan_SyncMaskFrameTime										—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bCan_SyncPeriod															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.25 rCan_SyncPeriod Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31, b30	Reserved	Read as 0.	R
b29 to b20	bCan_SyncMaskFrameTime	Time window in bit period unit dedicated for the transmission of “Sync frame” on the CAN Bus. • This timer configures the time between: – The rising edge of bCan_SyncMaskFrame bit. – Next emission of “Sync frame” • This time window dedicated for “Sync frame” allows the CAN controller to perform the following actions: – Send the last frame written by the CPU in rCan_WrTransmitBuffer – Manage the potential retries on this frame (arbitration loss) – Send a “Sync frame” initialized in rCan_SyncTransmitBuffer • To calculate the size of this time window, the user must take in account: – The time necessary to send on CAN bus the potential last frame contained in rCan_WrTransmitBuffer without retry. – The time necessary to manage 2 or 3 retries on CAN bus (arbitration loss). See **Section 6.5.9, Bus Arbitration**. – The time necessary by CAN controller to copy a “Sync frame” initialized in rCan_SyncTransmitBuffer to rCan_WrTransmitBuffer registers and to send a Transmission Request (bCan_TR or bCan_SRR) in rCan_CMR register. (Take 8 bits period unit). See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**. See **Figure 6.8, CANopen: Time Window dedicated for Emission of Periodic Sync Frame**. 10’h0: 1 bit period 10’h1: 2 bits periods 10’h2: 3 bits periods ... ... 10’h3FF: 1024 bits periods	R/W

Table 6.25 rCan_SyncPeriod Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
		• We always must have: – bCan_SyncPeriod greater than bCan_SyncMaskFrameTime greater than 8. The bit period unit is defined by rCan_BTR0 and rCan_BTR1 registers. See **Figure 6.3, CAN: General Structure of a Bit Period, Tsyncseg, Tseg1, Tseg2**.	
b19 to b16	Reserved	Read as 0.	R
b15 to b0	bCan_SyncPeriod	Period of “Sync frame” in bit period unit. See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. 16’h0: 1 bit period 16’h1: 2 bits periods 16’h2: 3 bits periods ... ... 16’hFFFF: 65536 bits periods The bit period unit is defined by rCan_BTR0 and rCan_BTR1 registers. See **Figure 6.3, CAN: General Structure of a Bit Period, Tsyncseg, Tseg1, Tseg2**.	R/W

6.4.24 rCan_SyncStatusInt — Sync Frame Interrupt Status Register

The Interrupt register allows the source of an interrupt to be identified. When one or more bits of this register are set, the CAN controller sends an interrupt to the CPU. This register is dedicated of “Sync frame” management.

- Status of overrun “Sync frame” interrupt
- Status of “Sync frame” interrupt

Address: 5210 4488h (CAN1)
5210 5488h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bCan_OverrunSyncFrameInt	bCan_SendSyncFrameInt
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.26 rCan_SyncStatusInt Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved	Read as 0.	R
b1	bCan_OverrunSyncFrameInt	Status of overrun “Sync frame” interrupt • An interrupt is generated when: – The time window defined by bCan_SyncMaskFrameTime and bCan_SyncPeriod bits are not correctly initialized. – Or many repetition frames (arbitration loss) on CAN Bus. See **Section 6.5.9, Bus Arbitration**. • Normally case: – No overrun “Sync frame” interrupt. – The emission of “Sync frame” is synchronous with time base bCan_SyncPeriod • Overrun case: – An Overrun “Sync frame” interrupt is generated. – The emission of “Sync frame” is not synchronous with time base bCan_SyncPeriod. – The emission of “Sync frame” is delayed to have enough time to close the emission of current frame contained in rCan_SyncTransmitBuffer registers with all repetition necessary on CAN Bus. 1: If bCan_SyncRunStop bit is set to “1”, (emission of “Sync frame” is enabled) This bit is set when: • As soon as the “Sync frame” is sent. • And if there not enough time to send “Sync frame” in time window controlled by bCan_SyncMaskFrame and bCan_SyncMaskFrameTime bits. 0: No interrupt generated Cleared to “0” by a write “1” in bCan_OverrunSyncFrameClearInt bit. See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**. See **Figure 6.8, CANopen: Time Window dedicated for Emission of Periodic Sync Frame**.	R

Table 6.26 rCan_SyncStatusInt Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b0	bCan_SendSyncFrameInt	Status of “Sync frame” interrupt 1: If bCan_SyncRunStop bit is set to “1”, emission of “Sync frame” is enabled. This bit is set as soon as the “Sync frame” is sent. 0: No interrupt generated Cleared to “0” by a write “1” in bCan_SendSyncFrameClearInt bit. See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**.	R

6.4.25 rCan_SyncMaskInt — Sync Frame Mask Interrupt Register

This register is used to select the events that cause an interrupt to be generated. This register is dedicated of “Sync frame” management.

Address: 5210 448Ch (CAN1)
5210 548Ch (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bCan_OverrunSyncFrameMaskInt	bCan_SendSyncFrameMaskInt
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.27 rCan_SyncMaskInt Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved	Read as 0.	R
b1	bCan_OverrunSyncFrameMaskInt	Enable or disable overrun “Sync frame” interrupt 1: No masked, enable overrun “Sync frame” interrupt 0: Masked, disable overrun “Sync frame” interrupt See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**. See **Figure 6.8, CANopen: Time Window dedicated for Emission of Periodic Sync Frame**.	R/W
b0	bCan_SendSyncFrameMaskInt	Enable or disable of “Sync frame” interrupt 1: No masked, enable “Sync frame” interrupt 0: Masked, disable “Sync frame” interrupt See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**.	R/W

6.4.26 rCan_SyncClearInt — Sync Frame Clear Interrupt Register

Address: 5210 4490h (CAN1)
5210 5490h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bCan_OverrunSyncFrameClearInt	bCan_SendSyncFrameClearInt
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.28 rCan_SyncClearInt Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved	Read as 0.	R
b1	bCan_OverrunSyncFrameClearInt	Acknowledge Overrun “Sync frame” interrupt [Write 1] Acknowledge Overrun “Sync frame” interrupt Clear bCan_OverrunSyncFrameInt bit to “0” [Write 0] No action Read as 0. See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**. See **Figure 6.8, CANopen: Time Window dedicated for Emission of Periodic Sync Frame**.	W
b0	bCan_SendSyncFrameClearInt	Acknowledge “Sync frame” interrupt [Write 1] Acknowledge “Sync frame” interrupt Clear bCan_SendSyncFrameInt bit to “0” [Write 0] No action Read as 0. See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**.	W

6.4.27 rCan_SyncStatus — Sync Frame Status Configuration Register

Status dedicated for “Sync frame” management. Used to configure the behavior of the CAN controller in the following modes:

- Run and Stop Mode of “Sync frame” emission
- Time window status of Mask Frame

Address: 5210 4494h (CAN1)
5210 5494h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	bCan_TimerOnlyMode	bCan_TimerOnlyIfBusOff	bCan_SyncMode	bCan_SyncMaskFrame	—	—	bCan_SyncRunStop
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.29 rCan_SyncStatus Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b7	Reserved	Read as 0.	R
b6	bCan_TimerOnlyMode	0: CAN Controller is running in full mode 1: CAN Controller is running in TimerOnlyMode. No Sync Frame launched, the CAN Controller only send the interrupt of the sync frame. This field is set by hardware.	R/W
b5	bCan_TimerOnlyIfBusOff	0: Disable all the “Sync frame” system if the CAN Controller detect a “Bus off” condition 1: Switch to “TimerOnlyMode” if the CAN Controller detect a “Bus off” condition ***Caution*)** • The CAN Controller can only detect a “Bus off” condition during the 8-bit period unit before a Sync Frame launch timing. • See **Figure 6.8, CANopen: Time Window dedicated for Emission of Periodic Sync Frame**. • In order to let the system detects the bus off, it is recommended to wait for a Sync Frame interrupt before clearing the bus off status.	R/W
b4	bCan_SyncMode	Resources used to send the “Sync frame” 2 functions mode can be used: – Transmission request mode – Self Reception request mode 1: Send “Sync frame” with a command written in bCan_SRR (Self Reception Request mode) 0: Send “Sync frame” with a command written in bCan_TR (Transmission Request mode) See **Section 6.5.15, Synchronization Frame**. See **Figure 6.8, CANopen: Time Window dedicated for Emission of Periodic Sync Frame**.	R/W

Table 6.29 rCan_SyncStatus Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3	bCan_SyncMaskFrame	Time window reserved for emission of “Sync frame” Enables or disables the CPU access in rCan_WrTransmitBuffer registers. Must be polled by CPU before writing a new CAN frame in rCan_WrTransmitBuffer. Before writing the last CAN frame in rCan_WrTransmitBuffer, the CPU must read “0” in bCan_SyncMaskFrame bit. 1: Time window reserved for emission of “Sync frame” The CPU can finish writing the current CAN frame in the rCan_WrTransmitBuffer registers, but is not allowed to write a new CAN frame in the buffer. 0: Window reserved for emission of standard CAN frame All CPU accesses in rCan_WrTransmitBuffer are authorized. This bit is managed by the dedicated timer, the size of the window is programmable with bCan_SyncMaskFrameTime bit in rCan_SyncPeriod register. This time window dedicated for “Sync frame” allows the CAN controller to perform the following actions: – Send the last frame written by the CPU in rCan_WrTransmitBuffer – Manage the potential retries on this frame (arbitration loss) – Send the Sync frame initialized in rCan_SyncTransmitBuffer See **Section 6.5.15, Synchronization Frame**. See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**. See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**. See **Figure 6.8, CANopen: Time Window dedicated for Emission of Periodic Sync Frame**.	R
b2, b1	Reserved	Read as 0.	R
b0	bCan_SyncRunStop	Status of “Sync frame” emission A specific sequence allows setting or clearing this bit. 1: “Run Mode”, Enable (Run) the emission of “Sync frame” Generation of bCan_SyncMaskFrame An interrupt is generated when a “Sync frame” is sent. The configuration of “Sync frame” must be valid before going in run, initialization by CPU of following registers: - bCan_SyncMode (Transmit mode used) - rCan_SyncTransmitBuffer (Sync frame data) - bCan_SyncPeriod (Period of “Sync frame”) - bCan_SyncMaskFrameTime (Time base of MaskFrame) 0: “Stop mode”, Disable (Stop) the emission of “Sync frame” All followings bits are cleared: - bCan_SyncRunStop bit - bCan_SyncMaskFrame bit All timers dedicated for “Sync frame” management. To write at this register, a specific sequence is necessary: – Write rCan_SyncClearSetRunStop with 32’h2052_756E Switch to “Run mode” – Write rCan_SyncClearSetRunStop with 32’h5374_6F70 Switch to “Stop mode” Only these values are accepted, other values are ignored. ***Caution*)** • Before to send a “Sync frame”, the CAN controller verifies the status of bCan_BS bit (Bus Status). • If bCan_BS is read to “1" (“Bus Off” mode), the CAN controller clears bCan_SyncRunStop bit to “0”. See **Section 6.5.15, Synchronization Frame**. See **Figure 6.5, CANopen: Start and Stop of Periodic Sync Frame**. See **Figure 6.8, CANopen: Time Window dedicated for Emission of Periodic Sync Frame**.	R

6.4.28 rCan_SyncClearSetRunStop — Sync Frame Generation Register

Address: 5210 4498h (CAN1)
5210 5498h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bCan_SyncClearSetRunStop															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bCan_SyncClearSetRunStop															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.30 rCan_SyncClearSetRunStop Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bCan_SyncClearSetRunStop	Run or stop the emission of “Sync frame” Allows setting or clearing with specific sequence bCan_SyncRunStop bit in rCan_SyncStatus register. To write at this register, a specific sequence is necessary: – Write rCan_SyncClearSetRunStop with 32’h2052_756E Switch to “Run mode” Enable emission of “Sync frame” Set “1” in bCan_SyncRunStop bit Generation of bCan_SyncMaskFrame An interrupt is generated when a “Sync frame” is sent – Write rCan_SyncClearSetRunStop with 32’h5374_6F70 Switch to “Stop mode” Disable emission of “Sync frame” Clear bCan_SyncRunStop bit to “0” Clear bCan_SyncMaskFrame bit to “0” Clear all timers dedicated for “Sync frame” management Only these values are accepted, other values are ignored. Before changing the “Sync frame” configuration, the firmware must change to “Stop mode” by this register. After writing the configuration, the firmware will have to change to “Run mode” by this register for activating the “Sync frame” mechanism. See **Section 6.5.15, Synchronization Frame**. See **Figure 6.5, CANopen: Start and Stop of Periodic Sync Frame**.	W

6.4.29 rCan_SyncPassiveError — Sync Passive Error Detection Register

Address: 5210 44A0h (CAN1)
5210 54A0h (CAN2)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bCan_SyncPassiveError															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.31 rCan_SyncPassiveError Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Read as 0.	R
b15 to b0	bCan_SyncPassiveError	Enable and configure the Passive Error detection system of the CAN module: • Writing “0” into bCan_SyncPassiveError disable the “Passive Error detection system”. • Writing any other value into bCan_SyncPassiveError enable the “Passive Error detection system”; the bCan_TimerOnlyMode will be automatically set if the launch of the “Sync Frame” has been delayed enough that the number of bit period left before the next “Sync Frame” is lesser than bCan_SyncPassiveError. See **Figure 6.7, CANopen: Triggering the Passive Error Detection System**.	R/W

6.5 Operation

6.5.1 Main Features Description

Messages for transmission are placed into the “Transmit Buffer” by the CPU for transmission on CAN Bus.

Messages received by the device are first filtered by an “Acceptance Filter”, then placed into the “Receive FIFO”. The CPU accesses the “Receive FIFO” through a 13-byte window referred to as the “Receive Buffer”. The use of the “Receive Buffer” in conjunction with the “Receive FIFO” allows the CPU to process one message while other messages are being received. The “Receive FIFO” has a total length of 64 bytes and is used in circular fashion, giving it the capacity to accommodate up to five “Extended Frame Format” messages at a time.

The “Bit Timing Logic” block is responsible for the baud rate of the device and is programmable. The range of baud rates supported depends on the frequency of the internal bus clock and can readily span a wider range of baud rates than the 125 Kbps - 1 Mbps picked out by the BOSCH specification.

The interface to the CAN bus is provided by the signals CAN_TXD for transmit and CAN_RXD for receive.

6.5.2 Operation Mode

The CAN controller has two main modes of operation:

- An “Operating Mode” in which data may be transmitted and received.
- A “Reset Mode” in which bus timing parameters and message acceptance filters can be set. Reset Mode also allows the Receive and Transmit Error Counters (bCan_RXERR, bCan_TXERR) and the Error Warning Limit (bCan_EWLR) to be changed.

Reset Mode is selected either by executing a hardware reset or by setting the bCan_RM bit in the Mode Register (rCan_MOD) to “1”.

The CAN controller is returned to Operating Mode by clearing the bCan_RM bit.

When switching from “Reset mode” to “Operating mode”, CAN controller is in “Waiting to Become Idle” state with bCan_TS and bCan_RS bits equal to 1; the system is ready to be used (“Idle” state) only when those two bits switch back to 0.

The CAN controller also supports a “Listen Only Mode” and a “Self Test Mode”, selectable through the Mode Register (rCan_MOD) in either Operating Mode or Reset Mode.

- In “Listen Only Mode”, the controller is only able to receive data. No transmission is possible. The controller does not even transmit any acknowledgement of data being successfully received. It is also forced to be “Error Passive” (See **Section 6.5.10, Error Handling**).
- In “Self Test Mode”, the controller sends and receives messages using the controller’s Self Reception feature without looking for acknowledgement from any remote node.

For further information about “Reset Mode”, See **Section 6.5.14, Reset Mode**.

For further information about the “Listen Only” and “Self Test” Modes, see the description of the rCan_MOD register.

CAUTION

The controller needs to be taken out of Reset Mode to transmit or receive any data.

6.5.3 Transmission

To transmit a message, the CAN controller must be in its Operating Mode (bCan_RM = 0).

Enabling the Transmit and the Error Warning interrupts is recommended but it is usually unnecessary to enable the Arbitration Loss and Bus Error interrupts as the CAN controller will automatically try again to send the message if bus arbitration is lost or if transmission errors occur while the message is being sent.

Data to be transmitted is written to the CAN's "Transmit Buffer" in either the "Standard Frame Format (SFF)" or "Extended Frame Format (EFF)". The Transmit Buffer comprises the 13 bytes between CAN offsets 12'h040 and 12'h070, giving space for one message frame containing up to eight bytes of data.

- Write in "Transmit Buffer": See rCan_WrTransmitBuffer registers.
- Read in "Transmit Buffer": See rCan_RdTransmitBuffer registers.
- For global layout of the "Transmit Buffer": **Section 6.5.11, Transmit Buffer Layout**.

CAUTION

Before writing the data to the buffer, the Transmit Buffer Status (bCan_TBS) needs to be checked to ensure that the buffer is "released" (bCan_TBS = "1"). Any data written to the buffer when the buffer is locked (bCan_TBS = "0") is simply lost without any indication.

Transmission of the data that has been written to the "Transmit Buffer" is initiated by issuing either:

- A Transmit Request through the Command Register (by setting bCan_TR = "1")
- Or a Self Reception Request (bCan_SRR = "1") if Self Reception of the message is required (See **Section 6.5.5, Self Reception**).

The CAN controller then starts a sequence of steps in which it prepares the data for transmission over the bus (including generating the appropriate 15-bit CRC), waits for the CAN bus to become idle then starts transmission. The moment transmission starts, the Transmit Status (bCan_TS) changes to "1" and the Transmission Request bit is cleared. The bit sequence is output on CAN_TXD.

Once transmission is in progress, the CPU simply has to wait for a Transmit Interrupt (bCan_TI) to occur (to enable this interrupt, set "1" in bCan_TIE) or for the Transmit Buffer to be released (set "1" in bCan_TBS) to indicate that transmission has finished.

If bus arbitration is lost (See **Section 6.5.9, Bus Arbitration**) or if transmission errors occur (See **Section 6.5.10, Error Handling**) while the message is being sent, the CAN controller will automatically try again to send the message.

A 15-bit Cycle Redundancy Check (CRC) is sent with each frame, generated from the start of frame, arbitration, control and data fields of the frame to be sent.

Full details are given in the CAN 2.0 specification.

Transmission of a message can be aborted by issuing an Abort Transmission command through the Command Register (bCan_AT = "1") provided transmission has not yet started. It is not possible to abort a message after it has started to be transmitted.

CAUTION

- To see if the original message has been either transmitted successfully or aborted, wait for the Transmit Buffer Status bit (bCan_TBS) to be set to "1" or a Transmit Interrupt to be generated (bCan_TI) (to enable this interrupt, set bCan_TIE bit), then check the Transmission Complete Status bit (bCan_TCS). (A Transmit Interrupt is generated even if the message was aborted because the Transmit Buffer Status bit changes to "released".)

- Issuing an Abort Transmission at the same time as a Transmit Request (bCan_TR = “1”) results in a “single-shot” transmission of the current message without any retry in the event of either transmission errors or loss of bus arbitration.

6.5.4 Reception

To receive messages, the CAN controller must be in its Operating Mode (bCan_RM = 0).

Reception is initiated by the CAN controller detecting a Start of Frame.

Data received by the controller is first filtered by the “Acceptance Filter” then passed to the “Receive FIFO”. The “Acceptance Filter” only passes on those messages with identifier bits that match the ones held in the “Acceptance Filter” registers.

The moment data starts being placed in the “Receive FIFO”, the Receive Status bit (bCan_RS) in Status Register (rCan_SR) goes to “1”. Then once the data has been received, the Receive Buffer Status bit (bCan_RBS) goes to “1” and a receive interrupt (bCan_RI) is generated (to enable this interrupt, set bCan_RIE bit).

The “Receive FIFO” is 64 bytes depth, allowing space for up to five full “Extended Frame Format (EFF)” messages, and is used in a circular fashion. If there is not enough space in the FIFO for the data being received, the Data Overrun Status bit (bCan_DOS) in the Status Register (rCan_SR) is set and the data frame being received is discarded. A Data Overrun Interrupt bCan_DOI is also generated (to enable this interrupt, set bCan_DOIE bit).

The data placed in the “Receive FIFO” is read through a 13 bytes window referred to as the “Receive Buffer”. This window is located at CAN offsets 12’h040 - 12’h070, it occupies the same address space as the “Transmit Buffer”. Like the “Transmit Buffer”, it is wide enough to accommodate one message containing up to eight bytes of data.

As each message is read from the “Receive FIFO”, the host CPU needs to release the window it currently has on the FIFO by issuing the Release Receive Buffer command (Set bCan_RRB = “1”) through the Command Register (rCan_CMR). If another message is waiting to be read in the “Receive FIFO”, this will immediately become available through the “Receive Buffer”. If no message is waiting, the Receive Status bit (bCan_RS), the receive interrupt bit (bCan_RI) and the Receive Buffer Status bit (bCan_RBS) will all be cleared.

- Read in “Receive buffer”: See rCan_RdReceiveBuffer registers.
- For global layout of the “Receive Buffer”: See **Section 6.5.12, Receive Buffer Layout**.

A count of the number of messages currently available in the Receive FIFO is given by the Receive Message Counter Register (see rCan_RMC register).

The data received is written at the position indicated by the RX FIFO Write Pointer with the pointer updated as each byte is written. The position at which data is currently being read is given by the RX FIFO Read Pointer (rCan_RBSA).

When the Write Pointer coincides with the Read Pointer, there is no longer space in the FIFO for the data being received. The data frame being received is discarded, the Data Overrun Status (bCan_DOS) bit in the Status Register (rCan_SR) is set and a Data Overrun Interrupt bCan_DOI is generated (to enable this interrupt, set “1” in bCan_DOIE bit). It is up to the CPU to recover from this loss of data.

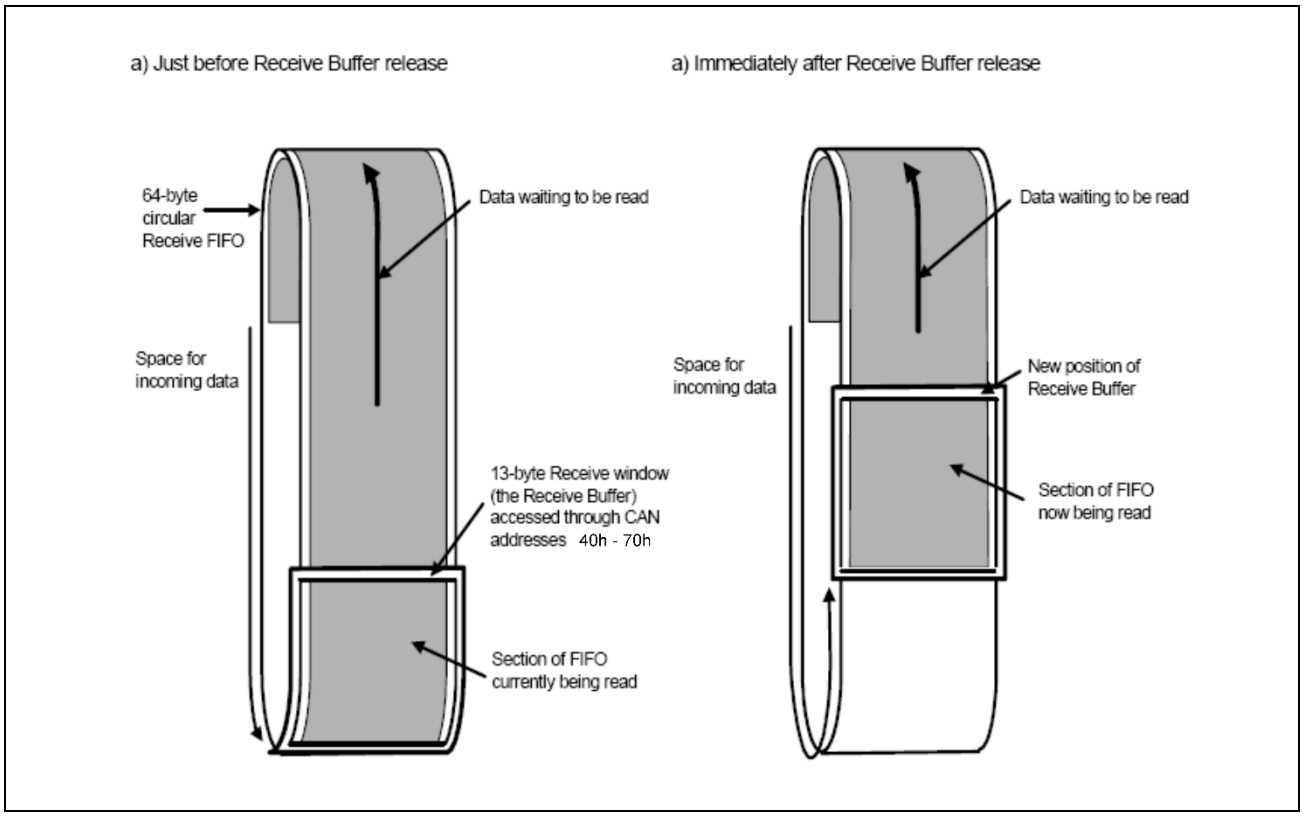

Figure 6.2 CAN: Receive Buffer Window or Receive FIFO

6.5.5 Self Reception

A feature of the CAN controller is that it allows the message it is transmitting to another CAN node to be simultaneously received by the CAN controller.

Self Reception of the current transmit message is selected by issuing a Self Reception Request through the Command Register (rCan_CMR) (set bCan_SRR bit = “1”).

The CAN controller automatically generates the Transmit and Receive Interrupts required for correct operation.

CAUTION

- If self-reception is requested at the same time as normal transmission (bCan_SRR and bCan_TR set simultaneously), the Self Reception Request is ignored.
- A special version of the Self Reception feature is offered by the CAN controller’s Self Test Mode in which a test message is both sent and received, but without requiring an acknowledgement from a remote node. This therefore allows a full test of the node containing the CAN controller without needing any other active node on the bus.
- See rCan_MOD register.

6.5.6 Sleep Mode

When there is no bus activity and no interrupts are pending, the CAN controller can be put into Sleep Mode. This function is implemented for firmware compatibility and has no impact on power saving.

Sleep Mode is selected by setting the Sleep Mode bit in the Mode Register (bCan_SM) to “1”. CAN_TXD goes high in Sleep Mode. Any of the following events will cause the CAN controller to “wake up” from Sleep Mode:

- Setting the Sleep Mode (bCan_SM) bit to “0”.
- Activity on the CAN bus input (CAN_RXD)

On waking up, the CAN controller will generate a Wake-Up Interrupt (bCan_WUI) (to enable this interrupt, set bCan_WUIE bit).

CAUTION

- If the CAN controller is awakened by bus activity, it will not receive any messages until after it has detected a Bus Free sequence of 11 recessive bits on the bus.
- You should also note that it is not possible to select Sleep Mode while the CAN controller is in Reset Mode.

6.5.7 Acceptance Filtering

Within a CAN network, all nodes receive all messages transmitted on the bus.

To allow a node to ignore messages that are not relevant to it, the CAN controller allows pre-filtering of received messages by applying a 4 bytes Acceptance Filter to received data. Only the messages with identifier bits that match the filter are passed to the Receive FIFO.

Normally message filtering is based upon the whole identifier, which can be 11 or 29 bits long depending on whether the received message is a standard or extended frame format. However, in the CAN controller, optional mask registers allow messages with groups of identifiers to be received and placed into the Receive FIFO by setting particular identifier bits to be “don’t care”.

The filtering is carried out using four 8-bit Acceptance Code Registers (rCan_ACR0..3 which hold the bit patterns to match), together with four 8-bit Acceptance Mask Registers (rCan_AMR0..3) which mark particular bits of the Acceptance Code bit patterns as “don’t care”. Both sets of registers are applied either as a single 32-bit filter to the first 4 bytes of each received message, or as two separate 16-bit filters to the first 2 bytes of the message. Where two filters are applied, messages are accepted when the identifier bits tested match at least one of the filters.

The precise application of the filters depends on whether the data is in Standard Frame Format or Extended Frame Format and on whether one or two filters are applied.

The CAN controller filters the incoming data stream, discarding any message that does not have the required bit pattern in its identifier.

The bit pattern against which the message identifier is matched is held in the Acceptance Code Registers (rCan_ACR0..3), masked by the values held in the Acceptance Mask Registers (rCan_AMR0..3).

- “0”s in rCan_AMR0..3 identify the bits at the corresponding positions in rCan_ACR0..3 which must be matched in the message identifier
- “1”s identify the corresponding bits as “don’t care”.

The bit patterns held as rCan_ACR0..3 can either be used as a single 4 byte filter or two shorter filters. The selection is made through the bCan_AFM bit of the Mode Register (rCan_MOD).

- If bCan_AFM= “1”, a single filter will be applied.

- If bCan_AFM= “0”, two filters will be applied. Where two filters are used, the incoming message will be accepted if its identifier matches either filter.

The way in which the bit patterns defined by rCan_ACR0..3 are applied further depend on whether the incoming message is in Standard Frame Format (SFF) or Extended Frame Format (EFF) as follows:

Table 6.32 Standard Frame Format, Single Filter

Standard Frame Format (SFF)		Acceptance Filtering Condition	
CAN Offset	Receive Buffer Field	Filter Code	Filter Mask
12’h040	RX Frame Information		
12’h044	RX Identifier 1 ID[28,27,26,25,24,23,22,21]	Filter1: rCan_ACR0[7:0]	Filter1: rCan_AMR0[7:0]
12’h048	RX Identifier 2 ID[20,19,18],RTR,X,X,X,X *X: No filtering*	Filter1: rCan_ACR1[7:4] Not used: rCan_ACR1[3:0]	Filter1: rCan_AMR1[7:4] Not used: rCan_AMR1[3:0]
12’h04C	RX Data byte 1	Filter1: rCan_ACR2[7:0]	Filter1: rCan_AMR2[7:0]
12’h050	RX Data byte 2	Filter1: rCan_ACR3[7:0]	Filter1: rCan_AMR3[7:0]

Table 6.33 Standard Frame Format, Dual Filter

Standard Frame Format (SFF)		Acceptance Filtering Condition	
CAN Offset	Receive Buffer Field	Filter Code	Filter Mask
12’h040	RX Frame Information		
12’h044	RX Identifier 1 ID[28,27,26,25,24,23,22,21]	Filter1: rCan_ACR0[7:0] Filter2: rCan_ACR2[7:0]	Filter1: rCan_AMR0[7:0] Filter2: rCan_AMR2[7:0]
12’h048	RX Identifier 2 ID[20,19,18],RTR,X,X,X,X *X: No filtering*	Filter1: rCan_ACR1[7:4] Filter2: rCan_ACR3[7:4]	Filter1: rCan_AMR1[7:4] Filter2: rCan_AMR3[7:4]
12’h04C	RX Data byte 1 Byte[7:4] Byte[3:0]	Filter1: rCan_ACR1[3:0] Filter1: rCan_ACR3[3:0]	Filter1: rCan_AMR1[3:0] Filter1: rCan_AMR3[3:0]
12’h050	RX Data byte 2 *No filtering*		

Table 6.34 Extended Frame Format, Single Filter

Extended Frame Format (EFF)		Acceptance Filtering Condition	
CAN Offset	Receive Buffer Field	Filter Code	Filter Mask
12’h040	RX Frame Information		
12’h044	RX Identifier 1 ID[28,27,26,25,24,23,22,21]	Filter1: rCan_ACR0[7:0]	Filter1: rCan_AMR0[7:0]
12’h048	RX Identifier 2 ID[20,19,18,17,16,15,14,13]	Filter1: rCan_ACR1[7:0]	Filter1: rCan_AMR1[7:0]
12’h04C	RX Identifier 3 ID[12,11,10,9,8,7,6,5]	Filter1: rCan_ACR2[7:0]	Filter1: rCan_AMR2[7:0]
12’h050	RX Identifier 4 ID[4,3,2,1,0],RTR,X,X *X: No filtering*	Filter1: rCan_ACR3[7:2] Not used: rCan_ACR3[1:0]	Filter1: rCan_AMR3[7:2] Not used: rCan_AMR3[1:0]

Table 6.35 Extended Frame Format, Dual Filter

Extended Frame Format (EFF)		Acceptance Filtering Condition	
CAN Offset	Receive Buffer Field	Filter Code	Filter Mask
12'h040	RX Frame Information		
12'h044	RX Identifier 1 ID[28,27,26,25,24,23,22,21]	Filter1: rCan_ACR0[7:0] Filter2: rCan_ACR2[7:0]	Filter1: rCan_AMR0[7:0] Filter2: rCan_AMR2[7:0]
12'h048	RX Identifier 2 ID[20,19,18,17,16,15,14,13]	Filter1: rCan_ACR1[7:0] Filter2: rCan_ACR3[7:0]	Filter1: rCan_AMR1[7:0] Filter2: rCan_AMR3[7:0]
12'h04C	RX Identifier 3 ID[12,11,10,9,8,7,6,5] *No filtering*		
12'h050	RX Identifier 4 ID[4,3,2,1,0],RTR,X,X *No filtering*		

6.5.8 Interrupts Generation

The CAN controller supports the generation of an interrupt for any of the following conditions:

- Bus activity while the CAN controller is in Sleep Mode (Wake-Up Interrupt, bCan_WUI)
- Receipt of a message (Receive Interrupt, bCan_RI)
- Completion of the current transmission (Transmit Interrupt, bCan_TI)
- Loss of Received data through the FIFO being full (Data Overrun Interrupt, bCan_DOI)
- Loss of Arbitration on the CAN bus (Arbitration Loss Interrupt, bCan_ALI)
- Error on the CAN bus (Bus Error Interrupt, bCan_BEI)
- CAN controller coming out of “Error Passive” state (Error Passive Interrupt, bCan_EPI)
- The number of errors either exceeding the Error Warning Limit (rCan_EWLR) or causing the device to go into “Bus Off” state (Error Warning Interrupt, bCan_EI)

Only when using “Sync frame” transmission mechanism

- Completion of the “Sync frame” transmission.
  - Status of “Sync frame” interrupt, bCan_SendSyncFrameInt
- Overrun on the transmission of “Sync frame”.
  - The transmission of “Sync frame” is delayed to have enough time to close the emission of current frame contained in rCan_SyncTransmitBuffer registers.
  - Status of overrun “Sync frame” interrupt, bCan_OverrunSyncFrameInt

Following a hardware reset, these interrupts are disabled. The user therefore needs to enable the ones they require in the Interrupt Enable Register (rCan_IER and rCan_SyncMaskInt). The selection of interrupts that are enabled is not however affected by a software reset (bCan_RM = “1” in the Mode Register (rCan_MOD)).

CAUTION

- Normally, CPU doesn’t need to monitor Loss of Arbitration events or individual bus errors as the CAN controller will automatically retry transmission when these happen.
- Reading the Interrupt Register (rCan_IR) automatically clears all the interrupts in the register with the exception of the Receive Interrupt (bCan_RI).

The following sections describe the actions to be taken in response to each type of interrupt.

6.5.8.1 Receive Interrupts

The generation of a Receive Interrupt indicates the availability of a message to be read in the Receive FIFO. The message is read through a 13-byte window onto the Receive FIFO referred the Receive Buffer, which is located at CAN offsets 12’h040 - 12’h070 (See rCan_RdReceiveBuffer register).

Once the message currently accessible through the Receive Buffer has been read, the CPU needs to release the window it currently has on the FIFO by issuing a Release Receive Buffer command (bCan_RRB= “1”). The RX FIFO Read Pointer (and hence the Receive Buffer Start Address) then moves to the position in the Receive FIFO at which the next message will start.

If there is an unread message at this position, this becomes immediately available to read through the Receive Buffer. If no message is available, the Receive Interrupt (bCan_RI) and Receive Buffer Status (bCan_RBS) bits will be cleared.

6.5.8.2 Transmit Interrupts

The generation of a Transmit Interrupt (bCan_TI) indicates the readiness of the Transmit Buffer to receive another message for transmission. The response made to this interrupt simply depends on whether there is further data to be sent. If there is, the transmission procedure outlined in Transmission chapter needs to be repeated. If not, the interrupt may be ignored.

6.5.8.3 Error Warning Interrupts

The generation of an Error Warning interrupt (bCan_EI) indicates either that the count of transmission errors (bCan_TXERR) or the count of reception errors (bCan_RXERR) has passed the EWL value held in the Error Warning Limit Register (rCan_EWLR), or that the CAN controller has been put into “Bus Off” state because the number of transmission errors (bCan_TXERR) has exceeded 255.

The count of reception errors is held in the rCan_RXERR register, the count of transmission errors is held in the rCan_TXERR register.

If the CAN controller has been placed in “Bus Off” state, the Bus Status bit (bCan_BS) will be set to “1” (Bus Off). In addition, the Reset Mode bit (bCan_RM) will have been set, causing a software reset and placing the CAN controller in Reset Mode where it will then stay until the host CPU clears the Reset Mode bit in the Mode Register (bCan_RM).

Furthermore, on its return to Operating Mode, the CAN controller will wait for 128 occurrences of the Bus Free sequence of 11 successive recessive bits (the minimum time defined by the CAN protocol) before becoming “Bus On” again.

CAUTION

- During this period, the progress that is being made towards “Bus On” can be monitored by reading the rCan_TXERR register. On leaving Reset Mode, this is initially set to 127. It then counts down through the required number of Bus Free sequences to become zero at the point when the device is allowed to become “Bus On” again.
- If the interrupt has been generated as a result of the EWL value being exceeded, it is up to the programmer what action is taken in response to the generation of this interrupt.

6.5.8.4 Data Overrun Interrupts

A Data Overrun Interrupt (bCan_DOI) is only generated when the required storage space for the received message is greater than the number of free bytes in the Receive FIFO. The Data Overrun Status bit (bCan_DOS) will also be set.

The required storage space is determined from the RTR, FF and DLC bits of the received message which respectively define:

- Whether the message is a Remote Transmission Request
- Whether it is a standard frame format or extended frame format message
- And the number of bytes included in the message.

The assessment of the space required is made after the message has been received. If insufficient space is available to store the message, the message will be lost.

The recovery that can be made when messages are lost will depend on the system design. However, experiencing significant numbers of Data Overrun events would suggest that the volume of data traffic has been under estimated and that the system would benefit from a larger memory buffer for incoming messages.

CAUTION

Any interrupt routine that handles Data Overrun events should finish by issuing a Clear Data Overrun command (bCan_CDO = 1) to clear the Data Overrun Status bit, because no further Data Overrun interrupt (bCan_DOI) will be generated while the Data Overrun Status bit (bCan_CDO) remains set.

6.5.8.5 Wakeup Interrupts

This function is implemented for firmware compatibility and has no impact on power saving.

- A Wake-Up Interrupt (bCan_WUI) is generated when the CAN controller is awakened from Sleep Mode.

Any of the following events will cause the CAN controller to "wake up" from Sleep Mode.

- Clearing the Sleep Mode bit (bCan_SM)
- Activity on the CAN bus input (CAN_RXD)
- See **Section 6.5.6, Sleep Mode**.

6.5.8.6 Error Passive Interrupts

The Receive Error (rCan_RXERR) and Transmit Error (rCan_TXERR) counters are respectively automatically incremented by one each time a Receive error or Transmit error occurs, and decremented by one by each successful reception or transmission.

If the accumulated total of either Receive or Transmit Errors goes over 127, the CAN controller goes into state in which further errors continue to be counted but individual interrupts are no longer generated. This state is described as "Error Passive" and an Error Passive Interrupt (bCan_EPI) is generated (to enable this interrupt, set "1" in bCan_EPIE) to signal that the Error Passive state has been entered.

The CAN controller remains in Error Passive state while either error count remains over 127. The Transmission Error count (rCan_TXERR) continues to be incremented and decremented while it remains over 127. The Receive Error count (rCan_RXERR), however, is automatically reduced to a value between 119 and 127 by each message that is successfully received, potentially taking the CAN controller out of Error Passive state.

A further Error Passive Interrupt (bCan_EPI) is generated when the CAN controller leaves Error Passive state.

6.5.8.7 Arbitration Loss Interrupts

The generation of an Arbitration Loss Interrupt (bCan_ALI) indicates that the CAN controller has lost control of the CAN bus while it was in the process of transmitting a message.

Normally, there is no need for any special action to be taken as the CAN controller will automatically try again to transmit the current message. The fact that arbitration has been lost may however be of importance if the option of a single-shot transmission has been taken (See **Section 6.5.3, Transmission**).

The bit position at which arbitration was lost will be held in the Arbitration Lost Capture Register (rCan_ALC). For details of the way in which this bit position is held, see **Section 6.5.9, Bus Arbitration**.

CAUTION

The details held in the Arbitration Lost Capture Register (rCan_ALC) are not cleared until this register has been read. As a result, no further information about arbitration loss can be held until the previously held details have been read from the rCan_ALC Register.

6.5.8.8 Bus Error Interrupts

The generation of a Bus Error Interrupt (bCan_BEI) indicates the occurrence of a transmission error on the CAN bus. Normally, there is no need for any special action to be taken as the CAN controller will automatically discard any incoming message in which bus errors have occurred and it will automatically try to send again any transmit message that experienced bus errors.

However, should additional information on a bus error be required, the type of error (bit/form/stuff/other) and the location of each error are captured in an Error Code Capture Register (rCan_ECC) where they remain until this register is read.

Experiencing significant numbers of such errors may however indicate that corrective action should be taken, so the CAN controller maintains two error counters.

- One for reception errors (rCan_RXERR)
- And one for transmission errors (rCan_TXERR)

Which are automatically incremented whenever an error occurs. Should either counter exceed the value held in the Error Warning Limit register (rCan_EWLR), an Error Warning interrupt (bCan_EI) is generated (to enable this interrupt, set “1” in bCan_EIE) while if either counter exceeds a count of 127, an Error Passive interrupt (bCan_EPI) is generated (to enable this interrupt, set “1” in bCan_EPIE).

An Error Warning interrupt (bCan_EI) will also be generated if the CAN controller goes into “Bus Off” state as a result of the count of transmission errors exceeding 255.

Both these additional interrupts are discussed above.

6.5.8.9 Transmit “Sync frame” Interrupts

CAUTION

Only when using “Sync frame” transmission mechanism.

Completion of the “Sync frame” transmission.

- If bCan_SyncRunStop bit is set to “1”, (emission of “Sync frame” is enabled), an interrupt (bCan_SendSyncFrameInt bit) is generated (to enable this interrupt, set “1” in bCan_SendSyncFrameMaskInt bit) when a “Sync frame” is sent.
- Acknowledge the interrupt by writing “1” in bCan_SendSyncFrameClearInt bit.
- See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**.

6.5.8.10 Transmit Overrun “Sync frame” Interrupts

CAUTION

Only when using “Sync frame” transmission mechanism.

Overrun on the transmission of “Sync frame”.

- The transmission of “Sync frame” is delayed to have enough time to close the emission of current frame contained in rCan_WrTransmitBuffer registers.
- Interrupt (bCan_OverrunSyncFrameInt bit) is generated (to enable this interrupt, set “1” in bCan_OverrunSyncFrameMaskInt bit) and interrupt (bCan_SendSyncFrameInt bit) is generated (to enable this interrupt, set “1” in bCan_SendSyncFrameMaskInt bit) when a “Sync frame” is sent.
- Acknowledge the interrupt by writing “1” in bCan_OverrunSyncFrameClearInt and bCan_SendSyncFrameClearInt bits.
- See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**.

6.5.9 Bus Arbitration

In CAN communication, the unit that has started transmitting first during a bus idle state gains the right to transmit. If two or more units start sending at the same time, they are arbitrated for contention by comparing their arbitration fields bitwise and the unit that has more dominant bits than other units gains the right to transmit.

The resources to manage it:
At any time, the CAN node that has control of the bus is the one with the lowest identifier. A CAN node that loses arbitration must withdraw and not attempt to control the CAN bus again until the CAN bus is idle.

When the device containing the CAN controller loses arbitration, an Arbitration Lost Interrupt (set “1” in bCan_ALI bit) will be generated, (to enable this interrupt, set bCan_ALIE bit of rCan_IER register), and the bit position at which arbitration was lost will be held in the bCan_ALC bits of rCan_ALC register

For details of the way in which this bit position is held, see bCan_ALC bits.

CAUTION

- The details held in the Arbitration Lost Capture Register are not cleared until this register has been read.
- As a result, no further information about arbitration loss can be held until the previously held details have been read from the register.

6.5.10 Error Handling

Errors in reception and transmission are handled according to the CAN 2.0B protocol specification.

The main rules on error detection are:

- All units can detect an error
  - Error detection function
- Any unit that has detected an error immediately transmits an error frame to notify the error to all other units simultaneously.
  - Error notification function
- If any message transmitting unit has detected an error, the unit forcibly aborts transmission. Then it attempts to retransmit repeatedly until its message is transmitted successfully.
  - Error recovery function

The type of error (bit, form, stuff, other) and the location of the error within the message frame are captured in an Error Code Capture (rCan_ECC) register (bCan_ECC_Code, bCan_ECC_Direction, bCan_ECC_Segment bits) where they remain until this register is read.

Bit Error:

- This error is detected when the output level of any bit and its level on the bus do not match.

Stuff Error:

- This error is detected when the same level is detected for six consecutive bits in the bit stuffing field.

CRC Error:

- This error is detected when the received CRC sequence differs from the result of CRC that is calculated from the received data.

Form Error:

- This error is detected when a fixed form bit field (CRC delimiter, ACK delimiter, EOF field) contains one or more illegal bits.

Ack Error:

- This error is detected when a transmit unit detected recessive bits in the ACK slot.

The CAN controller includes two error counters:

- One for reception errors (bCan_RXERR)
- One for transmission errors (bCan_TXERR)

Which are automatically incremented in accordance with the CAN 2.0B specification when an error occurs.

Successful reception and transmission also decrement the counters in accordance with the CAN 2.0 specification.

The CAN controller also includes an Error Warning Limit (bCan_EWLR) register, the value of which represents the number of errors in either reception or transmission at which a warning should be generated. The default value for the bCan_EWLR is 8’d96.

When either the Transmit Error Counter (bCan_TXERR) or the Receive Error Counter (bCan_RXERR) passes this value, the Error Status (bCan_ES) bit in the Status Register (rCan_SR) is set and an Error Warning Interrupt (bCan_EI) is generated (to enable this interrupt, set “1” in bCan_EIE).

If either counter goes over 127, the CAN controller goes into “Error Passive” state (as defined by the CAN protocol) and issues an Error Passive (Set “1” in bCan_EPI bit) Interrupt (to enable this interrupt, set “1” in bCan_EPIE bit).

Should the Transmit Error Counter exceed 255 (the limit of the counter), the Bus Status bit (bCan_BS) will be set to “1” (“Bus Off”), the CAN controller will be set into Reset Mode and an Error Warning Interrupt (bCan_EI) generated (to enable this interrupt, set “1” in bCan_EIE).

The CAN controller will then stay in Reset Mode until the host CPU clears the Reset Mode bit in the Mode Register (bCan_RM bit). Furthermore, on its return to Operating Mode, the CAN controller will wait for 128 occurrences of the Bus Free sequence (the minimum time defined by the CAN protocol) before becoming “Bus On” again.

The table below resumes the main rules to increment or decrement the bCan_RXERR and bCan_TXERR counters.

Table 6.36 Increment/Decrement of Transmit and Receive Error Counter

Error Event	Action Taken
Receiver detects an error	bCan_RXERR incremented by 1
Receiver detects dominant bit as the first bit after sending an Error flag	bCan_RXERR incremented by 8
Receiver detects a bit error while sending an Active error flag or an Overload flag	bCan_RXERR incremented by 8
Message successfully received	bCan_RXERR decremented by 1
Message is received successfully when the count had previously been above the Error Passive trigger level of 127	bCan_RXERR automatically set to a value between 119 and 127
14th consecutive dominant bit received after sending an Active error flag or an Overload flag	bCan_RXERR incremented by 8 and bCan_TXERR incremented by 8 both at this point and after each additional sequence of 8 consecutive dominant bits
8th consecutive dominant bit received after sending a Passive error flag	
Transmitter sends an error flag	bCan_TXERR incremented by 8
Transmitter detects a bit error while sending an Active error flag or an Overload flag	bCan_TXERR incremented by 8
Transmitter successfully transmits message	bCan_TXERR decremented by 1

6.5.11 Transmit Buffer Layout

The Transmit Buffer is subdivided into descriptor and data fields. The first byte of the descriptor field holds frame information. It describes the frame format (SFF or EFF), remote or data frame and the data length. This is then followed by either two identifier bytes for SFF or four bytes for EFF messages. The data field contains up to eight data bytes.

Table 6.37 Transmit Frame Format Description

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
CAN Offset	Field	CAN Offset	Field
12'h040	TX Frame Information	12'h040	TX Frame Information
12'h044	TX Identifier 1	12'h044	TX Identifier 1
12'h048	TX Identifier 2	12'h048	TX Identifier 2
12'h04C	TX Data byte 1	12'h04C	TX Identifier 3
12'h050	TX Data byte 2	12'h050	TX Identifier 4
12'h054	TX Data byte 3	12'h054	TX Data byte 1
12'h058	TX Data byte 4	12'h058	TX Data byte 2
12'h05C	TX Data byte 5	12'h05C	TX Data byte 3
12'h060	TX Data byte 6	12'h060	TX Data byte 4
12'h064	TX Data byte 7	12'h064	TX Data byte 5
12'h068	TX Data byte 8	12'h068	TX Data byte 6
12'h06C	Not used	12'h06C	TX Data byte 7
12'h070	Not used	12'h070	TX Data byte 8

6.5.11.1 Descriptor Field of the Transmit Buffer

The bit layout of the Descriptor Field of the Transmit Buffer is shown below, first for SFF then for EFF. The different elements of the Descriptor Field are explained in the following sections:

- FF: See Frame Format (FF)
- RTR: See Remote Request (RTR)
- DLC: See Data Length Code (DLC)
- ID: See Identifier (ID)

Table 6.38 Standard Transmit Frame (SFF) Format Description

CAN Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
12'h040	FF	RTR	X*1	X*1	DLC.3	DLC.2	DLC.1	DLC.0
12'h044	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21
12'h048	ID.20	ID.19	ID.18	X*2	X*1	X*1	X*1	X*1

Table 6.39 Extended Transmit Frame (EFF) Format Description

CAN Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
12'h040	FF	RTR	X*1	X*1	DLC.3	DLC.2	DLC.1	DLC.0
12'h044	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21
12'h048	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13
12'h04C	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5
12'h050	ID.4	ID.3	ID.2	ID.1	ID.0	X*2	X*1	X*1

Note 1. Don't care but it is recommended to put "0" to be compatible with Receive Buffer in case the Self Reception or the Self Test option is used.

Note 2. Don't care but it is recommended to match the RTR bit used in the Receive Buffer in case the Self Reception or the Self Test option is used.

6.5.11.2 Frame Format (FF)

The FF bit selects the type of frame format to be transmitted.

- "1" Selects Extended Frame Format (EFF).
- "0" Selects Standard Frame Format (SFF).

6.5.11.3 Remote Request (RTR)

The RTR bit is used to identify the frame as either a remote frame or a data frame (as defined in the CAN protocol).

- "1" Indicates a remote frame (A request for data from another node).
- "0" Indicates a data frame

6.5.11.4 Data Length Code (DLC)

The DLC [3:0] bits are used to specify the number of data bytes included in message being sent.

The maximum number of data bytes that can be included in a frame is eight so values of DLC [3:0] greater than eight are automatically interpreted as eight.

You should also note that, although no data bytes are transmitted from the local host in the case of a remote frame transmission, the data length of the remote frame should still be specified to avoid bus errors if two CAN controllers start a remote frame transmission with the same Identifier simultaneously.

6.5.11.5 Identifier (ID)

The identifier acts as the message's name, used in a receiver for acceptance filtering, and also determines the bus access priority.

The lower the binary value of the identifier the higher the priority.

In Standard Frame Format (SFF) the identifier consists of 11 bits (ID.28 to ID.18).

In Extended Frame Format (EFF) messages the identifier consists of 29 bits (ID.28 to ID.0). ID.28 is the most significant bit and is transmitted first on the bus.

6.5.11.6 Data Field

The data field should comprise the number of data bytes defined by the Data Length Code. The most significant bit of data byte 1 at CAN offset 12'h04C (SFF) or CAN offset 12'h054 (EFF) is transmitted first.

6.5.12 Receive Buffer Layout

The layout of the Receive Buffer is similar to the Transmit Buffer described in the previous section. Indeed, the configuration used was chosen specifically to be compatible with the layout of the Transmit Buffer.

Again, it is important to distinguish between Standard Frame Format (SFF) messages and the Extended Frame Format (EFF) messages.

The Receive Buffer is subdivided into descriptor and data fields. The first byte of the descriptor field holds frame information. It describes the frame format (SFF or EFF), remote or data frame and the data length. This is then followed by either two identifier bytes for SFF or four bytes for EFF messages. The data field contains up to eight data bytes.

Table 6.40 Receive Frame Format Description

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
CAN Offset	Field	CAN Offset	Field
12'h040	RX Frame Information	12'h040	RX Frame Information
12'h044	RX Identifier 1	12'h044	RX Identifier 1
12'h048	RX Identifier 2	12'h048	RX Identifier 2
12'h04C	RX Data byte 1	12'h04C	RX Identifier 3
12'h050	RX Data byte 2	12'h050	RX Identifier 4
12'h054	RX Data byte 3	12'h054	RX Data byte 1
12'h058	RX Data byte 4	12'h058	RX Data byte 2
12'h05C	RX Data byte 5	12'h05C	RX Data byte 3
12'h060	RX Data byte 6	12'h060	RX Data byte 4
12'h064	RX Data byte 7	12'h064	RX Data byte 5
12'h068	RX Data byte 8	12'h068	RX Data byte 6
12'h06C	Not used	12'h06C	RX Data byte 7
12'h070	Not used	12'h070	RX Data byte 8

The bit layout of the Descriptor Field of the Receive Buffer is shown below, first for SFF then for EFF. The different elements of the Descriptor Field are explained in the following sections:

- FF: See Frame Format (FF)
- RTR: See Remote Request (RTR)
- DLC See Data Length Code (DLC)
- ID: See Identifier (ID)

Table 6.41 Extended Transmit Frame (EFF) Format Description

CAN Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
12'h040	FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0
12'h044	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21
12'h048	ID.20	ID.19	ID.18	RTR	0	0	0	0

Table 6.42 Extended Receive Frame (EFF) Format Description

CAN Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
12'h040	FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0
12'h044	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21
12'h048	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13
12'h04C	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5
12'h050	ID.4	ID.3	ID.2	ID.1	ID.0	RTR	0	0

CAUTION

The received Data Length Code in the frame information byte (CAN Offset 12'h040) represents the length of the data sent, which may be greater than eight bytes. However, the maximum number of data bytes received will be eight.

6.5.13 Bit Period and Bus Timing Parameters

The bus timing parameters configure the CAN controller for the bit rate used on the CAN bus and set the point within each bit period at which the received bit stream is to be sampled. They also specify the degree to which the CAN controller may compensate for variations in the bit rates generated by other nodes by re-synchronizing to the bit stream.

To cater for variations in the bit rate generated by other nodes and for physical delay times both on the bus and within the CAN nodes, the bit period is seen as being composed of:

- A Synchronization segment
  - The Synchronization segment represents the part of the bit period in which the bit edge is expected to arrive.
- A Propagation segment
  - The Propagation segment represents the part of the bit time that is allowed to compensate for physical delay times.
- Two Phase Buffers
  - The two Phase Buffers surround the sampling point and are shortened or lengthened as necessary to re-synchronize to the incoming bit stream when the bit edge arrives outside of the Synchronization segment.

The length of each of these segments is defined as a number of "Time Quanta" (Tq),

- The Synchronization segment is always 1 Tq
- The Propagation segment may be 1 - 8 Tq
- The two Phase Buffers may be 1 - 8 Tq

The maximum amount by which the Phase Buffers can be lengthened or shortened is also defined as the Synchronization Jump Width. This is limited to 1 - 4 Tq and is also required to not be longer than either of the two Phase Buffers.

The timing parameters used on the CAN controller are selected through the two Bus Timing Registers (rCan_BTR0 and rCan_BTR1). Together they define the structure of the bit period.

- Both registers can only be written in Reset Mode. In Operating Mode, they are read only.

rCan_BTR0 defines:

- The Baud Rate Prescaler (bCan_BRP) defines the "time quantum" Tq of the clock for CAN as a multiple of the CAN_HCLK period.
- The number of time quanta (bCan_SJW) by which the bit period may be shortened or lengthened in attempting to re-synchronize with the current transmission.

rCan_BTR1 defines:

- The number of time quanta up to and after the point at which the sample is taken (bCan_TSEG1 and bCan_TSEG2).
  - bCan_TSEG1 represents the time between the Synchronization segment and the sample point (i.e. the Propagation segment plus the first Phase Buffer).
  - bCan_TSEG2 represents the time between the sample point and the end of the bit period (i.e. the second Phase Buffer).

- The number of samples taken (bCan_SAM), one or three.

The Baud Rate Prescaler defines the “time quantum” Tq of the clock for CAN as a multiple of the CAN_HCLK period.

The time quantum of the clock for CAN is given by:

Tq = 2 × Tperiod (CAN_HCLK) × (32 × bCan_BRP[5] + 16 x bCan_BRP[4]
+ 8 × bCan_BRP[3] + 4 × bCan_BRP[2] + 2 × bCan_BRP[1] + bCan_BRP[0] + 1)

Tseg1 and Tseg2 define the length of the bit period by giving the number of time quanta up to and after the point(s) at which incoming data will be sampled.

Tsyncseg = 1 × Tq
Tseg1 = Tq × (8 × bCan_TSEG1[3] + 4 x bCan_TSEG1[2] + 2 x bCan_TSEG1[1] + bCan_TSEG1[0] + 1)
Tseg2 = Tq × (4 × bCan_TSEG2[2] + 2 x bCan_TSEG2[1] + bCan_TSEG2[0] + 1)

CAUTION

- In theory, it is possible to define bit periods of between 4 and 25 Tq through these register settings.
- However, the bit periods used in practice are required to follow the BOSCH standard, which defines bit periods between 8 and 25 Tq in length.

To be sure that your utilization agreed with the CAN 2.0 specification, please follow these simple rules:

- “Resynchronization jump width” (SJW) must be set from 1 to 4 Tq.
- “Propagation Segment” must be set from 1 to 8 Tq.
- “1st Phase Buffer” must be set from “SJW” to 8 Tq.
- “2nd Phase Buffer” must be set from “1st Phase Buffer” to “1st Phase Buffer” + 2 Tq.

Applying to the Can controller registers:

- bCan_SJW = SJW − 1
- bCan_TSEG1 = “Propagation Segment” + “1st Phase Buffer” − 1
- bCan_TSEG2 = “2nd Phase Buffer” − 1

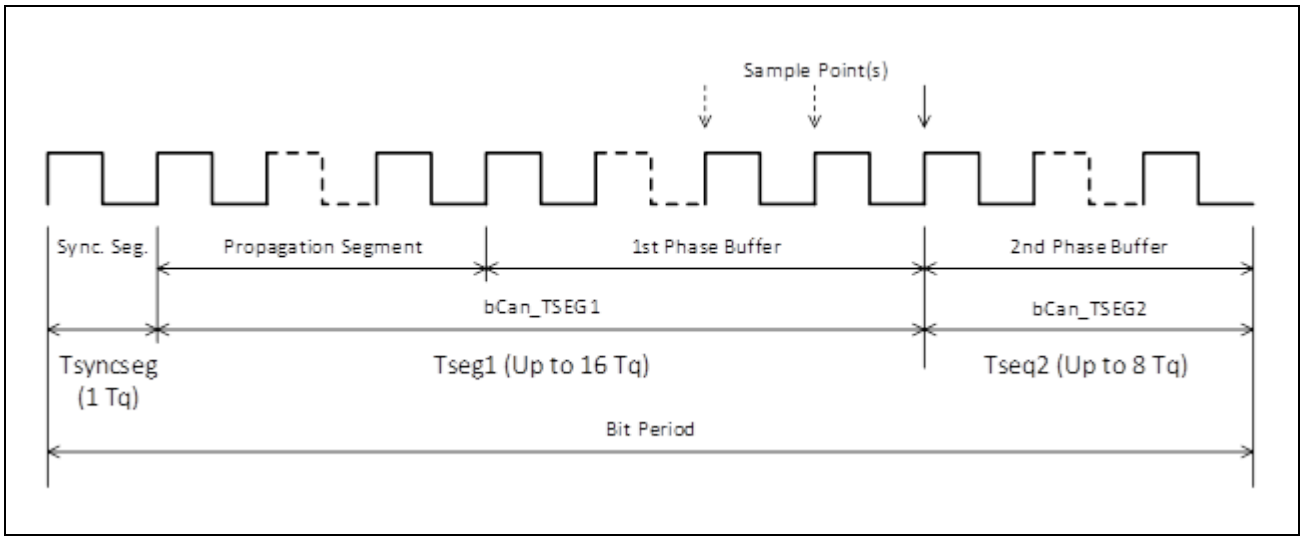

Figure 6.3 CAN: General Structure of a Bit Period, Tsyncseg, Tseg1, Tseg2

6.5.14 Reset Mode

The CAN Controller configuration to operate is set during Reset Mode. The CAN Controller is shifted to Reset Mode by “immediately after a hardware reset” or software reset (result of setting “1” to the Reset Mode bit (bCan_RM) in the Mode register (rCan_MOD)).

While in Reset Mode, you may wish to set the following aspects of the CAN controller’s operation:

- The bus timing parameters to be applied (These select the baud rate used on the CAN bus)
- The acceptance filters to be applied to received messages
- The required interrupts
- The desired error warning limit
- The required output mode

The CAN controller is returned to Operating Mode by clearing the Reset Mode bit (bCan_RM). You should also note that the CAN controller won’t send or receive any data until after it has detected a “Bus Free” sequence of 11 recessive bits on the bus (if the reset was triggered either by the host CPU or by hardware) or until it has detected 128 such sequences if the reset was caused by the Bus Status going to “Bus Off” (see rCan_TXERR register).

In running:

- Setting the Reset Mode bit in the Mode Register (bCan_RM) causes the current transmission/reception of any message to be aborted and the CAN controller to enter the Reset Mode (This happens on the next positive edge of the internal bus clock).
- The CAN controller also goes into Reset Mode on the Bus Status going to “Bus Off” (bCan_BS is set to “1”) as happens, for example, if the Transmit Error Counter (rCan_TXERR) goes over 255.

CAUTION

- If the Reset Mode was entered due to a “Bus Off” condition, the Error Warning Interrupt (bCan_EI) will be set (to enable this interrupt, set bCan_EIE bit).
- If the Reset Mode was entered due to a “Bus Off” condition, the Receive Error Counter (rCan_RXERR) will be cleared and the Transmit Error Counter (rCan_TXERR) will be initialized to 127 and used to count-down the CAN-defined bus-off recovery time consisting of 128 occurrences of 11 consecutive recessive bits.
- The CAN controller needs to be taken out of Reset Mode before any data can be sent or received.

6.5.15 Synchronization Frame

Only when using “Sync frame” transmission mechanism:

The Can link can be used to communicate with drives and to synchronize them by a synchronization frame (“Sync frame”). The configuration is limited to 8 drives and the target performance is to refresh 4 drives within 2 ms (or 8 drives within 4 ms). The “Sync frame” must be sent periodically, with a jitter as small as possible, the upper maximum limit value is 70 µs.

The “Sync frame” is a standard frame without data. Its length on the CAN bus is 44 bits split into:

- 1 bit: Start field (implicit)
- 12 bits: Arbitration field
- 6 bits: Control field
- 0 bits: Data field (could be up to 64 bits)
- 16 bits: CRC field (implicit)
- 2 bits: Ack field (implicit)
- 7 bits: End of frame (implicit)
- + bit stuffing according to the frame content

6.5.15.1 CANopen Synchronous Frame Configuration

Concerning the “Sync frame”, the main features of CAN controller are:

- Send a “Sync frame” periodically controlled by programmable dedicated time base in bit period unit:
  - Period of “Sync frame” (bCan_SyncPeriod bit in rCan_SyncPeriod register)
  - Time window dedicated for “Sync frame” (bCan_SyncMaskFrameTime bit in rCan_SyncPeriod register)
  - Allows the CAN controller to perform the following actions:
    Send the last frame written by the CPU in rCan_WrTransmitBuffer registers
    Manage the potential retries on this frame (arbitration loss).
    See **Section 6.5.9, Bus Arbitration**.
    Send a “Sync frame” initialized in rCan_SyncTransmitBuffer registers.
  - See **Figure 6.4, CANopen: Emission of Periodic Sync Frame**.
- A “Sync frame” to be transmitted is written to the CAN’s “Sync Frame Transmit Buffer” (rCan_SyncTransmitBuffer registers), controlled by bCan_SyncRunStop bit (Start and stop “Sync frame” emission) and bCan_SyncMaskFrame bit (Time window reserved for “Sync frame” emission).
  - See **Figure 6.5, CANopen: Start and Stop of Periodic Sync Frame**.
- When “Sync frame” is sent, send an interrupt to CPU (CAN_Int) controlled by:
  - Status Interrupts in rCan_SyncStatusInt register.
  - Mask interrupts in rCan_SyncMaskInt register.
  - Acknowledge interrupts in rCan_SyncClearInt register.
  - See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**.
- Function mode controlled by specifics write sequences
  - Write rCan_SyncClearSetRunStop with 32’h2052_756E.
    “Run mode”

Enable emission of “Sync frame”.
Set “1” in bCan_SyncRunStop bit.
Generation of bCan_SyncMaskFrame.
An interrupt is generated when a “Sync frame” is sent.

- Write rCan_SyncClearSetRunStop with 32’h5374_6F70
  “Stop mode”
  Disable emission of “Sync frame”
  Clear bCan_SyncRunStop bit to “0”.
  Clear bCan_SyncMaskFrame bit to “0”.
  Clear all timers dedicated for “sync frame” management

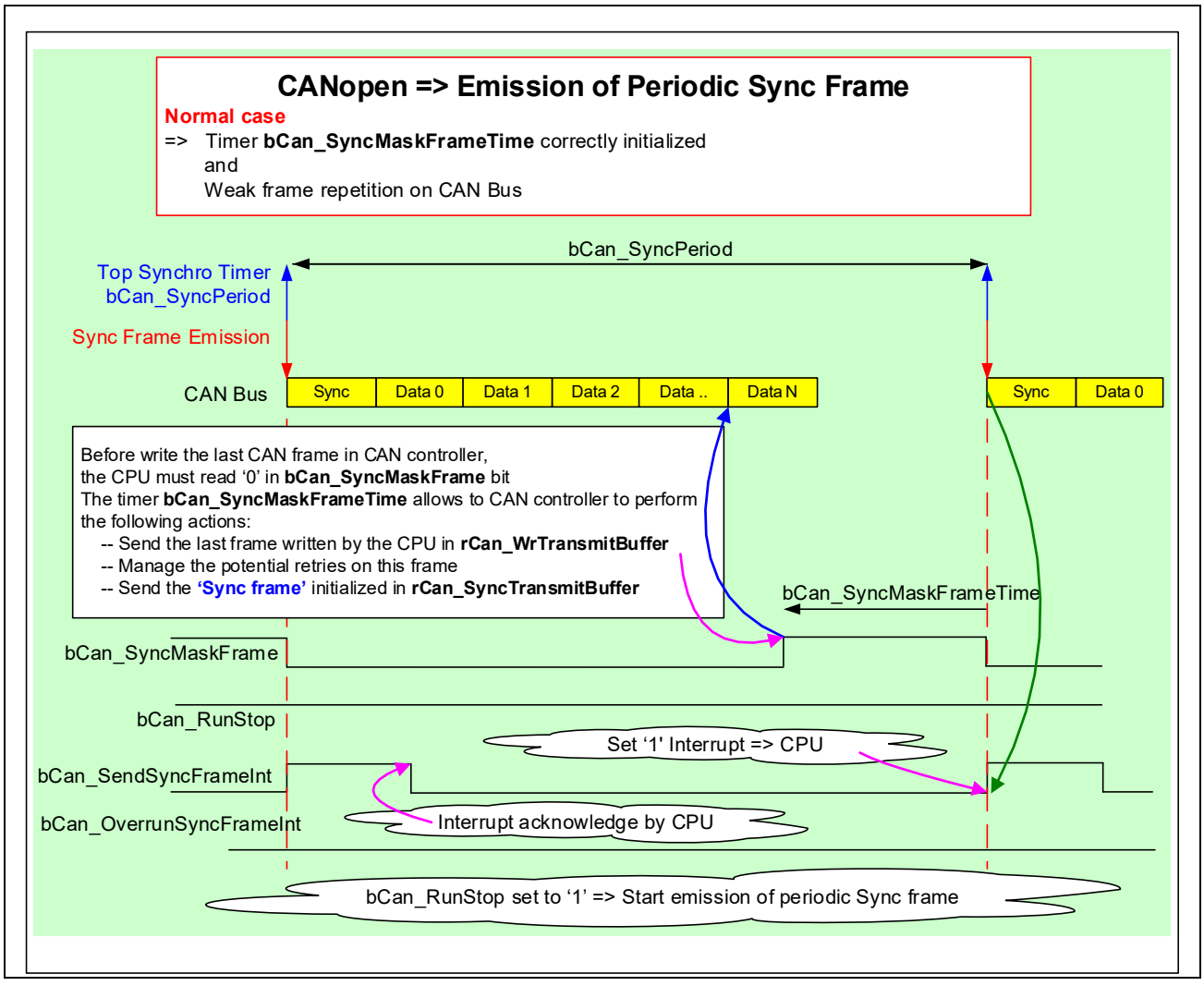

Figure 6.4 CANopen: Emission of Periodic Sync Frame

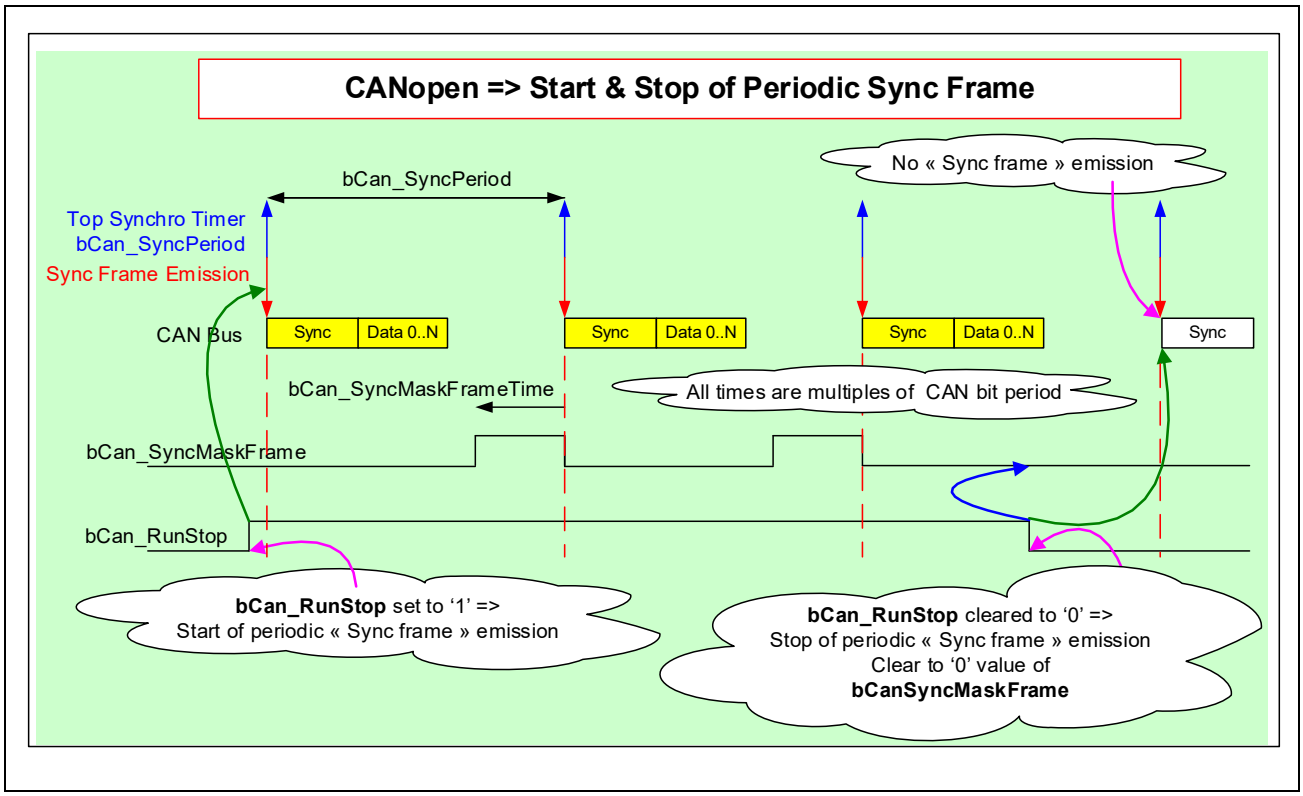

Figure 6.5 CANopen: Start and Stop of Periodic Sync Frame

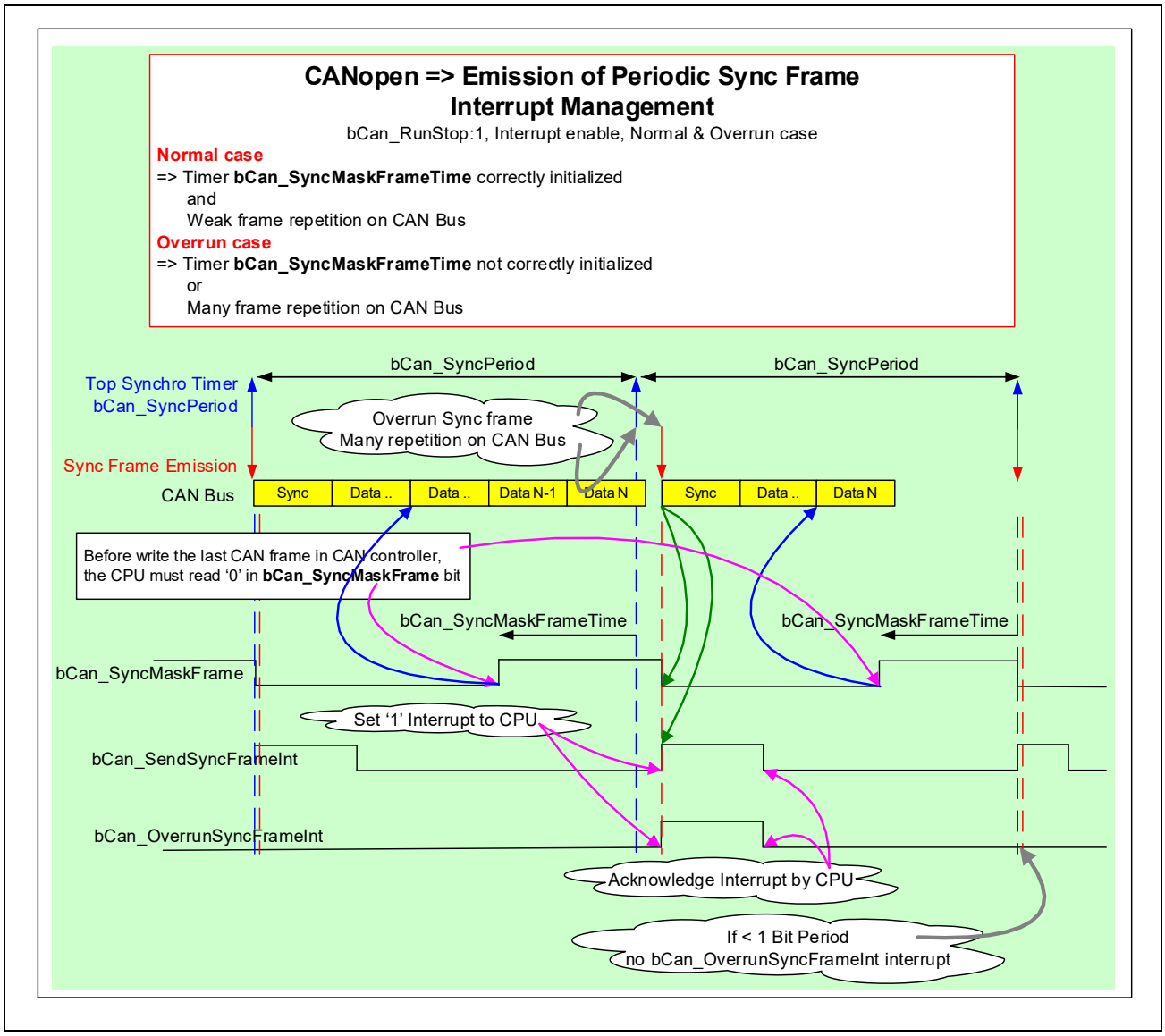

Figure 6.6 CANopen: Interrupt Management on Periodic Sync Frame

A secondary feature has also been implemented to allow the user to only have the sync frame interrupt without sending a “Sync frame” on the network.

There is three ways to enable this feature:

- By setting the bCan_TimerOnlyMode in the rCan_SyncStatus register; using this option deactivate the configuration and the launch of the sync frame but keep the interrupt system enabled, in this mode you can always access the CAN controller, even when bCan_SyncMaskFrameTime is enabled.
- By setting the bCan_TimerOnlyIfBusOff in the rCan_SyncStatus register; using this option will deactivate the configuration and the launch of the sync frame only if bCan_BS in the rCan_SR register is set, if such thing happens the system will automatically set the bCan_TimerOnlyMode in the rCan_SyncStatus.
- By setting the rCan_SyncPassiveError register to a value other than “0”; when activated, the “Passive error detection system” will check the number of Bit Periods between a delayed launch of a “Sync frame” and the end of the current “Sync period”, as soon as this number is lesser than bCan_SyncPassiveError, the system will automatically set the bCan_TimerOnlyMode in the rCan_SyncStatus.

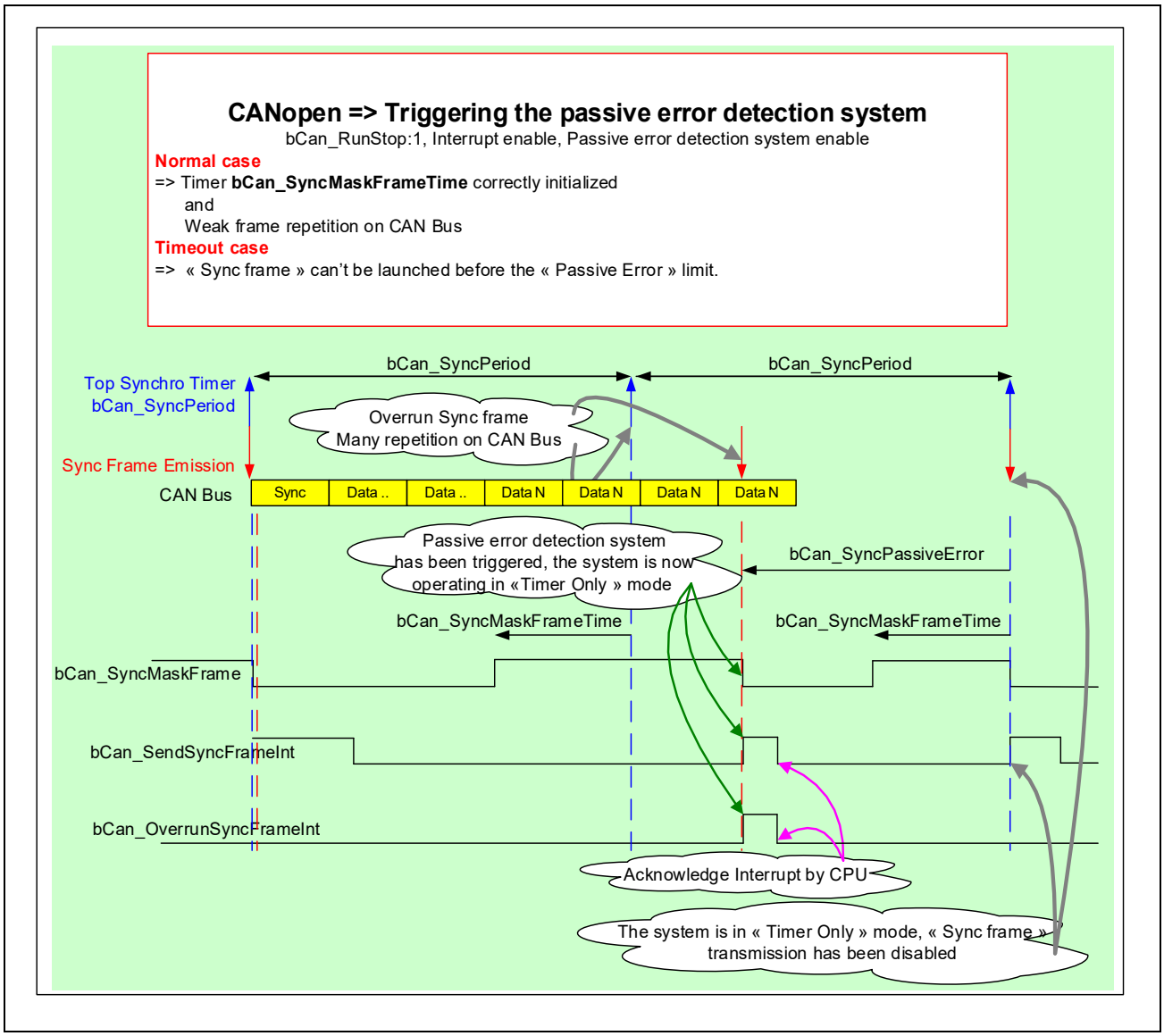

Figure 6.7 CANopen: Triggering the Passive Error Detection System

6.5.15.2 CANopen Emission of “Sync Frame”

Concerning the time window dedicated for “Sync frame”

This time window allows the CAN controller to perform the following actions:

- Send the last frame written by the CPU in rCan_WrTransmitBuffer registers.
- Manage the potential retries on this frame (arbitration loss)
- Send a “Sync frame” initialized in rCan_SyncTransmitBuffer registers.

To calculate the size of this time window, the user must take in account:

- The time necessary to send on CAN bus the potential last frame contained in rCan_WrTransmitBuffer without retry.
- The time necessary to manage 2 or 3 retries on CAN bus (arbitration loss).
  - See **Section 6.5.9, Bus Arbitration**.
- The time necessary by CAN controller to copy a “Sync frame” initialized in rCan_SyncTransmitBuffer to rCan_WrTransmitBuffer registers and to send a write command in rCan_CMR register.
  Take 8 bits period unit.

Before sending a “Sync frame”, the CAN controller will perform the following actions (starting 8 bits period unit before the end of the “Sync Period”):

[Step 1] Read bCan_BS bit (Bus Status)

1’b0 : “Bus on” mode, the CAN controller is involved in bus activities, jump to Step2

1’b1 : “Bus off” mode
If bCan_TimerOnlyIfBusOff is cleared to “0”: Clear bCan_SyncRunStop bit to “0”. All the “Sync frame” system is disabled, the transmission is aborted.
If bCan_TimerOnlyIfBusOff is set to “1”: The CAN controller automatically set “1” in bCan_TimerOnlyMode bit. The “Sync frame” transmission system is disabled, the transmission is aborted, the interrupt system is still enabled.

[Step 2] Read bCan_TBS bit (Transmit Buffer Status)

1’b1 : Transmit Buffer release, the buffer is available to write a new frame, jump to Step3

1’b0 : Transmit Buffer lock, transmission frame is running, jump to Step1

[Step 3] Copy “Sync frame” from rCan_SyncTransmitBuffer to rCan_WrTransmitBuffer

Send the “Sync frame” by a write in bCan_TR or bCan_SRR bit (depending on bCan_SyncMode bit) at the end of the “Sync period”.

[Step 4] Read bCan_BS bit (Bus Status)

1’b0 : “Bus on” mode, wait until bCan_TS is set to “1” before send the “Sync frame” interrupt and disable bCan_SyncMaskFrame bit.

1’b1 : “Bus off” mode
If bCan_TimerOnlyIfBusOff is cleared to “0”: Clear bCan_SyncRunStop bit to “0”. All the “Sync frame” system is disabled, the transmission is aborted.
If bCan_TimerOnlyIfBusOff is set to “1”: The CAN controller automatically set “1” in bCan_TimerOnlyMode

bit. The “Sync frame” transmission system is disabled, the transmission is aborted, the interrupt system is still enabled.

CAUTION

- If the “Sync frame” system is enabled and not in “TimerOnlyMode”, the CPU must not do any write access to the CAN controller during the 8 last bits period of the “Sync Period”: the configuration of the time base bCan_SyncMaskFrameTime must not be less than 8 bits period unit plus the time needed to accomplish the accesses (read/write) sequence link to the last check on the bCan_SyncMaskFrame.
- The “passive error detection system” is also checked during the Step1 and Step 4.
- The bus off detection during Step 1 and Step 4 is disabled when bCan_TimerOnlyMode is set to “1”.
- An overrun “Sync frame” interrupt is generated when there is not enough time to send “Sync frame”, please verify if:
  - The time window defined by bCan_SyncMaskFrameTime bit is correctly initialized.
  - Integrity of CAN Bus, many repetition frames (arbitration loss) on CAN Bus.
  - No others CAN master is sending frame(s) at the end of the “Sync Period”.

See **Section 6.5.9, Bus Arbitration**.

See **Figure 6.6, CANopen: Interrupt Management on Periodic Sync Frame**.

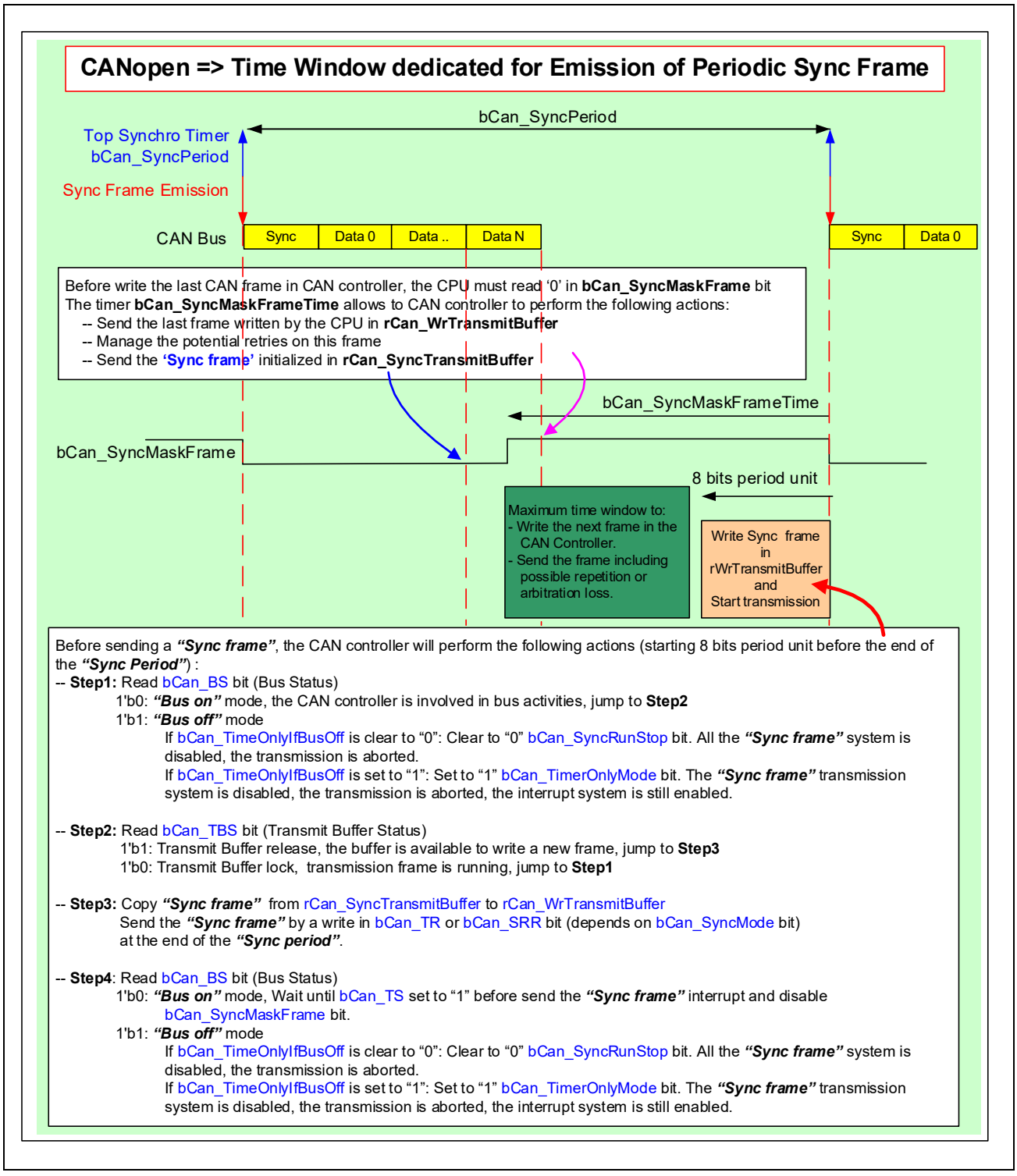

Figure 6.8 CANopen: Time Window dedicated for Emission of Periodic Sync Frame

6.5.16 Difference between CAN Controllers and Reference Philips SJA1000 Devices

This section summarizes the differences between the CAN controller and the reference Philips SJA1000 device.

Operating Modes:

- The reference device has two operating modes referred to as BasicCAN and PeliCAN. The CAN controller has a single operating mode which is broadly compatible with the SJA1000's PeliCAN mode.
- The differences between the CAN controller and the Philips SJA1000's PeliCAN mode are outlined below.

Handling of Transmitted Messages:

- The CAN controller does not copy transmitted messages to the Receive Buffer unlike the reference device.
- Instead, transmitted messages may be read back from CAN offsets 12'h180 - 12'h1B0.

Mode Register:

- Write access to rCan_MOD[3:1] is restricted to Reset Mode in the reference device. In the CAN controller, these bits can be written from either reset Mode or Operating Mode.

Output Control Register:

- The CAN controller does not support the output voltage level and polarity options selected through rCan_OCR[7:2] unlike the reference device.
- In the CAN controller, these bits are reserved and will return 0 when read.
- You should also note that the output modes available for selection through rCan_OCR[1:0] are limited to just Normal Output Mode

Clock Divider Register:

- This register is not supported

AHB Interface:

- All register's addresses of reference device are byte aligned.
- In CAN controller, all register's addresses is aligned by 32 bits with all bits 31..24 read to 0.

Sleep Mode:

- The reference device uses internal gating to disable the device on entry into Sleep Mode. The CAN_HCLK signal which drives the majority of the logic in the core can be gated instead.

6.6 Special Notice

The document CAN section is for informational and instructional purposes. It contains Mentor Graphics Corporation copyright and information. In consideration of that, any copy of this document shall retain all copyright and proprietary notices contained herein. For the avoidance of doubt, nothing contained herein shall be construed as conferring by implication, estoppel or otherwise any license or right under (i) any patent or trademark of Mentor Graphics or any third party or (ii) any Mentor Graphics Corporation copyright.

Section 7 ADC Controller and 12bit A/D Converters

7.1 Overview

ADC Controller Features

- Software conversion request
- Two level of priority are available
- Round robin management of simultaneous conversion requests with the same level of priority
- Higher priority conversion requests are inserted at the end of a lower priority conversion in progress
- When triggered, the S&H (Sample and hold) channels are sampled at the end of the conversion in progress but will be converted following the global round robin scheme
- DMA coupling
  - Start a DMA transaction on selected “End of Conversion” signal
  - Two channels available (Only one channel at the same time)
- Status register to summarize end of conversion information for each channel
- Virtual channel capability

ADC Core Features

- Up to 2 units
- Resolution 12-bit
- Sampling rate from 0.0625 MSPS to 1 MSPS
- Successive approximation
- Maximal conversion time 21 ADC_CLK
- Analog inputs
  8 channels per core (5 standard channels + 3 channels with sample/hold)
- Each channel has his own input trigger to start the conversion, the triggers are managed by the ADC Controller
- DNL, ±1.0 LSB (Max.) [at VAIN = 0.0 V to AVDD, f_{CLK} = 20 MHz]
- INL, ±4.0 LSB (Max.) [at VAIN = 0.0 V to AVDD, f_{CLK} = 20 MHz]
- Input leakage current is about 0.2 µA per input
- Power down mode
- ADC clock frequency from 4 MHz to 20 MHz

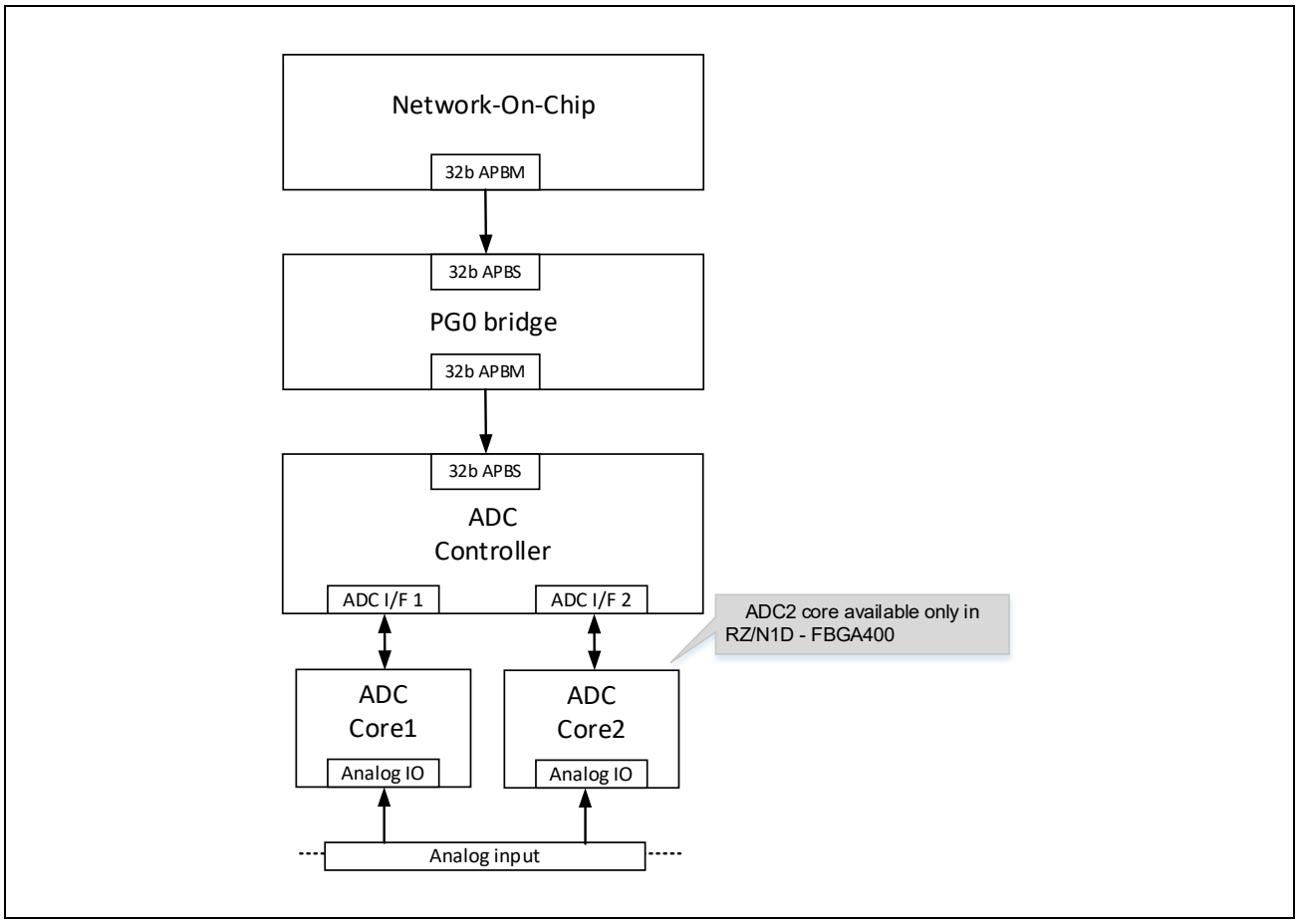

Figure 7.1 ADC Interfaces and Connections

7.1.1 Analog Buffer

ADC Cores share some of the analog Pads in order to reduce pinout of the second ADC.

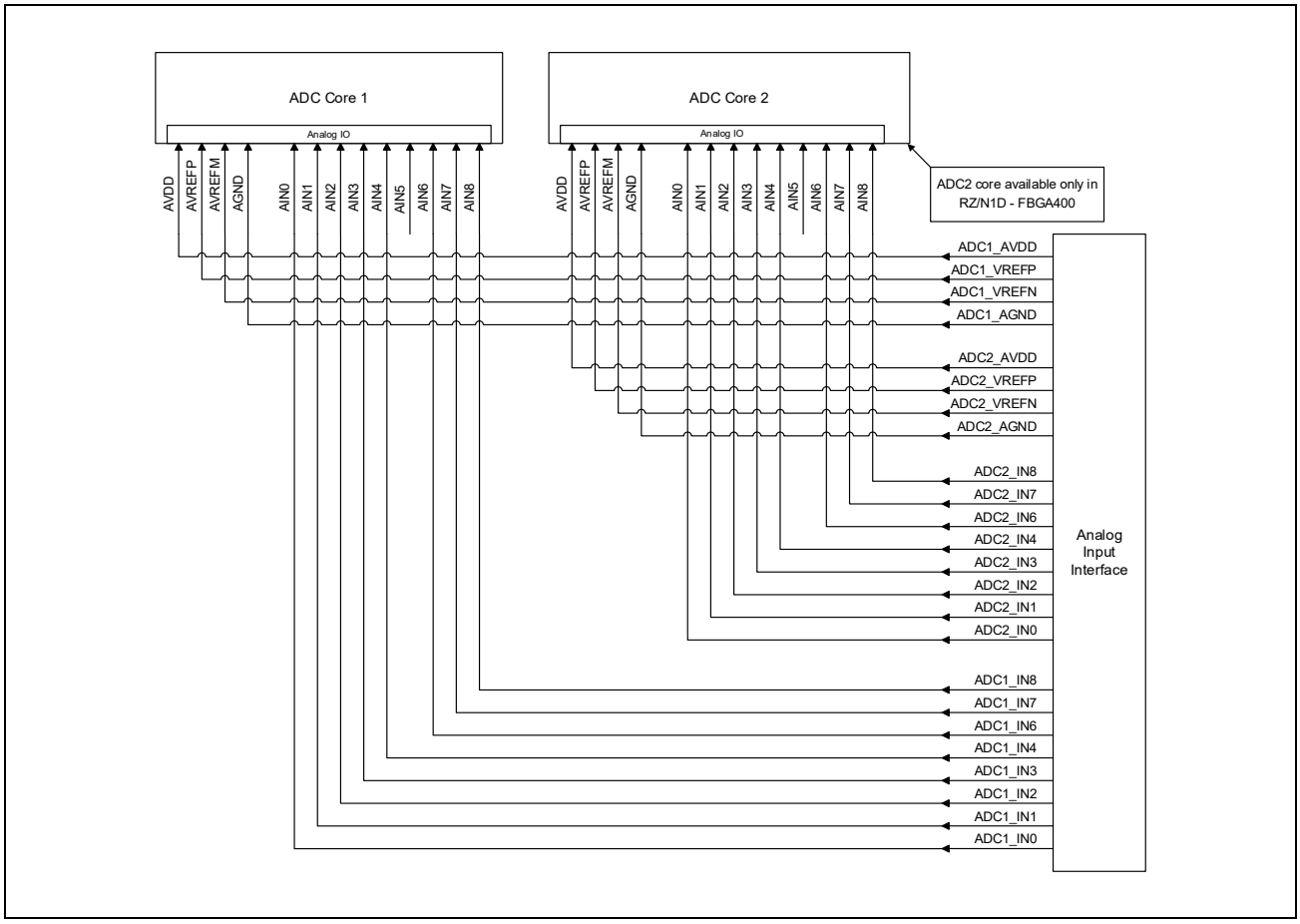

Figure 7.2 ADC Cores Analog Signals

7.2 Signal Interfaces

Signal Name	Input Output	Description
Clock		
ADC_PCLK	Input	Internal bus clock (APB)
ADC_CLK	Input	ADC clock
Interrupt		
ADC_Int	Output	Level sensitive interrupt output, Active High
External Signal		
ADC1_IN[8:6,4:0]	Input	Analog input pins for ADC1
ADC2_IN[8:6,4:0]	Input	Analog input pins for ADC2

7.3 Register Map

CAUTION

The number of available ADC depends on the package:
RZ/N1D FBGA400: ADC1 and ADC2, other: ADC1 only

7.3.1 Register Map ADC1

CAUTION

Available with all packages.

Table 7.1 A/D Converters Register Map

Address	Register Symbol	Register Name
4006 5000h	rADC_INTSTATUS0	Interrupt Status Before Masking
4006 5004h	rADC_INTSTATUS1	Interrupt Status After Masking
4006 5008h	rADC_INTCLR	Clear Interrupt
4006 500Ch	rADC_INTMASK	Mask Interrupt
4006 5010h	rADC_INTOVFSTATUS0	Interrupt Overflow Before Masking
4006 5014h	rADC_INTOVFSTATUS1	Interrupt Overflow After Masking
4006 5018h	rADC_INTCLROVF	Clear Interrupt Overflow
4006 501Ch	rADC_INTOVFMASK	Mask Interrupt Overflow
4006 5020h	rADC_PENDING	Start of Operation Pending
4006 5024h	rADC_PENDINGOVF	Start of Operation Pending Overflow
4006 5028h	rADC_PENDINGCLROVF	Clear Start of Operation Overflow
4006 502Ch	rADC_CONTROL	ADC Control
4006 5030h	rADC_FORCE	ADC Request
4006 5034h	rADC_SETFORCE	Set ADC Request
4006 5038h	rADC_CLRFORCE	Clear ADC Request
4006 503Ch	rADC_PRIORITY	ADC Priority Mode
4006 5040h	rADC_CONFIG	ADC Configuration
4006 50ACh	rADC_ACQS	ADC Control Sample and Hold
4006 50B0h + 4h × n	rADC_MASKLOCK[n] n = 0..3	Mask Data Locked [n]
4006 50C0h + 4h × n	rADC_VC[n] n = 0..15	ADC Control Register for Virtual Channel [n]
4006 5100h + 4h × n	rADC1_DATA[n] n = 0..15	ADC1 Conversion Data of Virtual Channel [n]
4006 5180h + 4h × n	rADC1_DATALOCK[n] n = 0..15	ADC1 DataLock[n] Register

7.3.2 Register Map ADC2

CAUTION

Available with package FBGA400 only.

Table 7.2 ADC2 Register Map

Address	Register Symbol	Register Name
4006 5140h + 4h × n	rADC2_DATA[n] n = 0..15	ADC2 Conversion Data of Virtual Channel [n]
4006 51C0h + 4h × n	rADC2_DATALOCK[n] n = 0..15	ADC2 DataLock[n] Register

7.4 Register Description

7.4.1 Register Description ADC1

7.4.1.1 rADC_INTSTATUS0 — Interrupt Status Before Masking

Address: 4006 5000h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_INTSTATUS0_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.3 rADC_INTSTATUS0 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_INTSTATUS0_VC	Interrupt Status Before Masking on Virtual channel ADC_VC[n] with n = 0..15 ADC_Int is asserted if bADC_INTSTATUS0_VC[n] = 1 and corresponding mask bit in rADC_INTMASK is set. Bit [n]: bADC_INTSTATUS0_VC[n] is trigger on rising edge of iADC_EOC_VC[n] 0: No interrupt 1: Interrupt has been set ***Note***) • If the ADC_VC[n] is configured in continuous mode (bADC_Continuous bit) then further trigger pulses are generated whenever a selected iADC_EOC_VC[n] event occurs even if the flag bit is set. • An ADC_VC[n] interrupt overflow event occurs in the bADC_INTOVFSTATUS0_VC[n] bit when the respective bADC_INTSTATUS0_VC[n] bit is set and a selected additional iADC_EOC_VC[n] trigger is generated. • Reading from this register does not clear any active interrupts.	R

7.4.1.2 rADC_INTSTATUS1 — Interrupt Status After Masking

Address: 4006 5004h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_INTSTATUS1_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.4 rADC_INTSTATUS1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_INTSTATUS1_VC	Interrupt Status After Masking on Virtual channel ADC_VC[n] with n = 0..15 This field shows logical AND value between bADC_INTMASK and bADC_INTSTATUS0_VC. Bit [n]: bADC_INTSTATUS1_VC[n] is trigger on rising edge of iADC_EOC_VC[n] 0: No interrupt 1: Interrupt has been set	R

7.4.1.3 rADC_INTCLR — Clear Interrupt

Address: 4006 5008h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_INTCLR_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.5 rADC_INTCLR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_INTCLR_VC	Clear Interrupt Status on Virtual channel ADC_VC[n] with n = 0..15 Clear respective interrupt bit in rADC_INTSTATUS0 and rADC_INTSTATUS1 registers. Bit [n]: bADC_INTCLR_VC[n] clear bADC_INTSTATUSx_VC[n] with x = 0..1 0: No action 1: Clears respective interrupt in bADC_INTSTATUS0_VC[n] and bADC_INTSTATUS1_VC[n] ***Note*)** • If software tries to set this bit n on the same clock cycle that hardware tries to set bADC_INTSTATUS0_VC[n] bit in the register, then hardware has priority and the bADC_INTSTATUS0_VC[n] bit will be set. In this case, the respective overflow bADC_INTOVFSTATUS0_VC[n] bit in the register will not be affected regardless of whether the bADC_INTSTATUS0_VC[n] bit was previously set or not. • Always read as 0.	W

7.4.1.4 rADC_INTMASK — Mask Interrupt

Address: 4006 500Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_INTMASK_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.6 rADC_INTMASK Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_INTMASK_VC	Mask Interrupt Status on Virtual channel ADC_VC[n] with n = 0..15 Bit [n]: bADC_INTMASK_VC[n] mask of bADC_INTSTATUS0_VC[n] 1: interrupt not masked 0: interrupt masked	R/W

7.4.1.5 rADC_INTOVFSTATUS0 — Interrupt Overflow Before Masking

Address: 4006 5010h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_INTOVFSTATUS0_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.7 rADC_INTOVFSTATUS0 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_INTOVFSTATUS0_VC	Interrupt Overflow Before Masking on Virtual channel ADC_VC[n] with n = 0..15 If the respective bADC_INTSTATUS0_VC[n] bit is set and a selected additional iADC_EOC_VC[n] trigger is generated, then an overflow condition occurs. ADC_Int is asserted if bADC_INTOVFMASK_VC[n] = 1 and corresponding mask bit in rADC_INTOVFMASK is 1. Bit [n]: bADC_INTOVFSTATUS0_VC[n] interrupt overflow of bADC_INTSTATUS0_VC[n] 0: No interrupt overflow event detected 1: Potential Interrupt overflow detected ***Note*)** The overflow bit does not care about the continuous mode bit state (bADC_Continuous). An overflow condition is generated irrespective of this mode selection.	R

7.4.1.6 rADC_INTOVFSTATUS1 — Interrupt Overflow After Masking

Address: 4006 5014h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_INTOVFSTATUS1_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.8 rADC_INTOVFSTATUS1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_INTOVFSTATUS1_VC	Interrupt Overflow After Masking on Virtual channel ADC_VC[n] with n = 0..15 This field shows logical AND value between bADC_INTOVFMASK and bADC_INTOVFSTATUS1_VC. Bit [n]: bADC_INTOVFSTATUS1_VC[n] interrupt overflow of bADC_INTSTATUS1_VC[n] 0: No interrupt overflow event detected 1: Potential Interrupt overflow detected and corresponding bADC_INTOVFMASK = 1 ***Note*)** The overflow bit does not care about the continuous mode bit state (bADC_Continuous). An overflow condition is generated irrespective of this mode selection.	R

7.4.1.7 rADC_INTCLROVF — Clear Interrupt Overflow

Address: 4006 5018h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_INTCLROVF_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.9 rADC_INTCLROVF Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_INTCLROVF_VC	Clear Interrupt Overflow on Virtual channel ADC_VC[n], with n = 0..15 Clear respective interrupt bit in rADC_INTOVFSTATUS0 and rADC_INTOVFSTATUS1 registers. Bit [n]: bADC_INTCLROVF_VC[n] clear bADC_INTOVFSTATUSx_VC[n] with x = 0..1 0: No action 1: Clears respective interrupt in bADC_INTOVFSTATUSx_VC[n] x = 0..1 ***Note*)** • If software tries to set this bit n on the same clock cycle that hardware tries to set the overflow bADC_INTOVFSTATUS0_VC[n] bit in register, then hardware has priority and the bADC_INTOVFSTATUS0_VC[n] bit will be set. • Always read as 0.	W

7.4.1.8 rADC_INTOVFMASK — Mask Interrupt Overflow

Address: 4006 501Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_INTOVFMASK_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.10 rADC_INTOVFMASK Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_INTOVFMASK_VC	Mask Interrupt Overflow on Virtual channel ADC_VC[n] with n = 0..15 Bit [n]: bADC_INTOVFMASK_VC[n] mask of bADC_INTOVFSTATUS0_VC[n] 1: interrupt not masked 0: interrupt masked	R/W

7.4.1.9 rADC_PENDING — Start of Operation Pending

Address: 4006 5020h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bADC_DMA1_RUNNING	bADC_DMA0_RUNNING	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_PENDING_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.11 rADC_PENDING Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31	bADC_DMA1_RUNNING	ADC DMA channel1 is running. When after scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run and all following conditions are satisfied: • Event detection: “Data copy into data lock registers” • Configured with bADC_Mode (rADC_VC[n] register) = 2’b11. • bADC_DMA_Request[1:0] (rADC_VC[n] register) is set to 2’b11 ADC DMA channel1 requests are started and run until detection end of DMA transfer. 1: DMA requests are running 0: No DMA request on same channel ***Note*)** • The bit will be automatically cleared when end of DMA transfer is detected on this channel. • If contention exists where this bit receives both a request to set and a request to clear on the same cycle, regardless of the source of either, this bADC_DMA1_RUNNING bit will be set and the request to clear will be ignored. In this case, the overflow bADC_DMA1_RUNNINGOVF bit in the rADC_PENDINGOVF register will not be affected regardless of whether this bit was previously set or not.	R
b30	bADC_DMA0_RUNNING	ADC DMA channel0 is running. Same as bADC_DMA1_RUNNING	R
b29 to b16	Reserved	Not used	R

Table 7.11 rADC_PENDING Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b15 to b0	bADC_PENDING_VC	Start of Operation Pending on Virtual channel ADC_VC[n] with n = 0..15 Bit [n]: bADC_PENDING_VC[n] is set when event received on ADC_VC[n] 1: Event has been received. Set when: - Rising edge detection on trigger selected by bADC_TrigSel of rADC_VC[n] - CPU changed bADC_FORCE_VC[n] bit to 1 by setting bADC_SETFORCE_VC bit - ADC request (conversion, sample and hold or other operation) is pending for ADC_VC[n] 0: No event received - No operation pending ***Note*)** • The bit will be automatically cleared when the respective ADC_VC[n] operation is started. • If contention exists where this bit n receives both a request to set and a request to clear on the same cycle, regardless of the source of either, this bADC_PENDING_VC[n] bit will be set and the request to clear will be ignored. In this case, the overflow bADC_PENDINGOVF_VC[n] bit in the rADC_PENDINGOVF register will not be affected regardless of whether this bit was previously set or not. • A conversion or sample and hold operation requested depends on bADC_Mode. • Cleared to 0 when a CPU write to rADC_CONFIG register starts the following operations: – bADC_POWER_DOWN (Enable or Disable PowerDown Mode) – bADC_SAMPLE_HOLD_ENABLE (Enable or disable Sample and Hold feature on each channel)	R

7.4.1.10 rADC_PENDINGOVF — Start of Operation Pending Overflow

Address: 4006 5024h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bADC_DMA1_RUNNINGOVF	bADC_DMA0_RUNNINGOVF	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_PENDINGOVF_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.12 rADC_PENDINGOVF Register Contents

Bit Position	Bit Name	Function	R/W
b31	bADC_DMA1_RUNNINGOVF	Overflow with ADC DMA channel1 is running Indicates a new bADC_DMA_Request[1:0]: 2'b11 was generated with a previous request was already running. 1: ADC DMA overflow on ADC_DMA1 0: No ADC overflow ***Note***) • An overflow condition does not stop ADC DMA transfer from being processed. It simply is an indication that a DMA request was missed. This means that the first DMA transfer block will be processed normally until the DMA finish occurred but the second DMA transfer block triggered is not processed.	R
b30	bADC_DMA0_RUNNINGOVF	Overflow with ADC DMA channel0 is running Indicates a new bADC_DMA_Request[1:0]: 2'b10 was generated with a previous request was already running. 1: ADC DMA overflow on ADC_DMA0 0: No ADC overflow ***Note***) • An overflow condition does not stop ADC DMA transfer from being processed. It simply is an indication that a DMA request was missed. This means that the first DMA transfer block will be processed normally until the DMA finish occurred but the second DMA transfer block triggered is not processed.	R
b29 to b16	Reserved	Not used	R
b15 to b0	bADC_PENDINGOVF_VC	Start of Operation Pending Overflow on Virtual channel ADC_VC[n] with n = 0..15 Indicates a new ADC_VC[n] event was generated while an existing event was already pending. Bit [n]: bADC_PENDINGOVF_VC[n] event overflow on ADC_VC[n] 1: Event overflow 0: No event overflow ***Note***) An overflow condition does not stop ADC_VC[n] events from being processed. It simply is an indication that a trigger was missed.	R

7.4.1.11 rADC_PENDINGCLROVF — Clear Start of Operation Overflow

Address: 4006 5028h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bADC_DMA1_RUNNINGCLROVF	bADC_DMA0_RUNNINGCLROVF	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_PENDINGCLROVF_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.13 rADC_PENDINGCLROVF Register Contents

Bit Position	Bit Name	Function	R/W
b31	bADC_DMA1_RUNNINGCLROVF	Clear Overflow with ADC DMA channel1 is running Writing a 1 will clear the bADC_DMA1_RUNNINGOVF bit. 1: Clear the bADC_DMA1_RUNNINGOVF bit 0: No Action ***Note***) • If software tries to set this bit on the same clock cycle that hardware tries to set the overflow bADC_DMA1_RUNNINGOVF bit in the rADC_PENDINGOVF register, then hardware has priority and the bADC_DMA1_RUNNINGOVF bit will be set. • Always read as 0.	W
b30	bADC_DMA0_RUNNINGCLROVF	Clear Overflow with ADC DMA channel0 is running Same as bADC_DMA1_RUNNINGCLROVF	W
b29 to b16	Reserved	Not used	R
b15 to b0	bADC_PENDINGCLROVF_VC	Clear Start of Operation Overflow on Virtual channel ADC_VC[n] with n = 0..15 Writing a 1 will clear the respective ADC_VC[n] pending flag overflow. Bit [n]: bADC_PENDINGCLROVF_VC[n] clear Event overflow on ADC_VC[n] 1: Clear respective bADC_PENDINGOVF_VC[n] in rADC_PENDINGOVF register. 0: No action ***Note***) • If software tries to set this bit n on the same clock cycle that hardware tries to set the overflow bADC_PENDINGOVF_VC[n] bit in the rADC_PENDINGOVF register, then hardware has priority and the bADC_PENDINGOVF_VC[n] bit will be set. • Always read as 0.	W

7.4.1.12 rADC_CONTROL — ADC Control

Address: 4006 502Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	bADC_BUSY	bADC_VC_BUSY	bADC_VC_RUN				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.14 rADC_CONTROL Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b7	Reserved	Not used	R
b6	bADC_BUSY	ADC Configuration Change in Running. All ADC operations in pending will be stopped. • Set to 1 when a CPU write to rADC_CONFIG register starts the following operations: bADC_POWER_DOWN - Enable or Disable PowerDown Mode bADC_SAMPLE_HOLD_ENABLE - Enable or disable Sample and Hold feature on each channel • Cleared to 0 when the following operations are finished bADC_POWER_DOWN - Enable or Disable PowerDown Mode bADC_SAMPLE_HOLD_ENABLE - Enable or disable Sample and Hold feature on each channel 1: ADCs Configuration change in running 0: ADCs Configuration change finished ***Note*)** • This bit must be polling by CPU until hardware reset by state machine (Configuration change finished) • After CPU read to 0, the configuration of virtual channel can be started.	R
b5	bADC_VC_BUSY	ADC_VC Busy Set when ADC_VC is running and cleared when it is in idle state (wait event) 0: ADC_VC is available to run next virtual channel 1: ADC_VC is busy and cannot run another virtual channel	R
b4 to b0	bADC_VC_RUN	ADC_VC Channel Status When bADC_VC_BUSY = 0, holds the value of the last executed ADC_VC When bADC_VC_BUSY = 1, reflects the ADC_VC currently being processed 5'h00: ADC_VC0 is currently processing or was last ADC_VC's executed ... ... 5'h0F: ADC_VC15 is currently processing or was last ADC_VC's executed 5'h1x: Reserved	R

7.4.1.13 rADC_FORCE — ADC Request

Address: 4006 5030h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_FORCE_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.15 rADC_FORCE Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_FORCE_VC	Force Start of Operation on Virtual channel ADC_VC[n] with n = 0..15 Writing a 1 to rADC_SETFORCE register will force to 1 the respective bADC_PENDING_VC[n] flag bit in the rADC_PENDING register. This can be used to initiate a software initiated operation. Force Start of Operation on each virtual channel ADC_VC[n] is associated directly with following allocation bits: Bit [n]: bADC_FORCE_VC[n] set start operation in bADC_PENDING_VC[n] bit 1: Force an ADC request to start once priority is given to ADC_VC[n] 0: No action. ***Note*)** • If software tries to set this bit on the same clock cycle that hardware tries to clear the bADC_PENDING_VC[n] bit in the rADC_PENDING register, then software has priority and the bADC_PENDING_VC[n] bit will be set. In this case, the overflow bit in the rADC_PENDINGOVF register will not be affected regardless of whether the bADC_PENDING_VC[n] bit was previously set or not. • In Single mode (bADC_Continuous bit is cleared to 1'b0) for each virtual channel ADC_VC[n] with n = 0..15 – ADC operation (conversion, sample and hold or other) is to be performed only once. bADC_FORCE_VC[n] bit is automatically cleared to 0 when operation ends on the selected channel. • In Continuous mode (bADC_Continuous bit is set to 1'b1) for each virtual channel ADC_VC[n] with n = 0..15 – ADC operation is continuously performed for the selected channels in sequence until bADC_FORCE_VC[n] bit is cleared by CPU write to rADC_CLRFORCE register. • Cleared to 0 when a CPU write to rADC_CONFIG register starts the following operations: – bADC_POWER_DOWN (Enable or Disable PowerDown Mode) – bADC_SAMPLE_HOLD_ENABLE (Enable or disable Sample and Hold feature on each channel)	R

7.4.1.14 rADC_SETFORCE — Set ADC Request

Address: 4006 5034h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_SETFORCE_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.16 rADC_SETFORCE Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_SETFORCE_VC	Set Force Start of Operation on Virtual channel ADC_VC[n] with n = 0..15 Set respective Force Start of Operation bit in rADC_FORCE register. Bit [n]: bADC_SETFORCE_VC[n] set bADC_FORCE_VC[n] bit 1: Set respective bADC_FORCE_VC[n] bit in rADC_FORCE register 0: No action, writes of 0 are ignored ***Note*)** Always read as 0.	W

7.4.1.15 rADC_CLRFORCE — Clear ADC Request

Address: 4006 5038h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_CLRFORCE_VC															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.17 rADC_CLRFORCE Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Not used	R
b15 to b0	bADC_CLRFORCE_VC	Clear Force Start of Operation on Virtual channel ADC_VC[n] with n = 0..15 Clear respective Force Start of Operation bit in rADC_FORCE register. Bit [n]: bADC_CLRFORCE_VC[n] clear bADC_FORCE_VC[n] bit 1: Clear respective bADC_FORCE_VC[n] bit in rADC_FORCE register 0: No action, writes of 0 are ignored ***Note*)** Always read as 0.	W

7.4.1.16 rADC_PRIORITY — ADC Priority Mode

Address: 4006 503Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	bADC_RR_Pointer					bADC_Priority				
Value after reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Table 7.18 rADC_PRIORITY Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b10	Reserved	Not used	R
b9 to b5	bADC_RR_Pointer	ADC_VC[n] Round Robin Pointer with n=0..15 Holds the value of the last converted round robin ADC_VC[n] to be used by the round robin scheme to determine order of conversions. 5'h00: – ADC_VC0 was last round robin ADC_VC to convert. – ADC_VC1 is highest round robin priority. 5'h01: – ADC_VC1 was last round robin ADC_VC to convert. – ADC_VC2 is highest round robin priority. ... 5'h0E: – ADC_VC14 was last round robin ADC_VC to convert. – ADC_VC14 is highest round robin priority. 5'h0F: – ADC_VC15 was last round robin ADC_VC to convert. – ADC_VC0 is highest round robin priority. 5'h10: – Reset value to indicate no ADC_VC has been converted. ADC_VC0 is highest round robin priority. – Set to this value when the bADC_Priority bit is written. If a conversion is currently in progress, it will be completed and then the new priority is become. – Set to this value when the device is reset or when a CPU write to rADC_CONFIG register is starting the following operations: ● bADC_POWER_DOWN (Enable or Disable PowerDown Mode) ● bADC_SAMPLE_HOLD_ENABLE (Enable or disable Sample and Hold feature on each channel) ● If a conversion is currently in progress, it will be completed and then the new priority is become. 5'h11..1F: – Invalid value	R

Table 7.18 rADC_PRIORITY Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b4 to b0	bADC_Priority	ADC_VC[n] Priority with n=0..15 Determines the cutoff point for priority mode and round robin arbitration for ADC_VC[n] 5'h00: ADC_VC priority is handled in round robin mode for all channels. 5'h01: VC0 is high priority, Rest of channels are in round robin mode. 5'h02: VC0..1 are high priority, VC2..15 are in round robin mode. 5'h03: VC0..2 are high priority, VC3..15 are in round robin mode. 5'h04: VC0..3 are high priority, VC4..15 are in round robin mode. 5'h05: VC0..4 are high priority, VC5..15 are in round robin mode. 5'h06: VC0..5 are high priority, VC6..15 are in round robin mode. 5'h07: VC0..6 are high priority, VC7..15 are in round robin mode. 5'h08: VC0..7 are high priority, VC8..15 are in round robin mode. 5'h09: VC0..8 are high priority, VC9..15 are in round robin mode. 5'h0A: VC0..9 are high priority, VC10..15 are in round robin mode. 5'h0B: VC0..10 are high priority, VC11..15 are in round robin mode. 5'h0C: VC0..11 are high priority, VC12..15 are in round robin mode. 5'h0D: VC0..12 are high priority, VC13..15 are in round robin mode. 5'h0E: VC0..13 are high priority, VC14..15 are in round robin mode. 5'h0F: VC0..14 are high priority, VC15 is in round robin mode. 5'h10: All VC[n], with n=0..15, are in high priority mode, arbitrated by VC[n] number 5'h11..1F: Others invalid selection	R/W

7.4.1.17 rADC_CONFIG — ADC Configuration

Address: 4006 5040h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	bADC_DMA	bADC_POWER_DOWN	bADC_SAMPLE_HOLD_ENABLE		
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Table 7.19 rADC_CONFIG Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b5	Reserved	Not used	R
b4	bADC_DMA	DMA channel Enable 1: DMA channel 0 and 1 enable 0: DMA channel 0 and 1 disable All current DMA request are forced in idle state, after DMA acknowledge reception for current access.	R/W
b3	bADC_POWER_DOWN	Enable or Disable PowerDown Mode In power down mode, ADCs cannot be used. This bit must be cleared before configuration of Virtual channel ADC_VC[n] with n = 0..15 1: Configure Analog A/D converter ADC1 and ADC2 in power down mode 0: Configure Analog A/D converter ADC1 and ADC2 in operation mode. When a CPU write is detected, we have the following action: - Set bADC_BUSY until end of ADCs configuration change. - Clear bADC_TrigEnable bit (Trigger disable) in all rADC_VC[n] registers to 0, with n = 0..15. - Clear bADC_PENDING_VC[n] bit (operation pending on Virtual channel ADC_VC[n]) in rADC_PENDING register to 0, with n = 0..15. - Clear bADC_FORCE_VC[n] bit (Force start of operation on Virtual channel ADC_VC[n]) in rADC_FORCE register to 0, with n = 0..15. - Set bADC_RR_Pointer bits to 5'h10 in rADC_PRIORITY register (ADC_VC0 is highest round robin priority. If a conversion is currently in progress, it will be completed and then the new priority is become). - Force virtual state machine in idle state at the end of current operation.	R/W

Table 7.19 rADC_CONFIG Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b2 to b0	bADC_SAMPLE_HOLD_ENABLE	Enable or disable Sample and Hold feature on each channel ADC1 and ADC2 convertor are coupled by virtual machine ADC_VC[n] with n = 0..15, the same function (sample and hold feature enable or disable) is configured on both ADCs Sample and Hold feature is associated directly with following allocation bits: Bit [0] configures- ADC1 and ADC2 physical channel6 Bit [1] configures- ADC1 and ADC2 physical channel7 Bit [2] configures- ADC1 and ADC2 physical channel8 0: Analog inputs connect multiplexer circuit directly and sample/hold circuit is not used (through mode). 1: Sample/hold function is available. When a CPU write is detected, we have the following action: - Set bADC_BUSY until end of ADCs configuration change. - Clear bADC_TrigEnable bit (Trigger disable) in all rADC_VC[n] registers to 0, with n = 0..15. - Clear bADC_PENDING_VC[n] bit (operation pending on ADC_VC[n]) in rADC_PENDING register to 0, with n = 0..15. - Clear bADC_FORCE_VC[n] bit (Force start of operation on ADC_VC[n]) in rADC_FORCE register to 0, with n = 0..15. - Force virtual state machine in idle state at the end of current operation.	R/W

7.4.1.18 rADC_ACQS — ADC Control Sample and Hold

Address: 4006 50ACh

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	bADC_TSHOH			bADC_TSHSET		bADC_TSHSAMP				
Value after reset	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0

Table 7.20 rADC_ACQS Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b10	Reserved	Read as “0”	R
b9 to b7	bADC_TSHOH	ADC Sample and Hold SHOUT to SHCNT hold time: t_{SHOH} Provides the number of ADC_CLK clocks to obtain the 300 ns of hold time between a falling edge of SHCNT and rising edge of SHOUT in the sample and hold circuit for ADC1 and ADC2. t_{SHOH}: Minimum Value is 300 ns. 3’b000: Invalid value 3’b001: Invalid value 3’b010: 2 cycles when 4 MHz ≤ f_{ADC_CLK} ≤ 6.66 MHz 3’b011: 3 cycles when 6.66 MHz < f_{ADC_CLK} ≤ 10 MHz … … 3’b110: 6 cycles when 16.66 MHz < f_{ADC_CLK} ≤ 20 MHz 3’b111: Invalid value	R/W
b6, b5	bADC_TSHSET	Delay from ADC Sample and Hold SHOUT to CONV setup time: t_{SHSET} Provides the number of ADC_CLK clocks to obtain the 100 ns of setup time between SHOUT and CONV in the sample and hold circuit for ADC1 and ADC2. t_{SHSET}: Minimum Value is 100 ns. 2’b00: Invalid value 2’b01: 1 cycle when 4MHz ≤ f_{ADC_CLK} < 10 MHz 2’b10: 2 cycles when 10MHz ≤ f_{ADC_CLK} ≤ 20 MHz 2’b11: Invalid value	R/W
b4 to b0	bADC_TSHSAMP	ADC Sample and Hold sampling time: t_{SHSAMP} Provides the number of ADC_CLK clocks to obtain the window sampling time in the sample and hold circuit for ADC1 and ADC2. t_{SHSAMP}: Minimum Value is 300 ns. 5’h0: Invalid value 5’h1: Invalid value 5’h2: 2 cycles 5’h3: 3 cycles … … 5’h7: 7 cycles 5’h8: 8 cycles … … 5’h1f: 31 cycles	R/W

7.4.1.19 rADC_MASKLOCK[n] — Mask Data Locked [n] (n = 0..3)

Address: 4006 50B0h +4h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC_MASKLOCK															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.21 rADC_MASKLOCK[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b16	Reserved	Read as “0”.	R
b15 to b0	bADC_MASKLOCK	Mask Data Locked Enable or disable a copy from respective registers: • rADC1_DATA[n] to rADC1_DATALOCK[n] with n = 0..15 • rADC2_DATA[n] to rADC2_DATALOCK[n] with n = 0..15 When after scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run and all following conditions are satisfied: • Event detection: “Data copy into data lock registers” • Configured with bADC_Mode (rADC_VC[n] register) = 2’b11. Following actions are executed: • bADC1_ChannelSel selects rADC_MASKLOCK0..3 registers used to enable or disable data copy from respective rADC1_DATA[n] to rADC1_DATALOCK[n] with n = 0..15 • bADC2_ChannelSel selects rADC_MASKLOCK0..3 registers used to enable or disable data copy from respective rADC2_DATA[n] to rADC2_DATALOCK[n] with n = 0..15 Each bADC_MASKLOCK[n] is associated directly with following allocation bits: • Bit [n]: bADC_MASKLOCK[n] is the mask data locked of following register. - rADC1_DATA[n] if selected by bADC1_ChannelSel - rADC2_DATA[n] if selected by bADC2_ChannelSel 0: No action, disable copy 1: bADC1_ChannelSel, enable copy from rADC1_DATA[n] to rADC1_DATALOCK[n] bADC2_ChannelSel, enable copy from rADC2_DATA[n] to rADC2_DATALOCK[n]	R/W

7.4.1.20 rADC_VC[n] — ADC Control Register for Virtual Channel [n] (n = 0..15)

Address: 4006 50C0h + 4h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	bADC_DMA_Request		bADC2_Enable
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bADC1_Enable	bADC_Continuous	bADC_TrigEnable	bADC_TrigSel					bADC_Mode		bADC2_ChannelSel			bADC1_ChannelSel		
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.22 rADC_VC[n] Register Contents (1/5)

Bit Position	Bit Name	Function	R/W
b31 to b19	Reserved	Read as “0”	R
b18, b17	bADC_DMA_Request	ADC DMA Request on Virtual Channel ADC_VC[n] with n = 0..15. When after scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run and all following conditions are satisfied: • Event detection: “Data copy into data lock registers” • Configured with bADC_Mode (rADC_VC[n] register) = 2’b11. • bADC_DMA_Request[1] is set to 1 • bADC_DMA bit in rADC_CONFIG register is enabled ADC DMA requests are started and run until detection end of DMA transfer. The ADC DMA channel selected depends on bADC_DMA_Request[0] bit status. 2’b00: ADC DMA Disable - No DMA coupling 2’b10: ADC DMA0 Enable - Set bADC_DMA0_RUNNING in rADC_PENDING 2’b11: ADC DMA1 Enable - Set bADC_DMA1_RUNNING in rADC_PENDING	R/W
b16	bADC2_Enable	ADC2 Enable on Virtual Channel ADC_VC[n] with n = 0..15. If set, enable a conversion, sample and hold or other operation on ADC2. For each virtual channel ADC_VC[n] with n = 0..15 0: ADC2 Disable – When ADC_VC[n] event selected is set (rising edge detection on trigger or CPU changed bADC_FORCE_VC[n] bit to 1 by setting bADC_SETFORCE_VC[n] bit) and when after scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run. The function selected and controlled by bADC_Mode, bADC2_ChannelSel (convert, sample and hold or other operation) are not executed on ADC2. 1: ADC2 Enable: – When ADC_VC[n] event selected is set (rising edge detection on trigger or CPU changed bADC_FORCE_VC[n] bit to 1 by setting bADC_SETFORCE_VC[n] bit) and when after scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run. The function selected and controlled by bADC_Mode, bADC2_ChannelSel (convert, sample and hold or other operation) must be executed on ADC2 ***Note*)** If not interrupt only mode and if ADC1 and ADC2 are both disabled, then dedicated End of Command (EoC) is anyway generated. ***Caution*)** • When ADC2 is not implemented (ex. RZ/N1S), this bit has not effect on virtual channel and is set to “0” by hardware. • When ADC2 is not connected (RZ/N1D-324), the physical channels are gated to “0”, no conversion or sample and hold on this channel are possible.	R/W

Table 7.22 rADC_VC[n] Register Contents (2/5)

Bit Position	Bit Name	Function	R/W
b15	bADC1_Enable	ADC1 Enable on Virtual Channel ADC_VC[n] with n = 0..15 Same as bADC2_Enable	R/W
b14	bADC_Continuous	ADC Continuous Mode on Virtual Channel ADC_VC[n] with n = 0..15. For each virtual channel ADC_VC[n] with n = 0..15 0: Single mode 1: Continuous mode ***Note*)** **In single mode:** • ADC operation (conversion, sample and hold or other) is to be performed only once. • bADC_TrigEnable bit is automatically cleared to 0 when operation ends on the selected channel. • bADC_FORCE_VC[n] bit is automatically cleared to 0 when operation ends on the selected channel. **In continuous mode:** • ADC operation is continuously performed for the selected channels in sequence until bADC_TrigEnable and bADC_FORCE_VC[n] bits are cleared by software or hardware reset or firmware ADC reset. • No change on bADC_TrigEnable and bADC_FORCE_VC[n] bits when operation ends on the selected channel. **In all case, single mode, continuous mode or other operation on ADC_VC[n] will be executed when all following conditions are satisfied:** • The dedicated ADC_VC[n] event selected is set (rising edge detection on trigger or CPU changed bADC_FORCE_VC[n] bit to 1 by setting bADC_SETFORCE_VC bit). • When after scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run. • Respective bADC2_Enable and bADC1_Enable of ADC2 and ADC1 is enabled.	R/W
b13	bADC_TrigEnable	Trigger Enable on Virtual Channel ADC_VC[n] with n = 0..15. For each virtual channel ADC_VC[n] with n = 0..15 0: Trigger disable – A virtual channel is in idle state and cannot activate by bADC_TrigSel selected trigger 1: Trigger enable – When this bit is set, a trigger condition configured by bADC_TrigSel will start an operation (conversion, sample and hold or other) when ADC_VC[n] is selected by ADC state machine to run. ***Note*)** • In single mode (bADC_Continuous bit is set to 0), this bit is automatically cleared to 0 when operation ends on the selected channel. • Cleared to 0 when a CPU write to rADC_CONFIG register starts the following operations: – bADC_POWER_DOWN (Enable or Disable PowerDown Mode) – bADC_SAMPLE_HOLD_ENABLE (Enable or disable Sample and Hold feature on each channel)	R/W

Table 7.22 rADC_VC[n] Register Contents (3/5)

Bit Position	Bit Name	Function	R/W
b12 to b8	bADC_TrigSel	Trigger Select on Virtual Channel ADC_VC[n] with n = 0..15. Configures which trigger will set to initiate an operation (conversion, sample and hold or other) to start once priority is given to ADC_VC[n] by ADC state machine. Only rising edge of trigger is used to start an event on ADC_VC[n]. Trigger selected on virtual channel ADC_VC[n] with n = 0..15: 5'h00 .. 5'h0f: Not used 5'h10: Select iADC_EOC_VC0 ... ... 5'h1f: Select iADC_EOC_VC15 ***Note*)** Each rising edge of iADC_EOC_VC[n] is trigger for bADC_INTSTATUS0_VC[n] with n = 0..15.	R/W
b7, b6	bADC_Mode	ADC Mode on Virtual Channel ADC_VC[n] with n = 0..15. Select the functions mode of virtual channel. The functions selected below depends on bADC2_Enable and bADC1_Enable for respective ADC2 and ADC1. See more details in bADC1_ChannelSel and bADC2_ChannelSel fields description. Mode is selected on virtual channel ADC_VC[n] with n = 0..15 if respective bADC2_Enable and bADC1_Enable of ADC2 and ADC1 is enabled: **[Event selected: Physical channel Convert]** • 2'b00: Physical channel convert – The bADC1_ChannelSel and bADC2_ChannelSel fields select the physical channel to convert once priority is given to ADC_VC[n] by ADC state machine to run. – At the end of command, the resulting value is stored in the rADC1_DATA[n] and rADC2_DATA[n] register. – An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n]. **[Event Selected: Sample and Hold, physical channel 6..8 only]** • 2'b01: Sample and hold – The bADC1_ChannelSel and bADC2_ChannelSel fields select the physical channel to sample and hold once priority is given to ADC_VC[n] by ADC state machine to run. – This feature is only available on physical channel ADC1_IN[8:6] and ADC2_IN[8:6]. – An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n]. ***Caution*)** Concerning Sample and hold on ADC1 and ADC2 • This feature is only available on physical channel 6..8 • Setting this field to 2'b01 on physical channel 0..4 is not effect. **[Event Selected: Only interrupt and end of command]** • 2'b10: – No action on physical channel. – An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n].	R/W

Table 7.22 rADC_VC[n] Register Contents (4/5)

Bit Position	Bit Name	Function	R/W
		[Event Selected: Data copy into data lock registers] • 2'b11: – No action on physical channel. – The bADC1_ChannelSel and bADC2_ChannelSel fields select the mask data locked (rADC_MASKLOCK0..3 registers) used to enable or disable copy from rADC[m]_DATA0..15 to rADC[m]_DATALOCK0..15 (with n = 0..1) once priority is given to ADC_VC[n] by ADC state machine to run. – An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n]. – A potential ADC DMA requests are started and run until detection end of DMA transfer. The ADC DMA channel selected depends on bADC_DMA_Request[1:0] bit status.	
b5 to b3	bADC2_ChannelSel	ADC2 Function on virtual channel ADC_VC[n] with n = 0..15. Several functions are available depending on bADC2_Enable and bADC_Mode bits. **[Event selected: None]** • bADC2_Enable = 0: – No action **[Event selected: Physical channel Convert]** • bADC2_Enable = 1 and bADC_Mode = 2'b00 – Selects the physical channel ADC2_IN[8:6,4:0] to be converted when ADC_VC[n] is received by the ADC state machine to run. – At the end of command, the resulting value is stored in the rADC2_DATA[n] register. – An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n]. • Physical channels selected to be converted by ADC2_ChannelSel fields: 3'b000: Select physical channel ADC2_IN0 ... ... 3'b100: Select physical channel ADC2_IN4 3'b101: Select physical channel ADC2_IN6 ... . 3'b111: Select physical channel ADC2_IN8 **[Event Selected: Sample and Hold, physical channel 6..8 only]** • bADC2_Enable = 1 and bADC_Mode = 2'b01 – Selects the physical channels ADC2_IN[8:6] to be sampled and hold when ADC_VC[n] is received by the ADC state machine to run. – An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n]. – Physical channels selected to be sampled and hold by bADC2_ChannelSel field: • bADC2_ChannelSel[2]: 1: Sample and hold on physical channel ADC2_IN8 0: No sample and hold • bADC2_ChannelSel[1]: 1: Sample and hold on physical channel ADC2_IN7 0: No sample and hold • bADC2_ChannelSel[0]: 1: Sample and hold on physical channel ADC2_IN6 0: No sample and hold ***Example***) If ADC2_ChannelSel[2:0]: 3'b101 Simultaneous sample and hold on ADC2_IN8, ADC2_IN6	R/W

Table 7.22 rADC_VC[n] Register Contents (5/5)

Bit Position	Bit Name	Function	R/W
		[Event Selected: Only interrupt and end of command] • bADC2_Enable = 1 and bADC_Mode = 2'b10: – No action on physical channel. – An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n] • bADC2_ChannelSel: 3'bxxx: Reserved **[Event Selected: Data copy into data lock registers]** • bADC2_Enable:1'b1 and bADC_Mode: 2'b11: – No action on physical channel. – The bADC2_ChannelSel field selects the mask data locked (rADC_MASKLOCK0..3 registers) used to enable or disable a copy from each rADC2_DATA0..15 to rADC2_DATALOCK0..15 registers once priority is given to ADC_VC[n] by ADC state machine to run. – An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n]. – A potential ADC DMA requests are started and run until detection end of DMA transfer. The ADC DMA channel selected depends on bADC_DMA_Request[1:0] bit status. • Mask data locked register selected by bADC2_ChannelSel field: 3'b000: Select rADC_MASKLOCK0 register mask. 3'b001: Select rADC_MASKLOCK1 register mask. 3'b010: Select rADC_MASKLOCK2 register mask. 3'b011: Select rADC_MASKLOCK3 register mask. 3'b1xx: Reserved ***Caution*)** • When ADC2 is not implemented (ex. RZ/N1S), these bits are not implementing and read as 0. • When ADC2 is not connected (RZ/N1D-324), the physical channels are gated to "0", no conversion or sample and hold on this channel are possible	
b2 to b0	bADC1_ChannelSel	ADC1 Function on virtual channel ADC_VC[n] with n = 0..15. Same as bADC2_ChannelSel	R/W

7.4.1.21 rADC1_DATA[n] — ADC1 Conversion Data of Virtual Channel [n] (n = 0..15)

Address: 4006 5100h + 4h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bADC1_DATA_Update	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bADC1_DATA											
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.23 rADC1_DATA[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31	bADC1_DATA_Update	Data has been updated since the last copy from rADC1_DATA[n] to respective rADC1_DATALOCK[n] register with n = 0..15 For each virtual channel ADC_VC[n] with n = 0..15: 0: Data does not be updated in bADC1_DATA 1: Data has been updated in bADC1_DATA ***Note*)** This bit is cleared to 0 when a copy from rADC1_DATA[n] to respective rADC1_DATALOCK[n] register is running when mask enable.	R
b30 to b12	Reserved	Read as “0”.	R
b11 to b0	bADC1_DATA	ADC1 conversion data for virtual channel ADC_VC[n] channel, with n = 0..15 After the ADC1 completes a conversion required on virtual channel ADC_VC[n], the digital result is placed in the corresponding rADC1_DATA[n] register, with n = 0..15. *Example*) • If ADC_VC5 is configured to convert the physical channel ADC1_IN0, the completed result of that conversion will be placed in rADC1_DATA5.	R

7.4.1.22 rADC1_DATALOCK[n] — ADC1 DataLock[n] Register (n = 0..15)

Address: 4006 5180h +4h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bADC1_DATALOCK_Update	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bADC1_DATALOCK											
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.24 rADC1_DATALOCK[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31	bADC1_DATALOCK_Update	Copy locked of bADC1_DATA_Update bit of rADC1_DATA[n] register, with n = 0..15 When an event: “Data copy into data lock registers” is running by ADC_VC state machine, this register is updated in under control of rADC_MASKLOCK0..3 registers. ***Note*)** After copy of bADC1_DATA_Update to bADC1_DATALOCK_Update (copy not masked), bADC1_DATA_Update is cleared by hardware.	R
b30 to b12	Reserved	Read as “0”	R
b11 to b0	bADC1_DATALOCK	Copy locked of bADC1_DATA field of rADC1_DATA[n] register, with n = 0..15 When an event: “Data copy into data lock registers” is running by ADC_VC state machine, this register is updated in under control of rADC_MASKLOCK0..3 registers. Data locked can be read by CPU and DMA in parallel with new acquisition in running on rADC1_DATA[n] register.	R

7.4.2 Register Description ADC2

CAUTION

These registers are only implemented in RZ/N1D, read as 0 in other.

7.4.2.1 rADC2_DATA[n] — ADC2 Conversion Data of Virtual Channel [n] (n = 0..15)

Address: 4006 5140h + 4h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bADC2_DATA_Update	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bADC2_DATA											
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.25 rADC2_DATA[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31	bADC2_DATA_Update	Data has been updated since the last copy from rADC2_DATA[n] to respective rADC2_DATALOCK[n] register with n = 0..15 For each virtual channel ADC_VC[n] with n = 0..15: 0: Data does not be updated in bADC2_DATA 1: Data has been updated in bADC2_DATA ***Note*)** This bit is cleared to 0 when a copy from rADC2_DATA[n] to respective rADC2_DATALOCK[n] lock register is running when mask enable.	R
b30 to b12	Reserved	Read as “0”	R
b11 to b0	bADC2_DATA	ADC2 conversion data for virtual channel ADC_VC[n] channel, with n = 0..15 After the ADC2 completes a conversion required on virtual channel ADC_VC[n], the digital result is placed in the corresponding rADC2_DATA[n] register, with n = 0..15. *Example*) • If ADC_VC7 is configured to sample the physical channel ADC2_IN3, the completed result of that conversion will be placed in rADC2_DATA7.	R

7.4.2.2 rADC2_DATALOCK[n] — ADC2 DataLock[n] Register (n = 0..15)

Address: 4006 51C0h +4h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bADC2_DATALOCK_Update	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bADC2_DATALOCK											
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.26 rADC2_DATALOCK[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31	bADC2_DATALOCK_Update	Copy locked of bADC2_DATA_Update bit of rADC2_DATA[n] register, with n = 0..15 When an event: “Data copy into data lock registers” is running by ADC_VC state machine, this register is updated in under control of rADC_MASKLOCK0..3 registers. ***Note*)** After copy of bADC2_DATA_Update to bADC2_DATALOCK_Update (copy not masked), bADC2_DATA_Update is cleared by hardware.	R
b30 to b12	Reserved	Read as “0”.	R
b11 to b0	bADC2_DATALOCK	Copy locked of bADC2_DATA field of rADC2_DATA[n] register, with n = 0..15 When an event: “Data copy into data lock registers” is running by ADC_VC state machine, this register is updated in under control of rADC_MASKLOCK0..3 registers. Data locked can be read by CPU and DMA in parallel with new acquisition in running on rADC2_DATA[n] register.	R

7.5 Operation

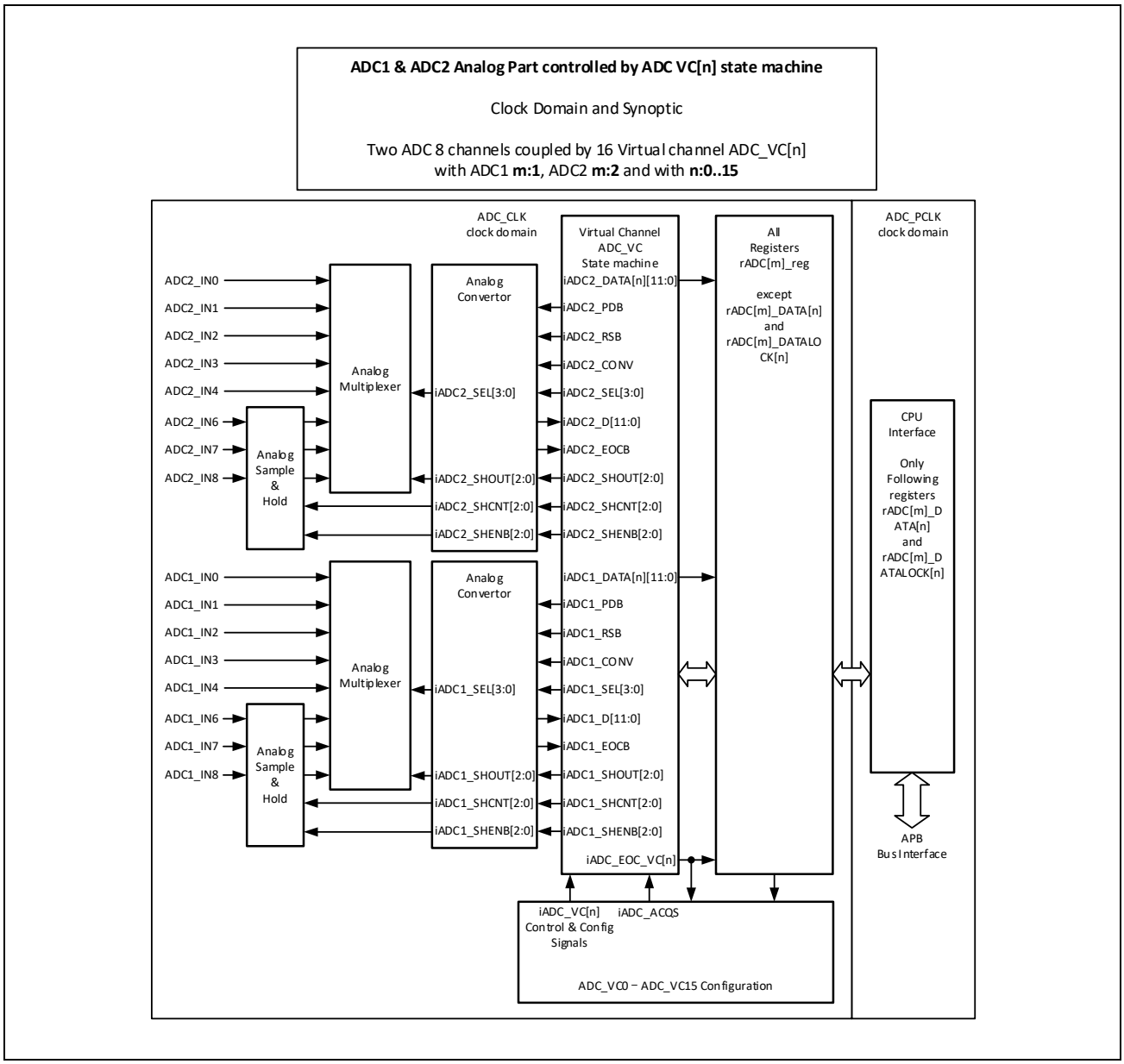

Figure 7.3 ADC Controller Synoptic

7.5.1 Virtual Channel ADC_VC Principle Operation

Contrary to standard ADC types, this ADC is not sequencer based. Instead, it is ADC_VC (Virtual channel), with based. The term ADC_VC is the configuration set defining an operation on a single or dual channel. In that set, we have the following configurations:

- The trigger source that starts an operation
- The physical channel to process
- Convert or sample and hold mode
- DMA transaction on selected “End of Conversion” signal
- Set an interrupt on selected “End of Conversion” signal
- Update Data lock register

Each ADC_VC is independently configured and can have any combination of the trigger, channel, and mode.

Multiple ADC_VCs can be configured for the same trigger, channel, and/or sample and hold as desired. This provides a very flexible means of configuring conversions ranging from individual samples of different channels with different triggers, to oversampling the same channel using a single trigger, to creating your own series of conversions of different channels all from a single trigger.

For example, ADC_VC[n]:

The trigger source of ADC_VC[n] is configured by a combination of the bADC_TrigSel, bADC_TrigEnable, and bADC_Continuous fields in the rADC_VC[n] register. Software can also force an ADC_VC[n] event with the rADC_FORCE register. The physical channel is configured with the bADC1_ChannelSel, bADC2_ChannelSel and bADC_Mode fields in the rADC_VC[n] register, and sample and hold window size with bADC_TSHSAMP bits in rADC_ACQS register.

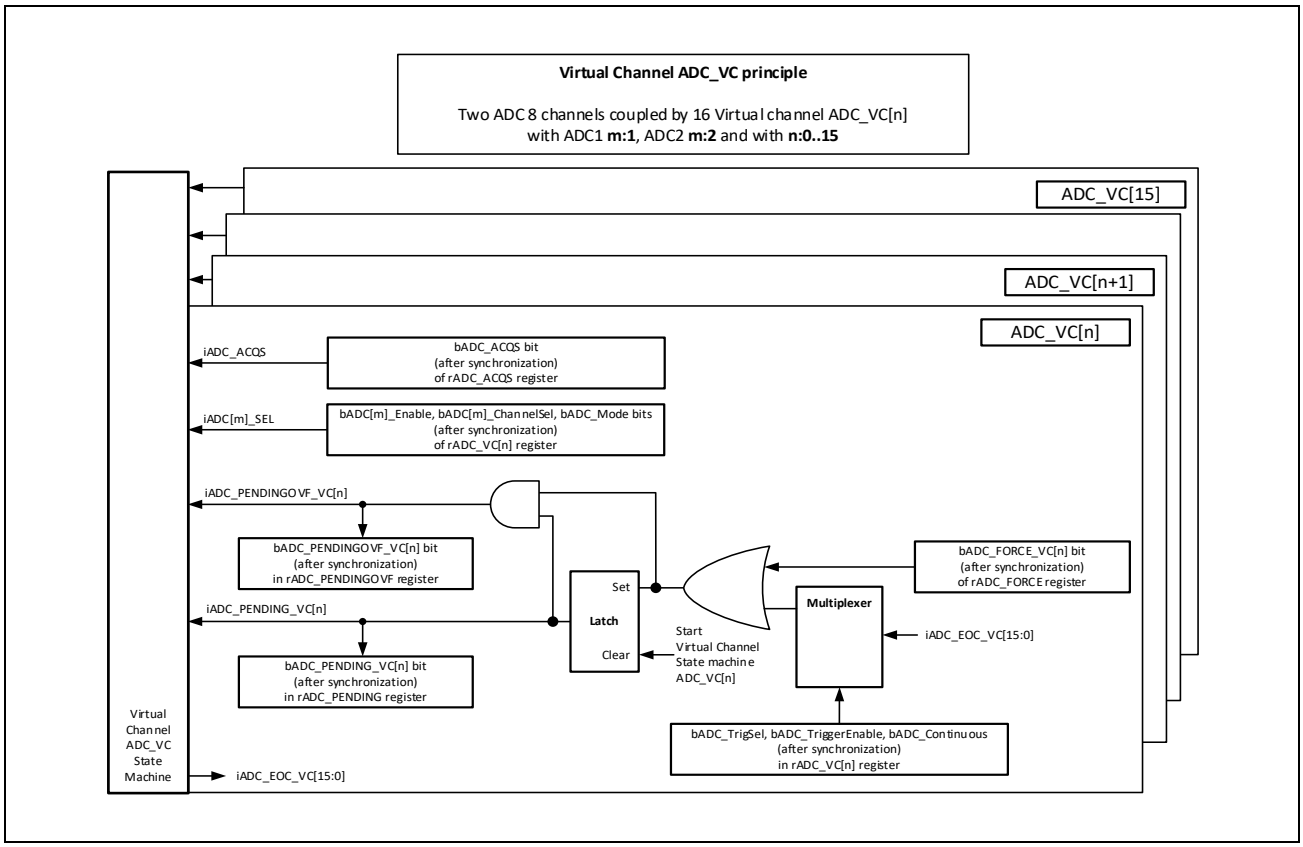

Figure 7.4 Virtual Channel ADC_VC Architecture

For example, you can configure a single conversion on channel ADC1_IN1 to occur when the iADC_EOC_VC4 is set. For this we'll use a virtual channel ADC_VC's. Then, setup one of the ADC_VC's using its rADC_VC[n] register. It makes no difference which ADC_VC[n] is chosen, so it uses ADC_VC8 as example.

It also makes no difference which ADC trigger we choose, so we'll use iADC_EOC_VC4. The fastest allowable sample window for the ADC is 6 cycles. Choosing the fastest time for the sample window and channel ADC1_IN1 for the channel to convert, we'll set the bADC_TSHSAMP field to 6, the bADC_TrigSel field to 5'h14, bADC_TrigEnable and bADC1_Enable in enable mode.

The resulting values written into the registers will be:

rADC_VC8:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b0 (ADC2 Disable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h14 (Select trigger iADC_EOC_VC4)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h1 (ADC1_IN1 channel selected)
- bADC2_ChannelSel: 3'hx (Not used)

rADC_ACQS:

- bADC_TSHSAMP: 5'h6 (Minimum value)

When configured as such, a single conversion of physical channel ADC1_IN1 will be started on a use iADC_EOC_VC4 event with the resulting value stored in the rADC1_DATA8 register.

In same time, we would like to have a conversion of physical ADC1_IN2 with oversampled by 4X on same trigger, we will use for this ADC_VC9..12.

The resulting values written into the registers will be:

rADC_VC8:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b0 (ADC2 Disable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h14 (Select trigger iADC_EOC_VC4)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h1 (ADC1_IN1 channel selected)
- bADC2_ChannelSel: 3'hx (Not used)

rADC_VC9..12 (Configured with same value):

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b0 (ADC2 Disable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h14 (Select trigger iADC_EOC_VC4)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h2 (ADC1_IN2 channel selected)
- bADC2_ChannelSel: 3'hx (Not used)

rADC_ACQS:

- bADC_TSHSAMP: 5'h6 (Minimum value)

rADC_PRIORITY:

- bADC_Priority: 5'h0
  Round Robin arbitration on all ADC_VC0..15

When configured as such, a single conversion of physical channel ADC1_IN1 will be started on a use iADC_EOC_VC4 event with the resulting value stored in the rADC1_DATA8 register.

Four conversions (oversampling 4X) of physical channel ADC1_IN2 will be started on a use iADC_EOC_VC4 event with the resulting value stored in the rADC1_DATA9..12 registers.

Additionally, in same time, we would like to have a conversion of physical ADC1_IN [8:6] and ADC2_IN[8:6] with simultaneous sample & hold started by use iADC_EOC_VC7, we will use for this ADC_VC0 for sample & hold event and ADC_VC1..3 for convert ADC1_IN [8:6] and ADC2_IN[8:6]. The sample window for the ADC2 is 12 cycles.

For this acquisition, the priority is high.

The resulting values written into the registers will be:

rADC_VC8:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b0 (ADC2 Disable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h14 (Select trigger iADC_EOC_VC4)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h1 (ADC1_IN1 channel selected)
- bADC2_ChannelSel: 3'hx (Not used)

rADC_VC9..12 (Configured with same value):

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b0 (ADC2 Disable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h14 (Select trigger iADC_EOC_VC4)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h2 (ADC1_IN2 channel selected)
- bADC2_ChannelSel: 3'hx (Not used)

rADC_VC0:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b1 (ADC2 Enable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h17 (Select trigger iADC_EOC_VC7)
- bADC_Mode: 2'b01 (Sample and hold)
- bADC1_ChannelSel: 3'h7 (ADC1_IN[8:6] channel selected)
- bADC2_ChannelSel: 3'h7 (ADC2_IN[8:6] channel selected)

rADC_VC1:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b1 (ADC2 Enable)
- bADC_Continuous: 1'b0 (Single conversion)

- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h17 (Select trigger iADC_EOC_VC7)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h5 (ADC1_IN6 channel selected)
- bADC2_ChannelSel: 3'h5 (ADC2_IN6 channel selected)

rADC_VC2:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b1 (ADC2 Enable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h17 (Select trigger iADC_EOC_VC7)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h6 (ADC1_IN7 channel selected)
- bADC2_ChannelSel: 3'h6 (ADC2_IN7 channel selected)

rADC_VC3:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b1 (ADC2 Enable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h17 (Select trigger iADC_EOC_VC7)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h7 (ADC1_IN8 channel selected)
- bADC2_ChannelSel: 3'h7 (ADC2_IN8 channel selected)

rADC_ACQS:

- bADC_TSHSAMP: 5'h0C

rADC_PRIORITY:

- bADC_Priority: 5'h4
  ADC_VC0..3 are high priority,
  ADC_VC4..15 are in round robin mode.

When configured as such, a single conversion of physical channel ADC1_IN1 will be started on a use iADC_EOC_VC4 event with the resulting value stored in the rADC1_DATA8 register.

Four conversions (oversampling 4X) of physical channel ADC1_IN2 will be started on a use iADC_EOC_VC4 event with the resulting value stored in the rADC1_DATA9..12 registers.

A sample and hold will be started on a use iADC_EOC_VC7 event and after that, three conversions (on previous sample and hold values) of physical channel ADC1_IN[8:6] and ADC2_IN[8:6] will be started with the resulting value stored in the rADC1_DATA1..3 and rADC2_DATA1..3 registers.

In example above, we can control of one ADC instance has no effect on the other one. So, they can be fully used simultaneously. The only common parts are external iADC_EOC_VC7.

7.5.2 Electric ADC Model and Acquisition Sample

External drivers vary in their ability to drive an analog signal quickly and effectively. Some circuits require longer times to properly transfer the charge into the sampling capacitor of an ADC.

To address this, the ADC supports control over the sample window length. bADC_TSHSAMP bit field in rADC_ACQS register has a 5 bits field that determines the sample and hold (S/H) window size t_{SHSAMP} for each ADC.

The value written to this field is one the number of cycles desired for the sampling window of ADC. The minimum time value of sample cycles allowed is 300 ns. The total sampling time is found by adding the sample window size to the conversion time of the ADC.

The figure below describes in detail the equivalent circuit of the input block and the sampling error generated by an external circuit, where:

- Analog input equivalent resistance (R_{IN}) = On resistance of multiplexer + On resistance of sampling SW
- Analog input equivalent capacitance (C_{IN}) = internal parasitic capacitance + sampling capacitance
- Impedance of signal source (R_s) = Output resistance of signal source + external resistance (low pass filter)
- External capacitance (C_e) = External parasitic capacitance + External capacitance (low pass filter)

The accuracy on ADC characteristics described on Electrical Characteristics doesn't include sampling error was generated with the effects by external circuits.

It is necessary to charge the analog input equivalent capacitance (C_{IN}) through the analog input equivalent resistance (R_{IN}) within 0.1LSB of input voltage and sampling time (t_{SHSAMP}) on sampling.

If the impedance of the signal source (R_s) is big, the charge time is insufficient for sampling and then sampling error occurs. Therefore, please decrease the impedance of signal source satisfactorily ($R_s < 300\ \Omega$, $C_e < 15$ pF).

(t_{SHSAMP}): Sample time window: 6 clocks periods of ADC_CLK frequency

- Min value: 300 ns with ADC_CLK = 20 MHz
- Max value: 1500 ns with ADC_CLK = 4 MHz

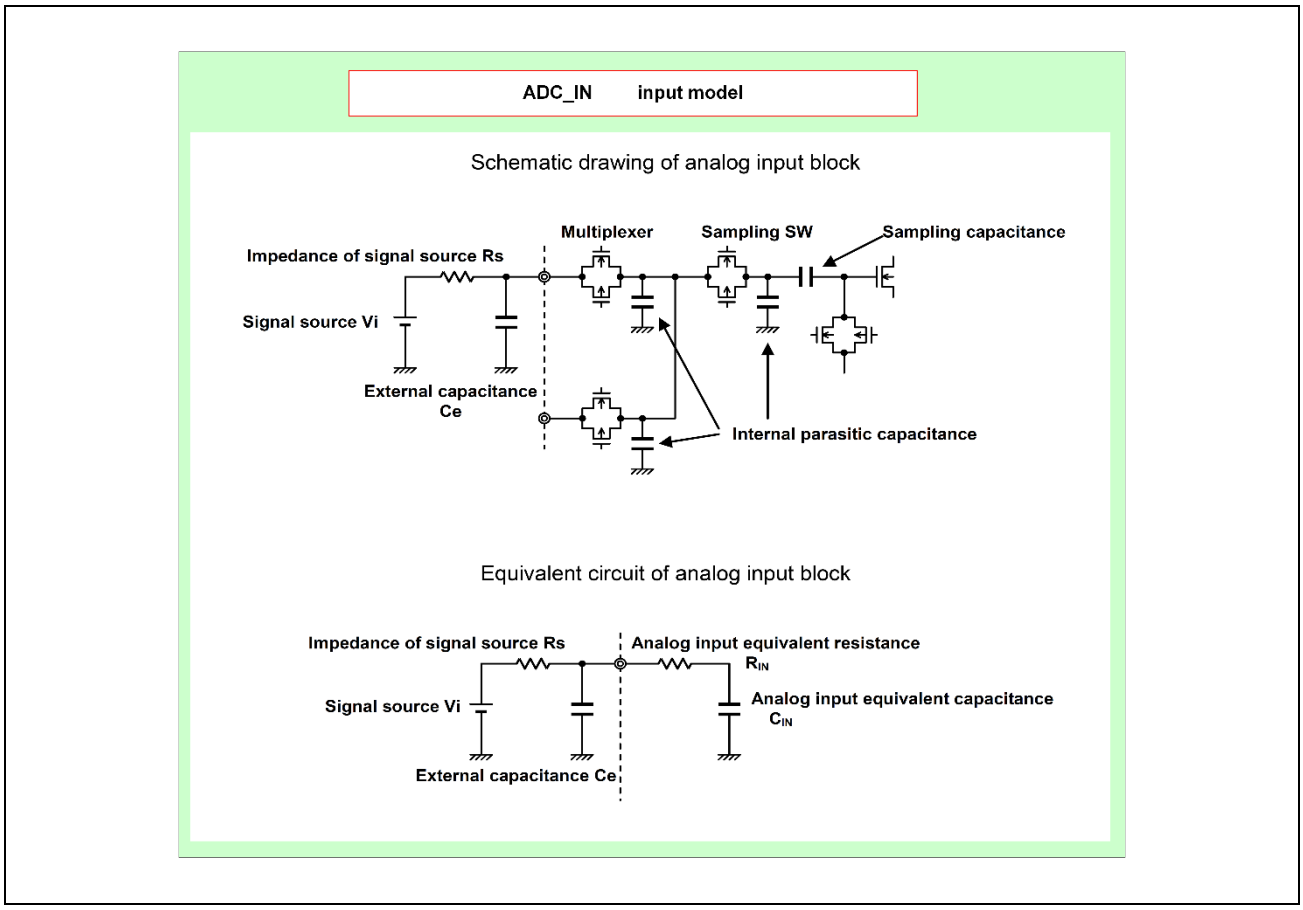

Figure 7.5 ADC Analog Model Input

Prefer to insert low pass filter into analog input for the elimination of unneeded signal with high frequency, ex. noise. Please select RC parameter of low pass filter (R_s and C_e) in accordance with the characteristics of the application.

On the application handling AC signals, it is necessary to decrease the impedance of the signal source fully. When AC characteristics are not important, you will decrease sampling error adding external capacitance with large capacity (C_e).

You can estimate sampling error as the following expression:

$$\text{Sampling error} \fallingdotseq \frac{C_{IN}}{Ce + C_{IN}} \times 4096\ [\text{LSB}]$$

If Ce is not so big, you can estimate sampling error as the following expression.

$$\text{Sampling error} \fallingdotseq \frac{C_{IN}}{Ce + C_{IN}} \times \exp\left(-\frac{t_{AS}}{Rs \times Ce + (R_s + R_{IN}) \times C_{IN}}\right) \times 4096\ [\text{LSB}]$$

When doing A/D conversion with switching multiplexer periodically and repeatedly, the sampling error as shown on the following expression occurs with the impedance of signal source (R_s) and the external capacitance (C_e)

$$\text{Sampling error} \fallingdotseq \left(\frac{C_{IN}}{Ce + C_{IN}} + \frac{R_S \times C_{IN}}{t_{SC}}\right) \times 4096\ [\text{LSB}]$$

t_{SC}: The period of the multiplexer scan, where multiplexer scan means the periodic and repeated A/D conversion with switching input of A/D converter.

For example, doing A/D conversion from ADC1_IN0 to ADC1_IN8 sequentially, and after doing ADC1_IN8, return to ADC1_IN0.

7.5.3 Trigger Selection and Event Management

Each ADC_VC[n] with n = 0..15 can be configured to start on one of many input triggers. Multiple ADC_VC[n] can be configured for the same channel if desired.

Following is a list of the available input triggers:

- CPU write (set bADC_SETFORCE_VC bit) in bADC_FORCE_VC[n] bit of rADC_FORCE register:
  - Force an ADC request (conversion, sample and hold or other operation, depending on configuration) to start once priority is given to ADC_VC[n]
- iADC_EOC_VC[n] event, End of Operation of ADC_VC[n]:
  - Driven by ADC_VC[n] state machine at End of Operation
  - This mode is useful if a continuous stream of conversions is desired from ADC_VC to another ADC_VC.
  - Configured by bADC_TrigEnable, bADC_TrigSel bits in rADC_VC[n] register
- Single mode:
  - In single mode, ADC operation (conversion, sample and hold or other) is to be performed only once.
  - bADC_TrigEnable bit is automatically cleared to 0 when operation ends on the selected channel.
  - bADC_FORCE_VC[n] bit is automatically cleared to 0 when operation ends on the selected channel.
  - Configured by bADC_Continuous: 1’b0 bit in rADC_VC[n] register
- Continuous mode:
  - In continuous mode, operation is continuously performed for the selected channels in sequence until bADC_TrigEnable and bADC_FORCE_VC[n] bits are cleared by software or hardware reset or firmware ADC reset.
  - No change on bADC_TrigEnable and bADC_FORCE_VC[n] bits when operation ends on the selected channel.
  - This mode is useful if a continuous stream of conversions is desired from ADC_VC
  - Configured by bADC_Continuous:1’b1 bit in rADC_VC[n] register

An event is received at bADC_PENDING_VC[n] bit of rADC_PENDING register of virtual channel ADC_VC[n], when we have:

- bADC_TrigEnable bit is set to 1’b1
  - Rising edge detection on trigger selected by bADC_TrigSel of virtual channel ADC_VC[n]
- CPU write (set bADC_SETFORCE_VC bit) in bADC_FORCE_VC[n] bit of rADC_FORCE register
  - Force an ADC request (conversion, sample and hold or other operation, depending on configuration) to start once priority is given to ADC_VC[n]
- An ADC event (conversion, sample and hold or other operation) is pending for ADC_VC[n]

The event pending in rADC_PENDING register (convert, sample and hold or other operation) must be executed on ADC1 and or ADC2 when:

- After scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run.

- If bADC1_Enable bit is set to 1, the respective operation controlled by bADC_Mode and bADC1_ChannelSel will be executed on ADC1.
- If bADC2_Enable bit is set to 1. the respective operation controlled by bADC_Mode and bADC2_ChannelSel will be executed on ADC2

- In same time, concerning event pending in rADC_PENDING register, we have:
  - The bADC_PENDING_VC[n] bit of rADC_PENDING register will be automatically cleared when the respective ADC_VC[n] operation is started.
  - In single mode, ADC operation (conversion, sample and hold or other) is to be performed only once. bADC_TrigEnable bit is automatically cleared to 0 when operation ends on the selected channel. bADC_FORCE_VC[n] bit is automatically cleared to 0 when operation ends on the selected channel.
  - In continuous mode, operation is continuously performed for the selected channels in sequence until bADC_TrigEnable and bADC_FORCE_VC[n] bits are cleared by software or hardware reset or firmware ADC reset.

An event overflow is received bADC_PENDINGOVF_VC[n] bit of rADC_PENDINGOVF register of virtual channel ADC_VC[n], when we have:

- A new ADC_VC[n] event was generated while an existing event was already pending.
- An overflow condition does not stop ADC_VC[n] events from being processed. It simply is an indication that a trigger was missed
- Writing 1 in bADC_PENDINGCLROVF_VC[n] bit of rADC_PENDINGCLROVF register will clear the bADC_PENDINGOVF_VC[n] bit.

See **Figure 7.4, Virtual Channel ADC_VC Architecture**.

7.5.4 Physical Channel Selection

Each ADC_VC[n] with ADC1 m:1, ADC2 m:2 and n = 0..15 can be configured to convert any of the available ADC[m]_IN[8:6,4:0] input channels.

- When an ADC_VC[n] is configured in convertor mode (bADC_Mode is set to 2’b00)
  - bADC[m]_ChannelSel (bADC1_ChannelSel and bADC2_ChannelSel) field of the rADC_VC[n] register defines which physical channel to convert
  - The conversion will be executed on ADC1 if bADC1_Enable is set to 1 and ADC2 if bADC2_Enable bit is set to 1.
  - At the end of command, the resulting value is stored in the rADC[m]_DATA[n] (rADC1_DATA[n] and rADC2_DATA[n]) registers.
- When an ADC_VC[n] is configured for simultaneous sampling mode (bADC_Mode is set to 2’b01)
  - bADC[m]_ChannelSel (bADC1_ChannelSel and bADC2_ChannelSel) field of the rADC_VC[n] register defines which physical channel to process sample and hold operation. In this case, sample and hold operation are only available on physical channel 6..8 of each ADC
  - The sample and hold will be executed on ADC1 if bADC1_Enable is set to 1 and ADC2 if bADC2_Enable bit is set to 1.

See **Figure 7.4, Virtual Channel ADC_VC Architecture**.

7.5.5 ADC Operation Priority

When multiple ADC_VC[n] with n = 0..15 flags are set at the same time, one of two forms of priority determines the order in which they are converted. The default priority method is round robin. In this scheme, no ADC_VC[n] has an inherent higher priority than another. Priority depends on the round robin pointer (bADC_RR_Pointer).

The bADC_RR_Pointer reflected in the rADC_PRIORITY register points to the last ADC_VC[n] converted. The highest priority ADC_VC[n] is given to the next value greater than the bADC_RR_Pointer value, wrapping around back to ADC_VC0 after ADC_VC15.

At reset the value is 16, since 0 indicates a conversion has already occurred. When bADC_RR_Pointer equals 16 the highest priority is given to ADC_VC0.

The bADC_RR_Pointer is reset when we have one of following conditions:

- A device reset
- The rADC_PRIORITY register is written
- A CPU write to rADC_CONFIG register is starting the following operations:
  - bADC_POWER_DOWN (Enable or Disable PowerDown Mode)
  - bADC_SAMPLE_HOLD_ENABLE (Enable or disable Sample and Hold feature on each channel)
  - If a conversion is currently in progress, it will be completed and then the new priority is become

An example of the round robin priority method is given in figure below:

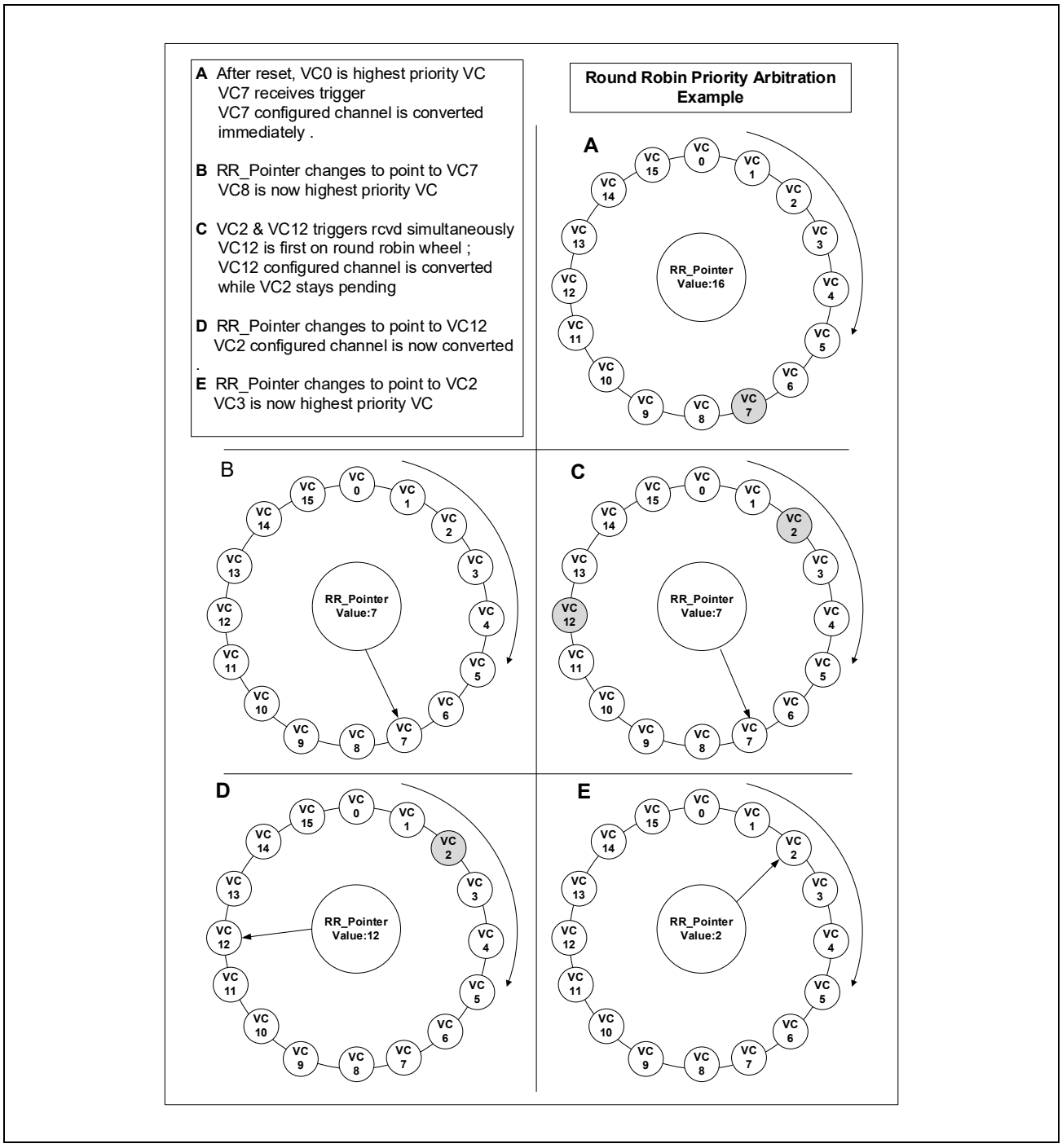

Figure 7.6 ADC Round Robin Priority Arbitration Example

The bADC_Priority field in the rADC_PRIORITY register can be used to assign high priority from a single to all of the ADC_VC's. When configured as high priority, an ADC_VC[n] will interrupt the round robin wheel after any current conversion completes and insert itself in as the next conversion. After its conversion completes, the round robin wheel will continue where it was interrupted. If two high priority ADC_VC's are triggered at the same time, the ADC_VC's with the lower number will take precedence.

High priority mode is assigned first to ADC_VC0, then in increasing numerical order. The value written in the bADC_Priority field defines the first ADC_VC's that is not high priority. In other words, if a value of 4 is written into bADC_Priority, then ADC_VC0..3 are defined as high priority, with ADC_VC0 the highest.

An example using high priority ADC_VC[n]'s is given in figure below:

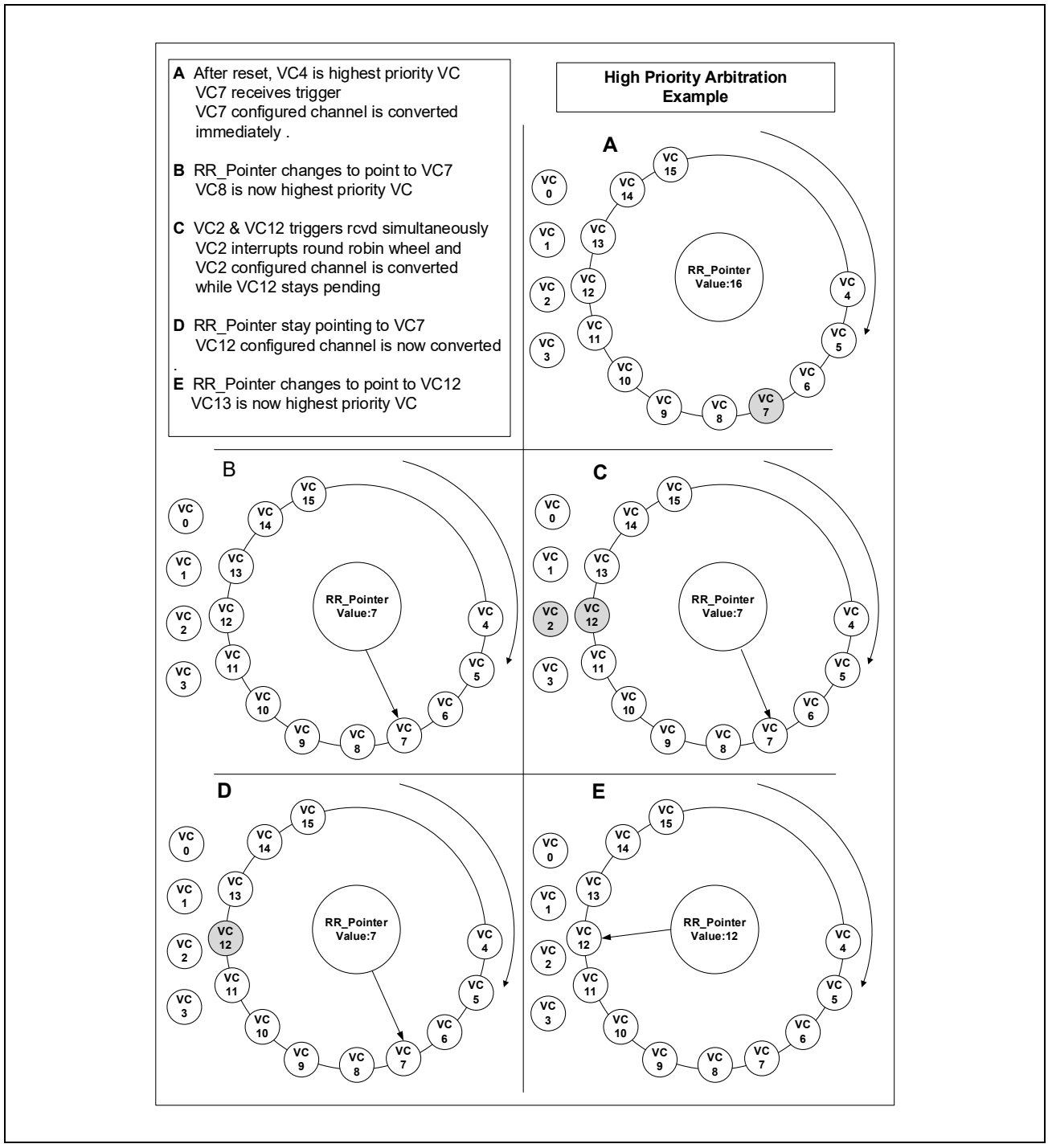

Figure 7.7 ADC High Priority Arbitration Example

7.5.6 Simultaneous Sample and Hold

In some applications, it is important to keep the delay between the sampling of two or more signals minimal. Each ADC contains triple sample and hold circuits to allow 3 different channels to be sampled simultaneously.

- When an ADC_VC[n] is configured for simultaneous sampling mode (bADC_Mode is set to 2'b01), bADC[m]_ChannelSel field of the rADC_VC[n] register defines which channel to process sample and hold operation.
- These features are only available on physical channel 6..8 of each ADC

See **Figure 7.4, Virtual Channel ADC_VC Architecture** and **Figure 7.3, ADC Controller Synoptic**.

In example below, we would like to have a conversion of physical ADC1_IN[8:6] and ADC2_IN[8:6] with simultaneous sample & hold started by use iADC_EOC_VC7, we will use for this ADC_VC0 for sample & hold event and ADC_VC1..3 for convert ADC1_IN [8:6] and ADC2_IN[8:6]. The sample window for the ADC2 is 12 cycles.

For this acquisition, the priority is high.

The resulting values written into the registers will be:

rADC_VC0:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b1 (ADC2 Enable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h17 (Select trigger iADC_EOC_VC7)
- bADC_Mode: 2'b01 (Sample and hold)
- bADC1_ChannelSel: 3'h7 (ADC1_IN[8:6] channel selected)
- bADC2_ChannelSel: 3'h7 (ADC2_IN[8:6] channel selected)

rADC_VC1:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b1 (ADC2 Enable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h17 (Select trigger iADC_EOC_VC7)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h5 (ADC1_IN6 channel selected)
- bADC2_ChannelSel: 3'h5 (ADC2_IN6 channel selected)

rADC_VC2:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b1 (ADC2 Enable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h17 (Select trigger iADC_EOC_VC7)
- bADC_Mode: 2'b00 (Physical channel to convert)

- bADC1_ChannelSel: 3'h6 (ADC1_IN7 channel selected)
- bADC2_ChannelSel: 3'h6 (ADC2_IN7 channel selected)

rADC_VC3:

- bADC1_Enable: 1'b1 (ADC1 Enable)
- bADC2_Enable: 1'b1 (ADC2 Enable)
- bADC_Continuous: 1'b0 (Single conversion)
- bADC_TrigEnable: 1'b1 (Trigger Enable)
- bADC_TrigSel: 5'h17 (Select trigger iADC_EOC_VC7)
- bADC_Mode: 2'b00 (Physical channel to convert)
- bADC1_ChannelSel: 3'h7 (ADC1_IN8 channel selected)
- bADC2_ChannelSel: 3'h7 (ADC2_IN8 channel selected)

rADC_ACQS:

- bADC_TSHSAMP: 5'h0C

rADC_PRIORITY:

- bADC_Priority: 5'h4
  ADC_VC0..3 are high priority,
  ADC_VC4..15 are in round robin mode.
  See **Figure 7.7, ADC High Priority Arbitration Example**.

A sample and hold will be started on a use iADC_EOC_VC7 event and after that, three conversions (on previous sample and hold values) of physical channel ADC1_IN[8:6] and ADC2_IN[8:6] will be started with the resulting value stored in the rADC1_DATA1..3 and rADC2_DATA1..3 registers.

In above example, we can control of one ADC instance has no effect on the other one. So, they can be fully used simultaneously. The only common parts are external iADC_EOC_VC7.

The coupling behavior is as follows:

- In same, trigger iADC_EOC_VC7 trigger will start a sample & hold process on ADC_VC0 and conversion on ADC_VC1..3
- After arbitration, higher priority, arbiter will give a hand now to ADC_VC0 to run simultaneous sample and hold operation on physical channel ADC1_IN[8:6] and ADC2_IN[8:6]
- After arbitration, higher priority, arbiter will give a hand now to ADC_VC1 on physical channel6 to convert data of each ADC
- After arbitration, higher priority, arbiter will give a hand now to ADC_VC2 on physical channel7 to convert data of each ADC
- After arbitration, higher priority, arbiter will give a hand now to ADC_VC3 on physical channel8 to convert data of each ADC
- After each conversion, the data result is stored in ADC[m]_DATA1..3 registers with m:1..2
- ADC_VC3 can be configured to generate interrupts at the end of all operations (sample & hold and convert).

7.5.7 End of Command (EOC) and Interrupt Operation

We have 16 independent ADC_VC[n] configuration sets, managed by 16 iADC_EOC_VC[n] pulse (End of Operation of ADC_VC[n]):

Rising edge of iADC_EOC_VC[n] set an interrupt bADC_INTSTATUS0_VC[n] and bADC_INTSTATUS1_VC [n] bits in rADC_INTSTATUS0 and rADC_INTSTATUS1 registers with:

- bADC_INTSTATUS0_VC[n] bit:
  - Interrupt status before masking on Virtual channel ADC_VC[n] with n = 0..15
- bADC_INTSTATUS1_VC [n] bit:
  - Interrupt status after masking on Virtual channel ADC_VC[n] with n = 0..15
  - When one bit is high in this register, an interrupt is on ADC_Int
- bADC_INTMASK_VC[n] bit:
  - Mask interrupt bit rADC_INTSTATUS0 register.
- bADC_INTCLR_VC[n] bit:
  - Clear respective interrupt bit in rADC_INTSTATUS0 and rADC_INTSTATUS1 registers.

Additionally, if the respective bADC_INTSTATUS0_VC[n] bit is set and a selected additional iADC_EOC_VC[n] trigger is generated, then an overflow condition occurs in bADC_INTOVFSTATUS0_VC[n] and bADC_INTOVFSTATUS1_VC[n] bits in rADC_INTOVFSTATUS0 and rADC_INTOVFSTATUS1 registers with:

- bADC_INTOVFSTATUS0_VC[n] bit:
  - Interrupt overflow status before masking on Virtual channel ADC_VC[n] with n = 0..15
- bADC_INTOVFSTATUS1_VC[n] bit:
  - Interrupt overflow status after masking on Virtual channel ADC_VC[n] with n = 0..15
  - When one bit is high in this register, an interrupt is on ADC_Int
- bADC_INTOVFMASK_VC[n] bit:
  - Mask respective interrupt bit rADC_INTOVFSTATUS0 and rADC_INTOVFSTATUS1 registers.
- bADC_INTCLROVF_VC[n] bit:
  - Clear respective interrupt bit in rADC_INTOVFSTATUS0 and rADC_INTOVFSTATUS1 registers.

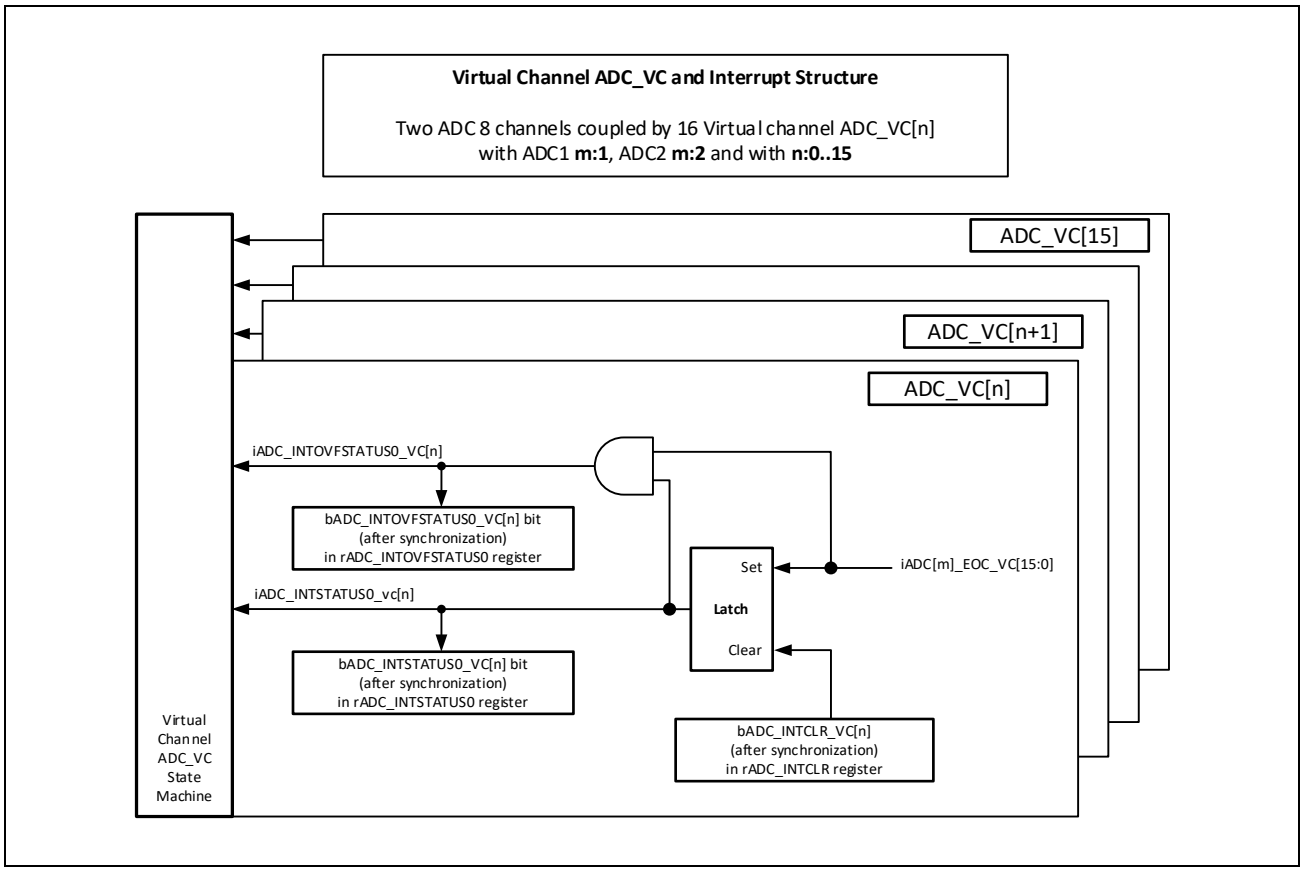

Figure 7.8 ADC Interrupt Structure

7.5.8 Data Copy in Data Lock Register

Each ADC_VC[n] with ADC1 m:1, ADC2 m:2 and n = 0..15 can be configured to data copy from rADC[m]_DATA0..15 to rADC[m]_DATALOCK0..15 when an ADC_VC[n] is configured in Data copy into data lock registers mode (bADC_Mode is set to 2'b11).

An event pending in rADC_PENDING register (Data copy into data lock registers operation) must be executed on ADC1 and or ADC2 when:

- After scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run.
  - If bADC1_Enable bit is set to 1, the respective operation controlled by bADC_Mode (set to 2'b11) and bADC1_ChannelSel fields will be executed on ADC1.
  - If bADC2_Enable bit is set to 1. the respective operation controlled by bADC_Mode (set to 2'b11) and bADC2_ChannelSel fields will be executed on ADC2
- The bADC1_ChannelSel and bADC2_ChannelSel fields select the mask data locked (rADC_MASKLOCK0..3 registers) used to enable or disable copy from rADC[m]_DATA0..15 to rADC[m]_DATALOCK0..15 (with m = 1..2).
- An ADC_VC[n] end of command iADC_EOC_VC[n] is generated for interrupt management or trigger input for another ADC_VC[n]
- A potential ADC DMA requests are started and run until detection end of DMA transfer. The ADC DMA channel selected depends on bADC_DMA_Request[1:0] bit status.

See figure below for more details:

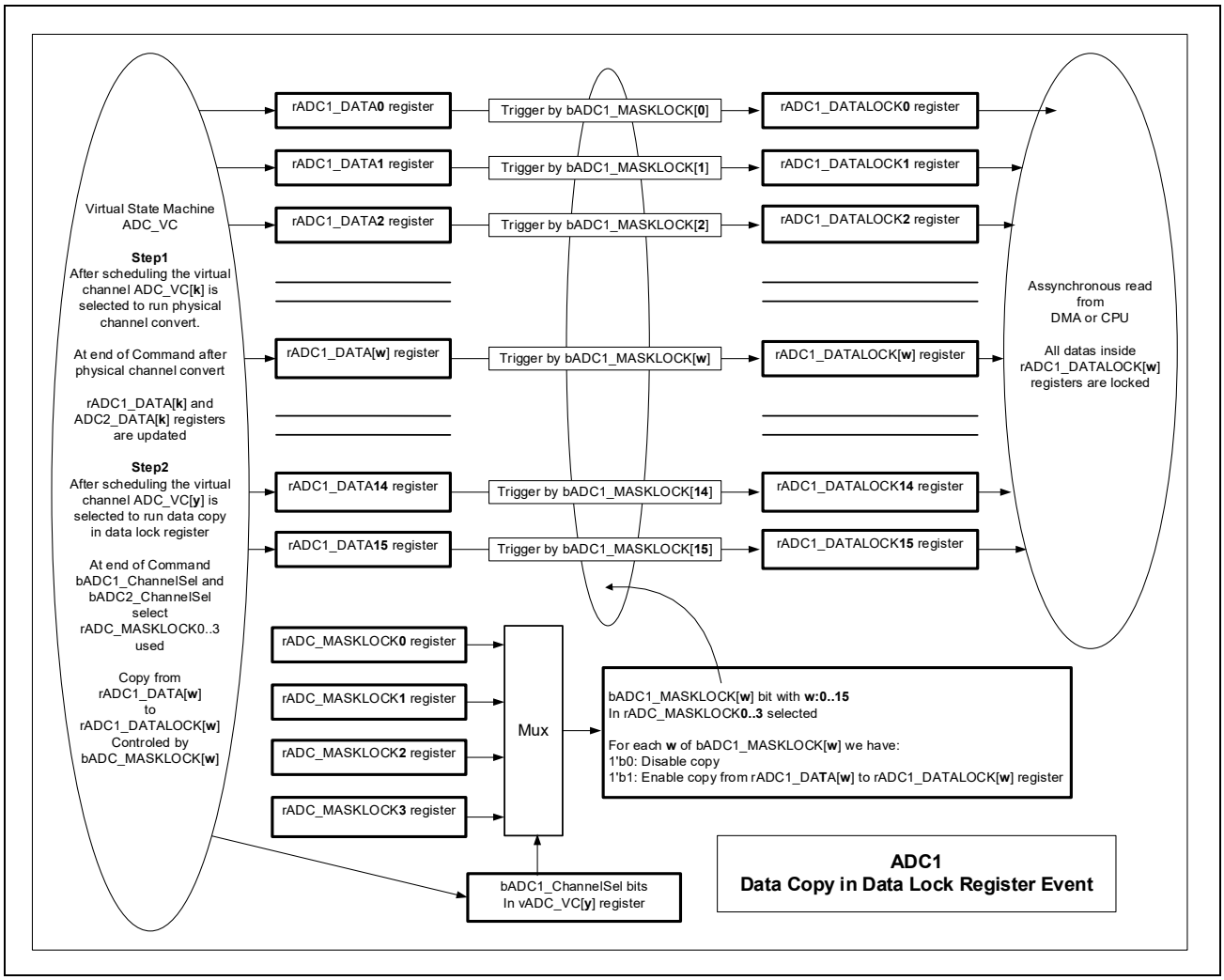

Figure 7.9 ADC Data Copy in Data Lock Register

7.5.9 Timing

7.5.9.1 Basic A/D Conversion on 3 Channels

A/D conversion without sample & hold feature is performed in 3 steps:

- A/D conversion on physical channel0, Controlled by Virtual channel ADC_VC1
- A/D conversion on physical channel1, Controlled by Virtual channel ADC_VC2
- A/D conversion on physical channel2, Controlled by Virtual channel ADC_VC3
- A/D conversion is performed channel by channel (depending on priority configuration in rADC_PRIORITY)
- Set an interrupt when End of Command detected
- Acknowledge by CPU
- Each A/D conversion is started by setting iADC[m]_CONV (with m = 1..2 depending ADCs) at the rising edge of clock.

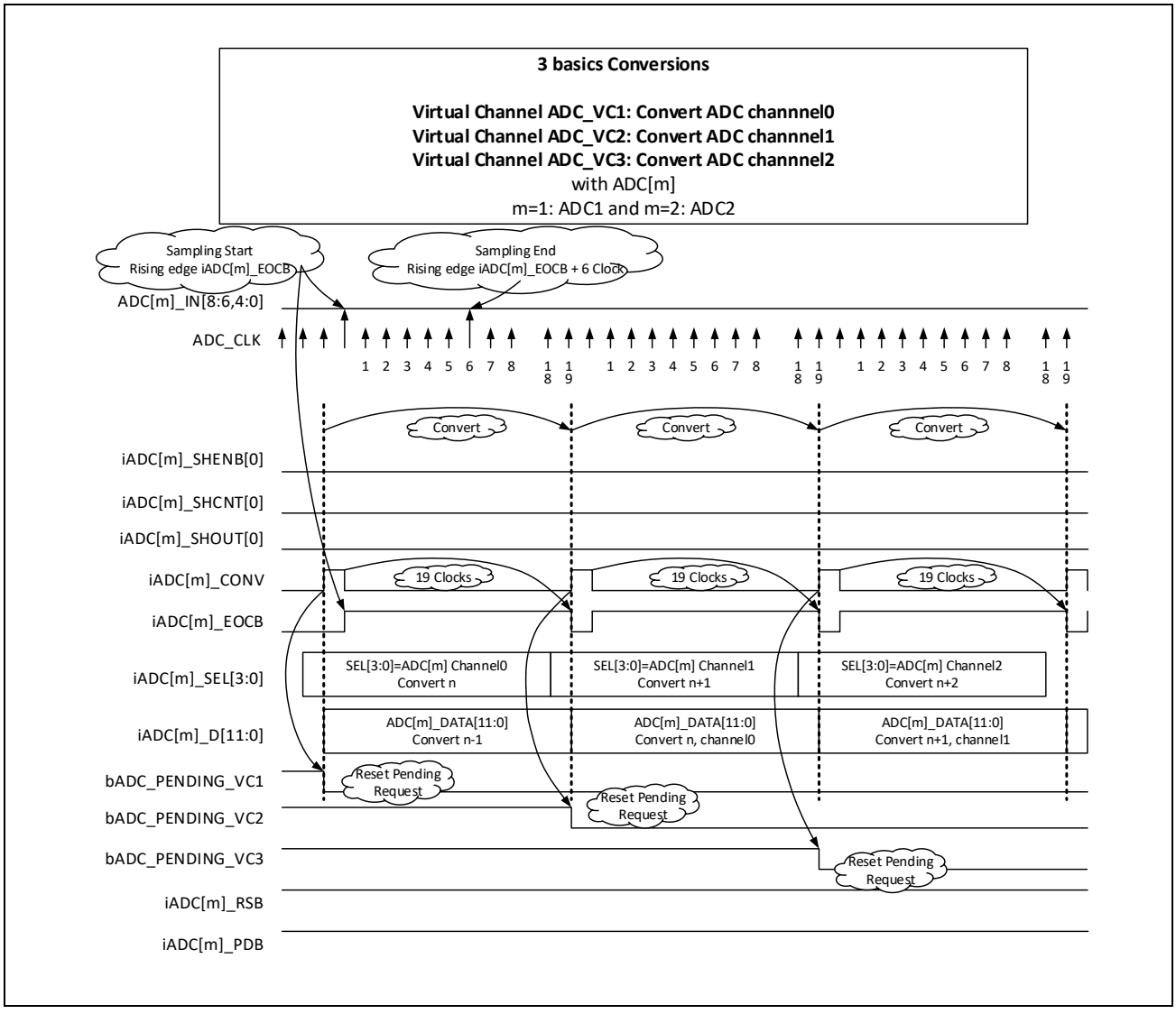

Figure 7.10 No Sample & Hold, Basics A/D Conversion on 3 Channels

7.5.9.2 Sample & Hold following by A/D Conversion on One Channel

A/D conversion with sample & hold feature is performed in 2 steps:

- Sample & Hold operation (without A/D conversion) on physical channel6
  - Controlled by Virtual channel ADC_VC0
- A/D conversion on physical channel6
  - Controlled by Virtual channel ADC_VC1
- A/D conversion is performed channel by channel (depending on priority configuration in rADC_PRIORITY)
- Set an interrupt when End of Command detected
- Acknowledge by CPU
- A/D conversion is started by setting iADC[m]_CONV (with m = 1..2) at the rising edge of clock.
- When you change ADCs from power down to operation mode, CPU should wait the recovery time (1 us) for it to be stable (read bADC_BUSY bit as 0 in rADC_CONTROL register).
- You can only do one A/D conversion by one sampling on sample/hold circuit. If you would like to do another A/D conversion, you should do sampling again.
- The hold time of sample & hold mode is 15 ms maximum and must be managed by firmware.

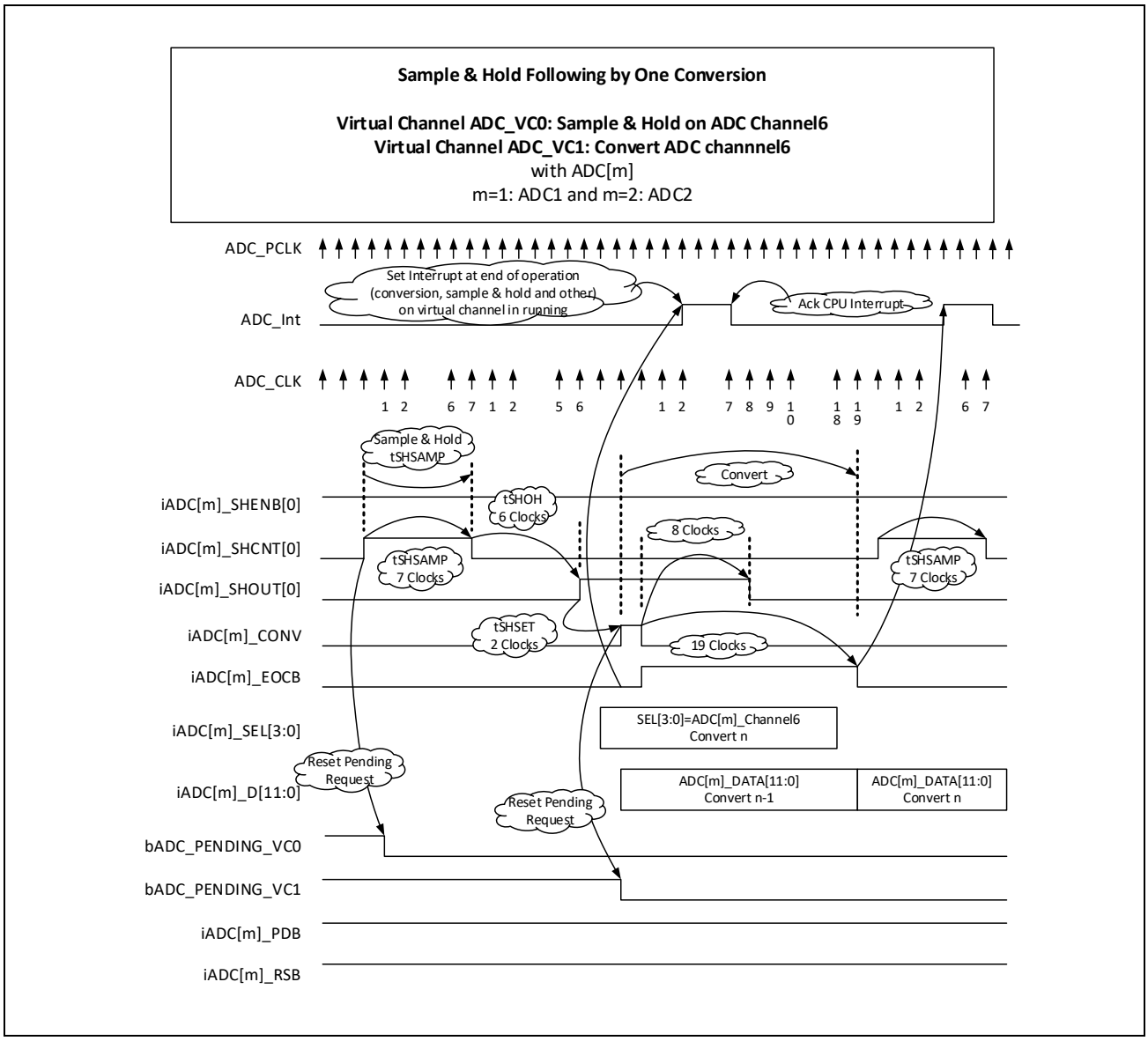

Figure 7.11 Sample & Hold following by A/D Conversion on One Channel

7.5.9.3 Sample & Hold following by A/D Conversion on 3 Channels

A/D conversion with sample & hold feature is performed in 4 steps:

- Sample & Hold operation (without A/D conversion) on physical channel 6..8
  - Controlled by Virtual channel ADC_VC0
- A/D conversion on physical channel 6 – Controlled by Virtual channel ADC_VC1
- A/D conversion on physical channel 7 – Controlled by Virtual channel ADC_VC2
- A/D conversion on physical channel 8 – Controlled by Virtual channel ADC_VC3
- A/D conversion is performed channel by channel (depending on priority configuration in rADC_PRIORITY)
- Set an interrupt when End of Command detected
- Acknowledge by CPU
- A/D conversion is started by setting iADC[m]_CONV (with m = 1..2) at the rising edge of clock.
- When you change ADCs from power down to operation mode, CPU should wait the recovery time (1 µs) for it to be stable (read bADC_BUSY bit as 0 in rADC_CONTROL register).
- You can only do three A/D conversions by one sampling on sample/hold circuit. If you would like to do another A/D conversion, you should do sampling again.
- The hold time of sample & hold mode is 15 ms maximum and must be managed by firmware.

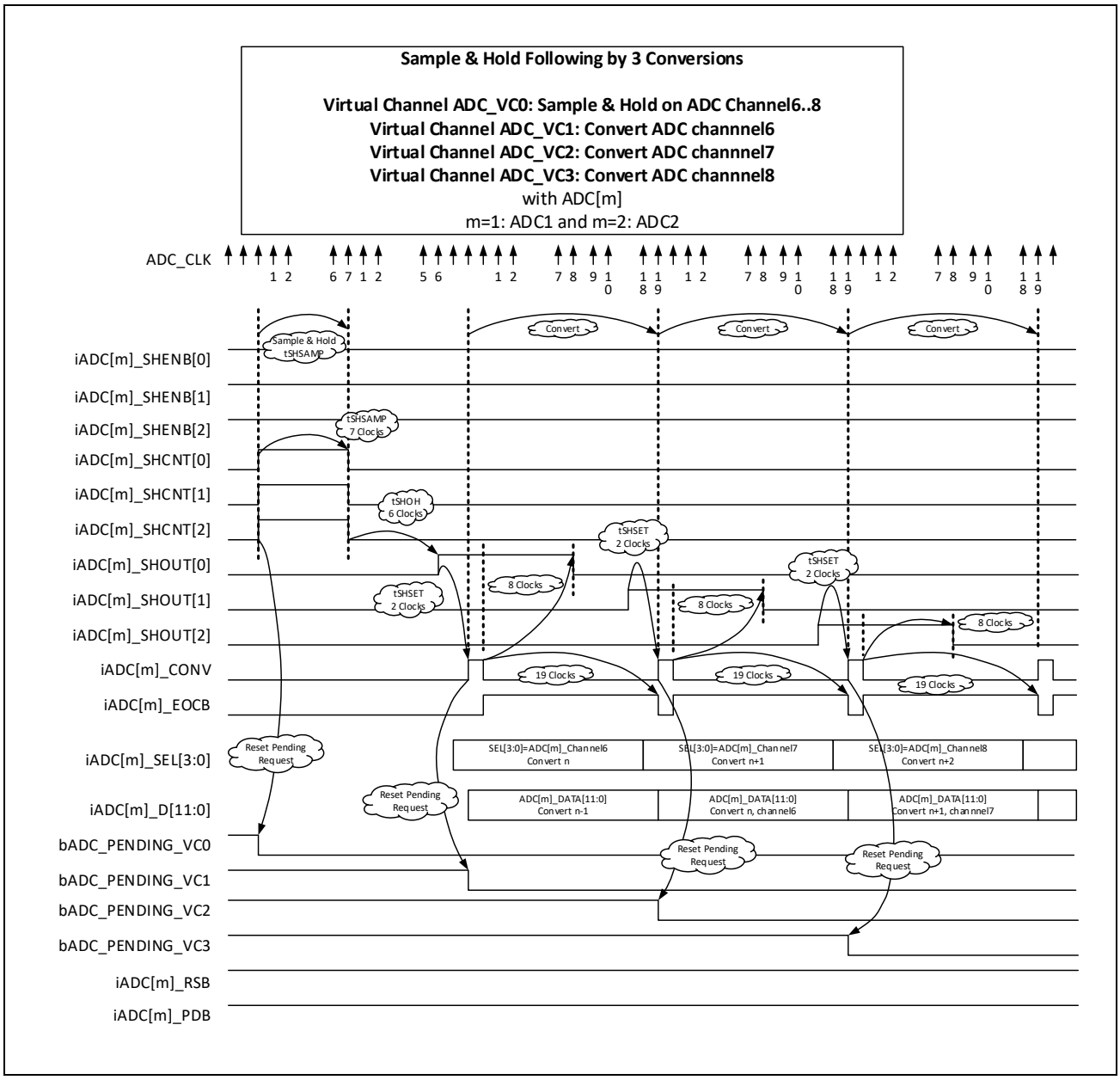

Figure 7.12 Sample & Hold following by A/D Conversion on 3 Channels

7.5.9.4 Power Down

Power Down Mode Sequence is performed in following steps:

- Current A/D conversion on physical channel 0
  - Controlled by Virtual channel ADC_VC1
- CPU Write sets 1 in bADC_POWER_DOWN bit of rADC_CONFIG register
  - Clear iADC[m]_PDB and iADC[m]_RSB signals to 0 (with m = 1..2)
  - Set 1 in bADC_BUSY bit of rADC_CONTROL register
- Clear bADC_TrigEnable bit in all rADC_VC[n] registers to 0 (Trigger disable), with n = 0..15
  - Clear bADC_PENDING_VC[n] bit (operation pending on Virtual channel ADC_VC[n]) in rADC_PENDING register to 0, with n = 0..15
  - Clear bADC_FORCE_VC[n] bit (Force start of operation on Virtual channel ADC_VC[n]) in rADC_FORCE register to 0, with n = 0..15
  - Stop current conversion in running and force virtual state machine in idle state at the end of current operation
- CPU polls bADC_BUSY bit in rADC_CONTROL register until read as 0 (Configuration change finished)

- CPU Write sets 0 in bADC_POWER_DOWN bit of rADC_CONFIG register
  - Set 1 in iADC[m]_PDB and iADC[m]_RSB signals (with m = 1..2)
  - Set 1 in bADC_BUSY bit of rADC_CONTROL register until read as 0 (Recovery time ≥ 1 µs)
  - After recovery time, ADC1 and ADC2 are configured in operate mode.
- CPU, now, must reconfigure all virtual channel ADC_VC[n] with n = 0..15
- A/D conversion on physical channel 4
  - Controlled by Virtual channel ADC_VC2
- A/D conversion is performed channel by channel (depending on priority configuration in rADC_PRIORITY)
- Set an interrupt when End of Command detected
- Acknowledge by CPU
- A/D conversion is started by setting iADC[m]_CONV (with m = 1..2) at the rising edge of clock.

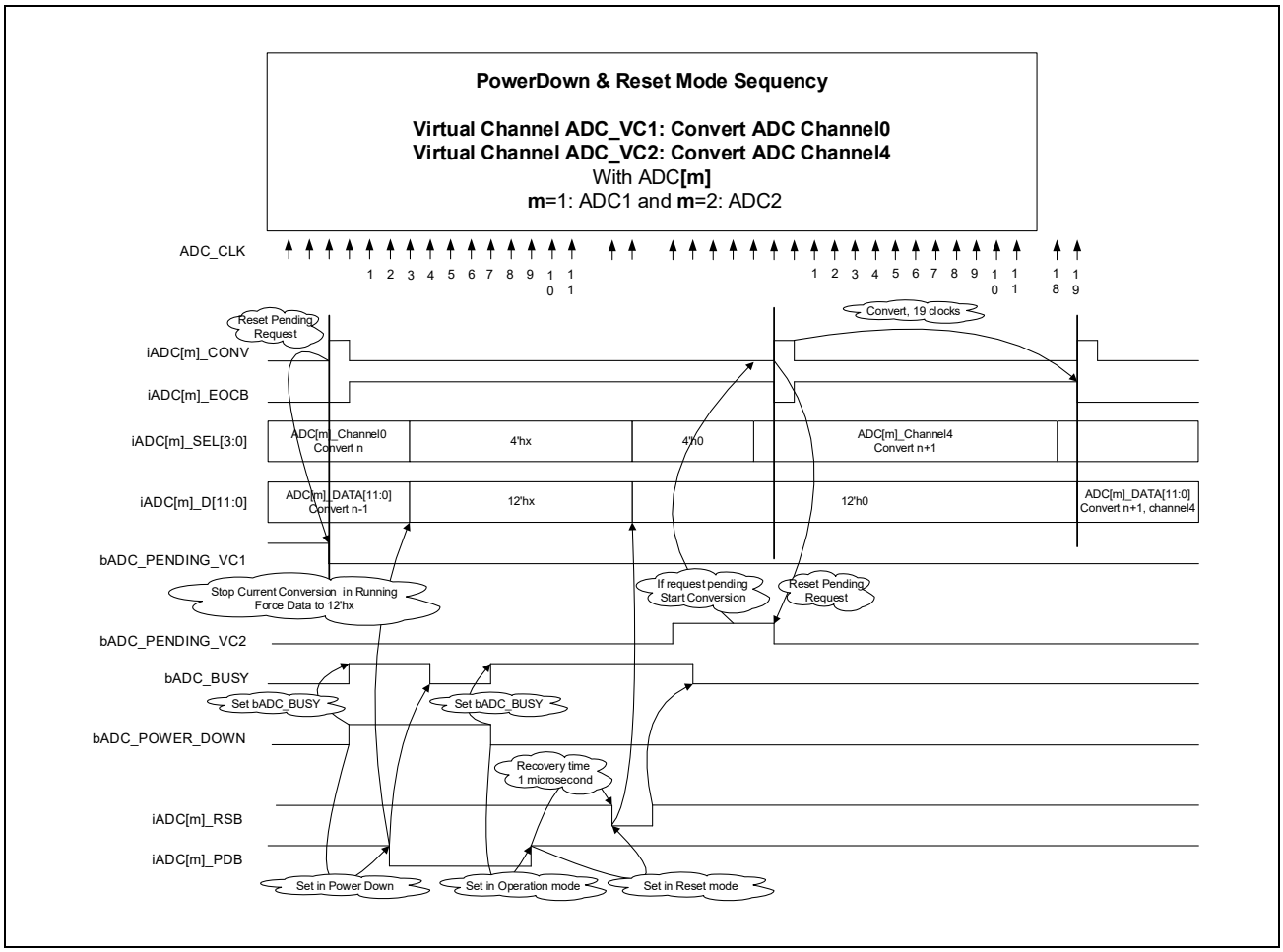

Figure 7.13 ADC PowerDown

7.5.9.5 A/D Conversion Rate

Based on previous figures, A/D conversion rates can be calculated.

Table 7.27 A/D Conversion Rate

Mode	Number of ADC Clock Periods	Number of ADC Clock Periods per Conversion (Average)
Basic A/D Conversion	20	20
3 channels simultaneous sample&hold	(7 + 6 + 2) + 3 × (20)	25
3 channels separated sample&hold	3 × ((7 + 6 + 2) + 20))	35

7.5.10 DMA control

The ADC controller has DMA capability. 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 the DMA. In this mode, DMA controller must be configured in DMAC flow controller mode. The DMA always transfers data using DMA burst transactions if possible, for efficiency.

The ADC controller uses two DMA channel to transfer ADC conversion values (registers rADC[m]_DATALOCK[n] with n = 0..15 and m = 1..2).

The DMA ADC flow control is managed by these followings DMA bits:

- bADC_DMA bit in rADC_CONFIG register
  - Enable or disable DMA channel 0 & 1
- bADC_DMA_Request bits in rADC_VC[n] registers with n = 0..15
  - When after scheduling by ADC_VC state machine, the ADC_VC[n] is selected to run and following events detected: Event: “Data copy into data lock registers”, an ADC DMA requests are started and run until detection end of DMA transfer. The ADC DMA channel selected depends on bADC_DMA_Request[1:0] bit status.
- bADC_DMA0_RUNNING and bADC_DMA1_RUNNING bits in rADC_PENDING register
  - ADC DMA requests are started and run until detection end of DMA transfer.
- rADC1_DATALOCK[n] and rADC2_DATALOCK[n] registers with n = 0..15
  - Copy locked of bADC1_DATA bits of rADC1_DATA[n] register, with n = 0..15 for DMA or CPU read
  - Copy locked of bADC2_DATA bits of rADC2_DATA[n] register, with n = 0..15 for DMA or CPU read
  - See **Figure 7.9, ADC Data Copy in Data Lock Register**.

To enable the DMA Controller interface on the ADC and enable the handshaking interface:

- DMA controller must be configured
  - Address of source and destination
  - Size of burst on source and destination
  - Size of block to transfer
  - DMAC flow controller mode
  - Channel allocation
  - Only one block.
  - Interrupt
- Set bADC_DMA bit in rADC_CONFIG register to enable ADC DMA channel
  - Set bADC_DMA_Request bits in rADC_VC[n] registers with n = 0..15 to start DMA transfer of channel selected when we have an event “Data copy into data lock registers”, then an ADC DMA request is started and run until detection end of DMA transfer.
- bADC_DMA0_RUNNING and bADC_DMA1_RUNNING bits in rADC_PENDING register give a current status on DMA transactions in running.
- Use only rADC1_DATALOCK[n] and rADC2_DATALOCK[n] registers with n = 0..15 to DMA transfer
  - Copy locked of rADC1_DATA[n] and rADC2_DATA[n] registers
  - See **Figure 7.9, ADC Data Copy in Data Lock Register**.

7.5.10.1 Overview on DMA Operation

CAUTION

- DMA controller must be configured in DMAC flow controller mode, because ADC does not know the size of block transferred.
- ADC supports only 32 bits width.

Recommended value to manage correctly all transactions in burst and single mode:

- Width of APB: 32 bits
- Burst line size: From 4 × 32 bits to 16 × 32 bits

For example, if the block size programmed into the DMA Controller is 12 and the burst transaction length is set to 4, 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 ADC makes a receive request to this channel, four data items are read from rADC1_DATALOCK[n] and rADC2_DATALOCK[n] registers. Three separate requests must be made to this DMA channel before all 12 data items are read.

CAUTION

The source transfer width settings in the DMAC, DMAC.CTL[n].SRC_TR_WIDTH must be set to 3'b010 because the ADC registers are 32 bits width.

7.6 Usage Notes

7.6.1 Restriction

If two channels of DMA are used simultaneously, DMA requests may not work properly.

At this time, bADC_DMA1_RUNNING/bADC_DMA0_RUNNING bit of rADC_PENDING register and bADC_DMA1_RUNNINGOVF/bADC_DMA0_RUNNINGOVF bit of rADC_PENDINGOVF register may not indicate proper values. Only single channel of DMA can be available.

Section 8 LCD Controller

8.1 Overview

The PG4 (Peripheral Group 4) Subsystem of RZ/N1 provides LCD Controller.

- Wide range of programmable LCD Panel resolutions
- Interface for 1 Port TFT LCD Panel:
  - 18-bit digital (6 bits/color)
  - 24-bit digital (8 bits/color)
- Programmable frame buffer bits per pixel (bpp)
  - 1, 2, 4, 8 bpp mapped through Color Palette to 18-bit LCD pixel
  - 16, 18 bpp directly drive 18-bit LCD pixel
  - 24 bpp directly drive 24-bit LCD pixel
- Color Palette RAM: 256 words x 16 bits
- Programmable output format
  - RGB 6:6:6 or 5:6:5 or 5:5:5 on 18-bit digital interface
  - RGB 5:6:5 on 24-bit digital interface (via 8 bpp through palette)
  - RGB 8:8:8 on 24-bit digital interface
- Hardware blink supported
- Two Pulse Width Modulation module for LCD panel LED backlight brightness control
  - LCD_PWM[1:0]
- Power up and down sequencing supported
- Integrated DMA
- Supported LCD Panel Characteristics:

Panel Interface	Bits per Pixel	Palette Size	Number of Color
18/24 bits (6 bits/color)	1	2 entries by 16 bits	2
18/24 bits (6 bits/color)	2	4 entries by 16 bits	4
18/24 bits (6 bits/color)	4	16 entries by 16 bits	16
18/24 bits (6 bits/color)	8	256 entries by 16 bits	256
18/24 bits (6 bits/color)	16	—	32768/65536
18/24 bits (6 bits/color)	18	—	262144
24 bits (8 bits/color)	8	256 entries by 16 bits	256
24 bits (8 bits/color)	24	—	16777216

- Two FIFO memories:
  - Output FIFO: 16 words 24 bits
  - Input FIFO: 1K words 64 bits

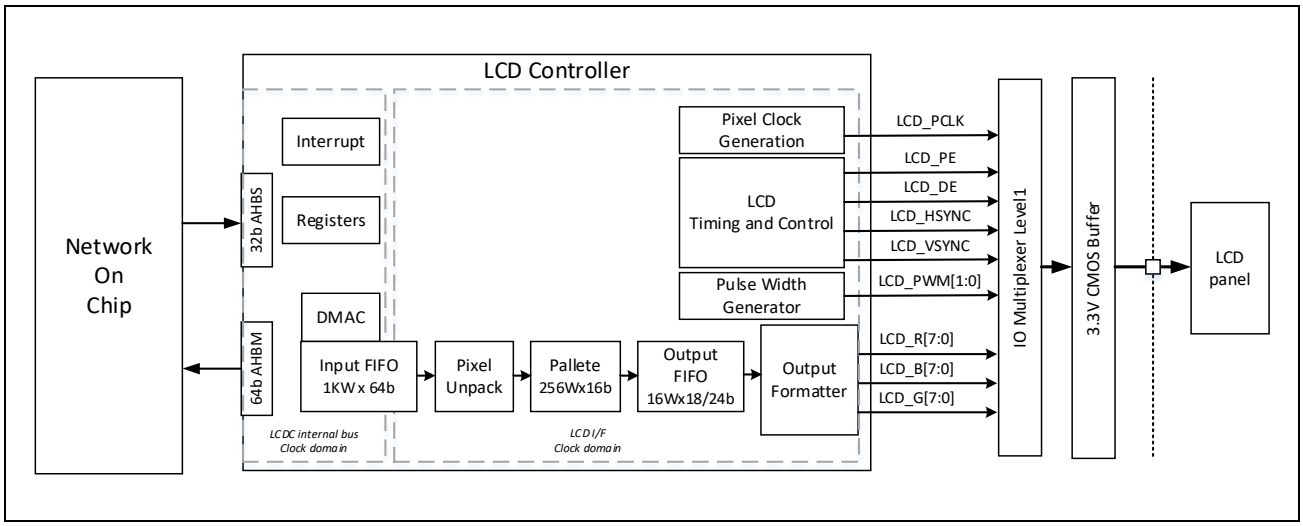

Figure 8.1 LCD Controller Synoptic

8.2 Signal Interfaces

Table 8.1 LCD Signal Interface

Signal Name	Input Output	Description
Clock		
LCD_HCLK	Input	Internal bus clock (AHB)
LCD_ECLK	Input	Reference clock (Pixel Clock domain)
Interrupt		
LCDC_Int	Output	Level sensitive interrupt output, Active High
External Signal		
LCD_PCLK	Output	Pixel Clock
LCD_HSYNC	Output	Horizontal Sync Pulse
LCD_VSYNC	Output	Vertical Sync Pulse
LCD_DE	Output	Data Enable
LCD_PE	Output	Power Enable
LCD_PWM[1:0]	Output	LCD LED Pulse Width Modulation
LCD_R[7:0]	Output	Red Data #Programmable Red/Blue swap mode
LCD_G[7:0]	Output	Green Data
LCD_B[7:0]	Output	Blue Data #Programmable Red/Blue swap mode

8.3 Register Map

Table 8.2 Register Map

Address	Register Symbol	Register Name
5300 4000h	rLcd_CR1	Control Register 1
5300 4008h	rLcd_HTR	Horizontal Timing Register
5300 400Ch	rLcd_VTR1	Vertical1 Timing Register
5300 4010h	rLcd_VTR2	Vertical2 Timing Register
5300 4014h	rLcd_PCTR	Pixel Clock Timing Register
5300 4018h	rLcd_ISR	Interrupt Status Register Before Masking
5300 401Ch	rLcd_IMR	Interrupt Mask Register
5300 4020h	rLcd_IVR	Interrupt Status Register After Masking
5300 4024h	rLcd_ISCR	Interrupt Scan Compare Register
5300 4028h	rLcd_DBAR	DMA Start Base Address of Frame Buffer Memory
5300 402Ch	rLcd_DCAR	DMA Current Base Address on going
5300 4030h	rLcd_DEAR	DMA End Address
5300 4034h	rLcd_PWMFR_0	PWM0 Frequency Register
5300 4038h	rLcd_PWMDCR_0	PWM0 Duty Cycle Register
5300 4044h	rLcd_HVTER	Horizontal and Vertical Timing Extension Register
5300 4048h	rLcd_HPPLOR	Horizontal Pixels-Per-Line Override Control
5300 404Ch	rLcd_PWMFR_1	PWM1 Frequency Register
5300 4050h	rLcd_PWMDCR_1	PWM1 Duty Cycle Register
5300 41F8h	rLcd_GPIOR	Blink Control
5300 41FCh	rLcd_CIR	Core Identification Register

8.3.1 Coding Palette (Palette Registers) Map

CAUTION

- The coding palette depends on the selected color mode.
- The symbol mapped to the address of the coding palette is different for each mode.
- The table below lists the different symbols (names) of the coding palette for all color modes.

Address	Register Symbol	Register Name
5300 4200h to 5300 43FCh	rLcd_PAL_RGB_555	Coding Palette when RGB 5:5:5 mode

Address	Register Symbol	Register Name
5300 4200h to 5300 43FCh	rLcd_PAL_RGB_565	Coding Palette when RGB 5:6:5 mode

Address	Register Symbol	Register Name
5300 4200h to 5300 43FCh	rLcd_PAL_BGR_555	Coding Palette when BGR 5:5:5 mode

Address	Register Symbol	Register Name
5300 4200h to 5300 43FCh	rLcd_PAL_BGR_565	Coding Palette when BGR 5:6:5 mode

8.4 Register Description

8.4.1 rLcd_CR1 — Control Register 1

Address: 5300 4000h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	bLcd_FBP	bLcd_LPS	bLcd_FDW	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_PSS	bLcd_OPS			bLcd_VSP	bLcd_HSP	bLcd_PCP	bLcd_DEP	bLcd_EBO	bLcd_EPO	bLcd_RGB	bLcd_BPP			bLcd_LPE	bLcd_LCE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.3 rLcd_CR1 Register Contents (1/3)

Bit Position	Bit Name	Function	R/W
b31 to b20	Reserved	Read as 0.	R
b19	bLcd_FBP	Frame Buffer 24 bpp Packed Word 0: No Packing, one 24 bpp per 32-bit frame buffer word, upper 8-bit is unused 1: 24 bpp Packing, four 24 bpp pixels packed within 3 frame buffer words ***Caution*)** The packed mode (bLcd_FBP = 1) must be used with 24 bpp mode only (bLcd_BPP = 3'b110).	R/W
b18	bLcd_LPS	LCD Port Select 0: One LCD Port Output (Default) 1: Reserved	R/W
b17, b16	bLcd_FDW	FIFO DMA Request Burst Size Allows burst size configuration of DMA read request in word. The size of elementary word is 64 bits. 2'b00: FIFO DMA request for 4-beat burst when FIFO has room for 4 or more words 2'b01: FIFO DMA request for 8-beat burst when FIFO has room for 8 or more words 2'b10: FIFO DMA request for 16-beat burst when FIFO has room for 16 words 2'b11: Reserved 2'b10 is recommended in terms of bandwidth.	R/W
b15	bLcd_PSS	Palette Load Source Select Keep the initial value 0, this controller does not support for Palette Load Source Select. 0: Load Palette from Slave Bus (CPU transfer used via palette registers) 1: (not available)	R/W

Table 8.3 rLcd_CR1 Register Contents (2/3)

Bit Position	Bit Name	Function	R/W
b14 to b12	bLcd_OPS	Output Pixel Select Bit[12]: bLcd_OPS0: 0: 16 bpp Output Format RGB or BGR 5:6:5 1: 16 bpp Output Format RGB or BGR 5:5:5 Bit[13]: bLcd_OPS1: Valid for bLcd_BPP = 1, 2, 4, 8, 16, 18 bits per pixel. 0: RGB 5:6:5 or 5:5:5 placed on LSB of each 3x8bits RGB interface to LCD panel LCD_R,G,B[5:0] LCD_R,G,B[5:1] 1: RGB 5:6:5 or 5:5:5 placed on MSB of each 3x8bits RGB interface to LCD panel LCD_R,G,B[7:2] LCD_R,G,B[7:3] Bit[14]: bLcd_OPS2: Reserved	R/W
b11	bLcd_VSP	Vertical Sync Polarity 0: LCD_VSYNC signal is active HIGH 1: LCD_VSYNC signal is active LOW	R/W
b10	bLcd_HSP	Horizontal Sync Polarity 0: LCD_HSYNC signal is active HIGH 1: LCD_HSYNC signal is active LOW	R/W
b9	bLcd_PCP	Pixel Clock Polarity 0: Output data signals driven on LCD_PCLK rising edge 1: Output data signals driven on LCD_PCLK falling edge	R/W
b8	bLcd_DEP	Data Enable Polarity 0: LCD_DE signal is active LOW in active display mode 1: LCD_DE signal is active HIGH in active display mode	R/W
b7	bLcd_EBO	Big or Little Endian Byte Ordering Mode in Palette 0: Little endian 1: Big endian	R/W
b6	bLcd_EPO	Big or Little Endian Pixel Ordering within Byte Selects pixel endian ordering within byte for pixels 1, 2 4 bpp only. 0: Little endian 1: Big endian	R/W
b5	bLcd_RGB	Select RGB or BGR Format Mode in Palette 0: RGB mode. Red and Blue data out of Palette NOT swapped 1: BGR mode. Red and Blue data swapped in Output Formatter	R/W
b4 to b2	bLcd_BPP	LCD Bits Per Pixel 3'b000: 1 bpp, 2 16-bit entries of Palette used 3'b001: 2 bpp, 4 16-bit entries of Palette used 3'b010: 4 bpp, 16 16-bit entries of Palette used 3'b011: 8 bpp, 256 16-bit entries of Palette used 3'b100: 16 bpp, No Palette lookup table used 3'b101: 18 bpp, No Palette lookup table used 3'b110: 24 bpp, No Palette lookup table used 3'b111: Reserved	R/W
b1	bLcd_LPE	LCD Power Enable. Directly drives output LCD_PE. Typically used to enable power to the LCD panel. 0: LCD Power Disabled 1: LCD Power Enabled	R/W

Table 8.3 rLcd_CR1 Register Contents (3/3)

Bit Position	Bit Name	Function	R/W
b0	bLcd_LCE	LCD Controller Enable 0: LCD Controller Disabled 1: LCD Controller Enabled Panel signals released to active levels and first frame fetch and display commences. ***Note*)** • When bLcd_LCE = 0, CPU should initialize: Horizontal and vertical Timing rLcd_HTR, rLcd_VTR1, rLcd_VTR2 registers. Extended Horizontal and vertical Timing rLcd_HPPLOR and rLcd_HVTER registers. Pixel Clock RESET released to inactive (no reset state) in rLcd_PCTR register. DMA controller register (rLcd_DBAR). Palette Register. • While bLcd_LCE = 0, LCD panel signals LCD_PCLK, LCD_HSYNC, LCD_VSYNC, LCD_DE, LCD_R[7:0], LCD_G[7:0] and LCD_B[7:0] are held to logic 0. • When bLcd_LCE goes from 1 to 0, LCD timing unit waits for current frame display to finish before forcing panel signals to logic 0. ***Caution*)** Once bLcd_LCE is set to 1, bLcd_LCE must remain 1 throughout the duration of display operation. bLcd_LCE can only be brought low just prior to power-down. bLcd_LCE CANNOT be set to 1, then followed by cleared to 0 (in order to re-configure the controller), then be followed by set to 1 again.	R/W

8.4.2 rLcd_HTR — Horizontal Timing Register

See **Figure 8.2, LCD Horizontal Timing** and **Section 8.5.3, Timing and Control** about the timing.

Address: 5300 4008h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bLcd_HSW								bLcd_HBP							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_PPL								bLcd_HFP							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.4 rLcd_HTR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b24	bLcd_HSW	Horizontal Sync Width • Width of LCD_HSYNC in LCD_PCLK clock periods • Valid values 0 to 255 • Program with LCD_HSYNC desired sync width minus 1 ***Note*)** See bLcd_HSWE of rLcd_HVTER register (Horizontal Sync Width Extension) for the 2-bit extension to bLcd_HSW.	R/W
b23 to b16	bLcd_HBP	Horizontal Back Porch • Number of LCD_PCLK clock periods to wait from end of horizontal Sync Width to beginning of first pixel in a line for display, or LCD_DE active • Valid values 0 to 255 ***Note*)** See bLcd_HBPE of rLcd_HVTER register (Horizontal Back Porch Extension) for the 2-bit extension to bLcd_HBP.	R/W
b15 to b8	bLcd_PPL	Horizontal Pixels-Per-Line • Number of pixels per line • Actual pixels-per-line = 16 × bLcd_PPL. • Maximum pixels-per-line = 16 × 255 = 4080 • Valid values 1 to 255 When bLcd_HPOE is set to 1 in rLcd_HPPLOR register (Horizontal Pixels-Per-Line Override), the bLcd_HPPLO field overrides bLcd_PLL field. bLcd_HPPLO used for panels with pixels-per-line not divisible by 16.	R/W
b7 to b0	bLcd_HFP	Horizontal Front Porch • Number of LCD_PCLK clock periods to add from end of LCD_DE active period to beginning of LCD_HSYNC • Valid values 0 to 255 ***Note*)** See bLcd_HFPE of rLcd_HVTER register (Horizontal Front Porch Extension) for the 2-bit extension to bLcd_HFP.	R/W

8.4.3 rLcd_VTR1 — Vertical1 Timing Register

See **Figure 8.3, LCD Vertical Timing** and **Section 8.5.3, Timing and Control** about the timing.

Address: 5300 400Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	bLcd_VBP							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_VFP								bLcd_VSW							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.5 rLcd_VTR1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b24	Reserved	Read as 0.	R
b23 to b16	bLcd_VBP	Vertical Back Porch • Number of LCD_HSYNC line periods to wait from end of Vertical Sync Width to beginning of first line with active pixels for display • Valid values 0 to 255 ***Note*)** See bLcd_VBPE of rLcd_HVTER register (Vertical Back Porch Extension) for the 2-bit extension to bLcd_VBP.	R/W
b15 to b8	bLcd_VFP	Vertical Front Porch • Number of LCD_HSYNC line periods at end of frame, before LCD_VSYNC active period • Valid values 0 to 255 ***Note*)** See bLcd_VFPE of rLcd_HVTER register (Vertical Front Porch Extension) for the 2-bit extension to bLcd_VFP.	R/W
b7 to b0	bLcd_VSW	Vertical Sync Width: • Width of LCD_VSYNC pulse in LCD_HSYNC line periods • Valid values 0 to 255 ***Note*)** See bLcd_VSWE of rLcd_HVTER register (Vertical Sync Width Extension) for the 2-bit extension to bLcd_VSW.	R/W

8.4.4 rLcd_VTR2 — Vertical2 Timing Register

See **Figure 8.3, LCD Vertical Timing** and **Section 8.5.3, Timing and Control** about the timing.

Address: 5300 4010h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bLcd_LPP											
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.6 rLcd_VTR2 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b12	Reserved	Read as 0.	R
b11 to b0	bLcd_LPP	Lines-Per-Panel • Number of active lines per frame in a panel • Valid values 1 to 4095 lines	R/W

8.4.5 rLcd_PCTR — Pixel Clock Timing Register

Address: 5300 4014h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	bLcd_PCR	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.7 rLcd_PCTR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b11	Reserved	Read as 0.	R
b10	bLcd_PCR	Pixel Clock Domain Reset 0: Pixel Clock RESET held active (reset state) All logic on the Pixel Clock domain held in reset 1: Pixel Clock RESET released to inactive (no reset state). In this case, after programming configuration register, the LCD Controller can be started. All logic on the Pixel Clock domain held in reset when bLcd_PCR = 0. The reference clock LCD_ECLK of Pixel Clock domain should be programmed in System Controller with frequency required depending on LCD resolution. When clock is configured and stabilized, then the reset for the Pixel Clock domain can be de-asserted via write with bLcd_PCR set to 1.	R/W
b9 to b0	Reserved	Read as 0.	R

8.4.6 rLcd_ISR — Interrupt Status Register Before Masking

Writing 1 to this bit clears it, no action if writing 0.

Address: 5300 4018h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	bLcd_LDD	bLcd_BAU	bLcd_VCT	bLcd_MBE	bLcd_FER	bLcd_IFO	bLcd_IFU	bLcd_OFO	bLcd_OFU
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.8 rLcd_ISR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b9	Reserved	Read as 0.	R
b8	bLcd_LDD	LCD Controller Disable Done Interrupt Status Before Masking (AND logic) When bLcd_LCE in rLcd_CR1 register goes from 1 to 0 (from LCD Controller active to inactive mode), LCD timing unit waits for current frame being displayed to finish before LCD Controller goes into inactive mode and output signals to panel forced to logic 0. Once LCD Controller goes into inactive mode, bLcd_LDD is set to 1 and an interrupt is generated (if not masked). 0: LCD Controller has not been disabled 1: LCD Controller has been disabled generates LCDC_Int (if not masked)	R/W
b7	bLcd_BAU	DMA Base Address field Update To bLcd_DCAR Interrupt Status Before Masking (AND logic) 0: DMA Base Address field (bLcd_DBAR) NOT transferred to Current Address field (bLcd_DCAR) 1: DMA Base Address field (bLcd_DBAR) transferred to Current Address field (bLcd_DCAR) generates LCDC_Int (if not masked)	R/W
b6	bLcd_VCT	Vertical Compare Triggered Interrupt Status Before Masking (AND logic) 0: When Vertical Scan Compare programmed, trigger not reached 1: When Vertical Scan Compare programmed, trigger reached generates LCDC_Int (if not masked) See rLcd_ISCR register definition for Vertical Scan Compare programming triggers.	R/W
b5	bLcd_MBE	DMA Master AHB Bus Error Interrupt Status Before Masking (AND logic) 0: No Error 1: DMA Master AHB Bus encountered an error generates LCDC_Int (if not masked)	R/W
b4	bLcd_FER	Input or Output FIFO Error, Underrun or Overrun Interrupt Status Before Masking (AND logic) 0: No Error 1: Any of bLcd_OFU, bLcd_OFO, bLcd_IFU, bLcd_IFO error (OR logics) bits set to active 1, set also bLcd_FER and generates LCDC_Int (if not masked)	R/W
b3	bLcd_IFO	Input FIFO Overrun Interrupt Status Before Masking (AND logic) 0: No Error 1: Input FIFO Overrun Error generates LCDC_Int (if not masked)	R/W
b2	bLcd_IFU	Input FIFO Underrun Interrupt Status Before Masking (AND logic) 0: No Error 1: Input FIFO Underrun Error generates LCDC_Int (if not masked)	R/W
b1	bLcd_OFO	Output FIFO Overrun Interrupt Status Before Masking. (AND logic) 0: No Error 1: Output FIFO Overrun Error generates LCDC_Int (if not masked)	R/W

Table 8.8 rLcd_ISR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b0	bLcd_OFU	Output FIFO Underrun Interrupt Status Before Masking (AND logic) 0: No Error 1: Output FIFO Underrun Error generates LCDC_Int (if not masked)	R/W

8.4.7 rLcd_IMR — Interrupt Mask Register

Address: 5300 401Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	bLcd_LDDM	bLcd_BAUM	bLcd_VCTM	bLcd_MBEM	bLcd_FERM	bLcd_IFOM	bLcd_IFUM	bLcd_OFOM	bLcd_OFUM
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.9 rLcd_IMR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b9	Reserved	Read as 0.	R
b8	bLcd_LDDM	LCD Controller Disable Done, Mask Enable/Disable 0: Disable LDD interrupt 1: Enable LDD interrupt (Driven by bLcd_LDD)	R/W
b7	bLcd_BAUM	DMA Base Address Register Update To bLcd_DCAR, Mask Enable/Disable 0: Disable BAU interrupt 1: Enable BAU interrupt (Driven by bLcd_BAU)	R/W
b6	bLcd_VCTM	Vertical Compare Triggered, Mask Enable/Disable 0: Disable VCT interrupt 1: Enable VCT interrupt (Driven by bLcd_VCT)	R/W
b5	bLcd_MBEM	DMA Master AHB Bus Error, Mask Enable/Disable 0: Disable MBE interrupt 1: Enable MBE interrupt (Driven by bLcd_MBE)	R/W
b4	bLcd_FERM	Input or Output FIFO Error, Underrun or Overrun, Mask Enable/Disable 0: Disable FER interrupt 1: Enable FER interrupt (Driven by bLcd_FER)	R/W
b3	bLcd_IFOM	Input FIFO Overrun, Mask Enable/Disable 0: Disable IFO interrupt 1: Enable IFO interrupt (Driven by bLcd_IFO)	R/W
b2	bLcd_IFUM	Input FIFO Underrun, Mask Enable/Disable 0: Disable IFU interrupt 1: Enable IFU interrupt (Driven by bLcd_IFU)	R/W
b1	bLcd_OFOM	Output FIFO Overrun, Mask Enable/Disable 0: Disable OFO interrupt 1: Enable OFO interrupt (Driven by bLcd_OFO)	R/W
b0	bLcd_OFUM	Output FIFO Underrun, Mask Enable/Disable 0: Disable OFU interrupt 1: Enable OFU interrupt (Driven by bLcd_OFU).	R/W

8.4.8 rLcd_IVR — Interrupt Status Register After Masking

Address: 5300 4020h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	bLcd_LDDV	bLcd_BAUV	bLcd_VCTV	bLcd_MBEV	bLcd_FERV	bLcd_IFOV	bLcd_IFUV	bLcd_OFOV	bLcd_OFUV
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.10 rLcd_IVR Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b9	Reserved	Read as 0.	R
b8	bLcd_LDDV	LCD Controller Disable Done Interrupt Status After Masking (AND logic) When bLcd_LCE in rLcd_CR1 register goes from 1 to 0 (from LCD Controller active to inactive mode), LCD timing unit waits for current frame being displayed to finish before LCD Controller goes into inactive mode and output signals to panel forced to logic 0. Once LCD Controller goes into inactive mode, if interrupt not masked, bLcd_LDDV is set to 1 and an interrupt is generated. 0: LCD Controller has not been disabled 1: LCD Controller has been disabled generates LCDC_Int	R
b7	bLcd_BAUV	DMA Base Address Register Update To bLcd_DCAR Interrupt Status After Masking (AND logic) 0: DMA Base Address Register (bLcd_DBAR) NOT transferred to Current Address Register (bLcd_DCAR) 1: DMA Base Address Register (bLcd_DBAR) transferred to Current Address Register (bLcd_DCAR) generates LCDC_Int	R
b6	bLcd_VCTV	Vertical Compare Triggered Interrupt Status After Masking (AND logic) 0: When Vertical Scan Compare programmed, trigger not reached 1: When Vertical Scan Compare programmed, trigger reached generates LCDC_Int See rLcd_ISCR register definition for Vertical Scan Compare programming triggers.	R
b5	bLcd_MBEV	DMA Master AHB Bus Error Interrupt Status After Masking (AND logic) 0: No Error 1: DMA Master AHB Bus encountered an error generates LCDC_Int	R
b4	bLcd_FERV	Input or Output FIFO Error, Underrun or Overrun Interrupt Status After Masking. (AND logic) 0: No Error 1: Any of bLcd_OFUV, bLcd_OFOV, bLcd_IFUV, bLcd_IFOV error (OR logics) bits set to active 1 set also bLcd_FERV and generates LCDC_Int	R
b3	bLcd_IFOV	Input FIFO Overrun Interrupt Status After Masking (AND logic) 0: No Error 1: Input FIFO Overrun Error generates LCDC_Int	R
b2	bLcd_IFUV	Input FIFO Underrun Interrupt Status After Masking (AND logic) 0: No Error 1: Input FIFO Underrun Error generates LCDC_Int	R
b1	bLcd_OFOV	Output FIFO Overrun Interrupt Status After Masking (AND logic) 0: No Error 1: Output FIFO Overrun Error generates LCDC_Int	R

Table 8.10 rLcd_IVR Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b0	bLcd_OFUV	Output FIFO Underrun Interrupt Status After Masking (AND logic) 0: No Error 1: Output FIFO Underrun Error generates LCDC_Int	R

8.4.9 rLcd_ISCR — Interrupt Scan Compare Register

Address: 5300 4024h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	bLcd_VSC		
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.11 rLcd_ISCR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b3	Reserved	Read as 0.	R
b2 to b0	bLcd_VSC	Vertical Scan Compare 3’b000: bLcd_VSC inactive 3’b100: Start of Vertical Sync Width (bLcd_VSW) pulse 3’b101: Start of Vertical Back Porch (bLcd_VBP) 3’b110: Start of Active Frame Window 3’b111: Start of Vertical Front Porch (bLcd_VFP) 3’b001, 3’b010, 3’b011: Reserved	R/W

8.4.10 rLcd_DBAR — DMA Start Base Address of Frame Buffer Memory

Address: 5300 4028h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bLcd_DBAR															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_DBAR													—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.12 rLcd_DBAR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b3	bLcd_DBAR	DMA Base Address Start address of frame buffer memory. This value is transferred to bLcd_DCAR at the start of each vertical frame period. bLcd_DBAR should be written when the LCD Controller is disabled (bLcd_LCE set to 0) or immediately after the bLcd_BAU is set to 1, signifying bLcd_DBAR transferred to bLcd_DCAR register.	R/W
b2 to b0	Reserved	Read as 0.	R

8.4.11 rLcd_DCAR — DMA Current Base Address on Going

Address: 5300 402Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bLcd_DCAR															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_DCAR															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.13 rLcd_DCAR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bLcd_DCAR	DMA Current Address Current address DMA controller is accessing data from frame buffer memory. Continuously reflects the next 64 bits frame buffer word to be loaded into the input FIFO. bLcd_DBAR is transferred to bLcd_DCAR at the start of each vertical frame period.	R

8.4.12 rLcd_DEAR — DMA End Address

Address: 5300 4030h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bLcd_DEAR															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_DEAR															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.14 rLcd_DEAR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bLcd_DEAR	DMA End Address End address of frame buffer memory. Compared to register bLcd_DCAR to determine last frame buffer memory address to read from. ***Caution*)** bLcd_DEAR value must be configured with End of Block to transfer + 1 + 8 Example: • Start of Block is 32'h5000_0000 • End of Block is 32'h5000_FFFF • Size Block: 64 KB (32'h1_0000) Please configures: • bLcd_DBAR: 32'h5000_0000 • bLcd_DEAR: 32'h5001_0008 (+1+8)	R/W

8.4.13 rLcd_PWMFR_0 — PWM0 Frequency Register

Address: 5300 4034h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	bLcd_PWMFCE_0	bLcd_PWMFCD_0					
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_PWMFCD_0															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.15 rLcd_PWMFR_0 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b23	Reserved	Read as 0.	R
b22	bLcd_PWMFCE_0	PWM0 Frequency Clock Enable 0: LCD_PWM[0] inactive 1: LCD_PWM[0] active ***Note*)** • The output of this PWM0 unit drives the BLINK logic. • This mode is active when bLcd_BlinkOn is set to “1” and managed by bLcd_BlinkMode inside Lcd_GPIOR register.	R/W
b21 to b0	bLcd_PWMFCD_0	PWM0 Frequency Clock (PWM_CLK) Divider The frequency of the internal PWM0 unit derived from LCD_ECLK clock. 0: LCD_ECLK / (256 × 1) 1: LCD_ECLK / (256 × 2) 2: LCD_ECLK / (256 × 3) … … ***Note*)** • The output of this PWM0 unit drives the BLINK logic. • This mode is active when bLcd_BlinkOn is set to “1” and managed by bLcd_BlinkMode inside Lcd_GPIOR register.	R/W

8.4.14 rLcd_PWMDCR_0 — PWM0 Duty Cycle Register

Address: 5300 4038h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bLcd_PWMDC_0							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.16 rLcd_PWMDCR_0 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bLcd_PWMDC_0	PWM0 Duty Cycle Register The duty cycle of LCD_PWM[0] output. 0: 1/256 1: 2/256 2: 3/256 ... ... ***Note***) • The output of this PWM0 unit drives the BLINK logic. • This mode is active when bLcd_BlinkOn is set to “1” and managed by bLcd_BlinkMode inside Lcd_GPIOR register.	R/W

8.4.15 rLcd_HVTER — Horizontal and Vertical Timing Extension Register

See **Figure 8.3, LCD Vertical Timing**, **Figure 8.2, LCD Horizontal Timing**, and **Section 8.5.3, Timing and Control** about the timing.

Address: 5300 4044h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	bLcd_VSWE		—	—	—	—	—	—	bLcd_HSWE	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bLcd_VBPE		—	—	bLcd_VFPE		—	—	bLcd_HBPE		—	—	bLcd_HFPE	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.17 rLcd_HVTER Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b26	Reserved	Read as 0.	R
b25, b24	bLcd_VSWE	Vertical Sync Width Extension Append as two most significant bits to bLcd_VSW in rLcd_VTR1 register (Vertical Sync Width), forming 10-bit field defining the complete Vertical Sync Width value.	R/W
b23 to b18	Reserved	Read as 0.	R
b17, b16	bLcd_HSWE	Horizontal Sync Width Extension Append as two most significant bits to bLcd_HSW in rLcd_HTR register (Horizontal Sync Width), forming 10-bit field defining the complete Horizontal Sync Width value.	R/W
b15, b14	Reserved	Read as 0.	R
b13, b12	bLcd_VBPE	Vertical Back Porch Extension Append as two most significant bits to bLcd_VBP in rLcd_VTR1 register (Vertical Back Porch), forming 10-bit field defining the complete Vertical Back Porch value.	R/W
b11, b10	Reserved	Read as 0.	R
b9, b8	bLcd_VFPE	Vertical Front Porch Extension Append as two most significant bits to bLcd_VFP in rLcd_VTR1 register (Vertical Front Porch), forming 10-bit field defining the complete Vertical Front Porch value	R/W
b7, b6	Reserved	Read as 0.	R
b5, b4	bLcd_HBPE	Horizontal Back Porch Extension Append as two most significant bits to bLcd_HBP in rLcd_HTR register (Horizontal Back Porch), forming 10-bit field defining the complete Horizontal Back Porch value.	R/W
b3, b2	Reserved	Read as 0.	R
b1, b0	bLcd_HFPE	Horizontal Front Porch Extension Append as two most significant bits to bLcd_HFP in rLcd_HTR register (Horizontal Front Porch), forming 10-bit field defining the complete Horizontal Front Porch value.	R/W

8.4.16 rLcd_HPPLOR — Horizontal Pixels-Per-Line Override Control

See **Figure 8.2, LCD Horizontal Timing** and **Section 8.5.3, Timing and Control** about the timing.

Address: 5300 4048h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bLcd_HPOE	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bLcd_HPPLO											
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.18 rLcd_HPPLOR Register Contents

Bit Position	Bit Name	Function	R/W
b31	bLcd_HPOE	Horizontal Pixels-Per-Line Override Enable 0: Horizontal Pixels-Per-Line Override Disable bLcd_PPL field of rLcd_HTR register used Used for panels with pixels-per-line divisible by 16 1: Horizontal Pixels-Per-Line Override Enable. bLcd_HPPLO field of rLcd_HPPLOR overrides bLcd_PPL field Used for panels with pixels-per-line not divisible by 16	R/W
b30 to b12	Reserved	Read as 0.	R
b11 to b0	bLcd_HPPLO	Horizontal Pixels-Per-Line Override Number of Actual pixels per line. Overrides bLcd_PPL field of rLcd_HTR register when bLcd_HPOE is set to 1. For panels with pixels-per-line not divisible by 16, set bLcd_HPOE = 1 and bLcd_HPPLO to the exact number of pixels-per-line. Thus, for an 800 × 600 panel, use bLcd_HPPLO set to 800. Valid values 1 to 4095 pixels per line.	R/W

8.4.17 rLcd_PWMFR_1 — PWM1 Frequency Register

Address: 5300 404Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	bLcd_PWMFCE_1	bLcd_PWMFCD_1					
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_PWMFCD_1															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.19 rLcd_PWMFR_1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b23	Reserved	Read as 0.	R
b22	bLcd_PWMFCE_1	PWM1 Frequency Clock Enable 0: LCD_PWM[1] inactive 1: LCD_PWM[1] active	R/W
b21 to b0	bLcd_PWMFCD_1	PWM1 Frequency Clock (PWM_CLK) Divider The frequency of the internal PWM1 unit derived from LCD_ECLK clock. 0: LCD_ECLK / (256 × 1) 1: LCD_ECLK / (256 × 2) 2: LCD_ECLK / (256 × 3) … …	R/W

8.4.18 rLcd_PWMDCR_1 — PWM1 Duty Cycle Register

Address: 5300 4050h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	bLcd_PWMDC_1							
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.20 rLcd_PWMDCR_1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b8	Reserved	Read as 0.	R
b7 to b0	bLcd_PWMDC_1	PWM1 Duty Cycle Register The duty cycle of LCD_PWM[1] output. 0: 1/256 1: 2/256 2: 3/256	R/W

8.4.19 rLcd_GPIOR — Blink Control

Address: 5300 41F8h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bLcd_BlinkMode	bLcd_BlinkOn
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.21 rLcd_GPIOR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved	Read as 0.	R
b1	bLcd_BlinkMode	Blink Brightness Mode 1: Half Bright Blink, switching between ON and OFF: [ON] LCD_R[7:0] = {R7,R6,R5,R4,R3,R2,R1,R1}, R1 value is duplicated on LCD_R[1:0] LCD_G[7:0] = {G7,G6,G5,G4,G3,G2,G1,G0} LCD_B[7:0] = {B7,B6,B5,B4,B3,B2,B1,B1}, B1 value is duplicated on LCD_B[1:0] [OFF] LCD_R[7:0] = {0,R7,R6,R5,R4,R3,R2,R1} LCD_G[7:0] = {0,G7,G6,G5,G4,G3,G2,G1} LCD_B[7:0] = {0,B7,B6,B5,B4,B3,B2,B1} 0: Black blink, switching between ON and OFF: [ON] LCD_R[7:0] = {R7,R6,R5,R4,R3,R2,R1,R1}, R1 value is duplicated on LCD_R[1:0] LCD_G[7:0] = {G7,G6,G5,G4,G3,G2,G1,G0} LCD_B[7:0] = {B7,B6,B5,B4,B3,B2,B1,B1}, B1 value is duplicated on LCD_B[1:0] [OFF] LCD_R[7:0] = 8’b00000000 LCD_G[7:0] = 8’b00000000 LCD_B[7:0] = 8’b00000000 The blink mode is active when bLcd_BlinkOn is set to “1”. The blink frequency depends bLcd_PWMFCD_0 and bLcd_PWMFCE_0 inside rLcd_PWMFR_0 register.	R/W
b0	bLcd_BlinkOn	Blink function Enable or Disable 0: Blink Mode disabled 1: Blink Mode enabled	R/W

8.4.20 rLcd_CIR — Core Identification Register

Address: 5300 41FCh

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	bLcd_MN							
Value after reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_BW				bLcd_BI				bLcd_REV							
Value after reset	0	1	0	0	0	0	0	1	0	0	0	0	1	1	1	1

Table 8.22 rLcd_CIR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b24	Reserved	Read as 0.	R
b23 to b16	bLcd_MN	Core Identification Register - Model Number 8’h90: Model Number	R
b15 to b12	bLcd_BW	Core Identification Register - Bit Width of DMA Master Bus 4’h4: 64 Bits	R
b11 to b8	bLcd_BI	Core Identification Register - DMA Master Bus Interface 4’h1: AHB_BUS	R
b7 to b0	bLcd_REV	Core Identification Register - Revision 8’h0F: Version 1.15	R

8.4.21 Coding Palette (Palette registers) Description

CAUTION

- The coding palette depends on the selected color mode.
- The symbol mapped to the address of the coding palette is different for each mode.
- Also, the meaning (function) of the fields depends on the color mode.
- The tables below provide information for all color modes.

See **Figure 8.7, LCD Output Formatting, bLcd_BPP: 1, 2, 4, 8** table.

See **Figure 8.8, LCD Output Formatting, bLcd_BPP: 16, 18, 24** table.

See **Section 8.5.8, Palette Lookup Table**.

See **Section 8.5.9, Output FIFO and Formatter**.

8.4.21.1 rLcd_PAL_RGB_555 — Coding Palette when RGB 5:5:5 Mode

Coding Palette setting, bLcd_RGB = 0, bLcd_OPS0 = 1 in rLcd_CR1 register

Address: 5300 4200h - 5300 43FCh

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	bLcd_RED1					bLcd_GREEN1					bLcd_BLUE1				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	bLcd_RED0					bLcd_GREEN0					bLcd_BLUE0				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.23 rLcd_PAL_RGB_555 Register Contents

Bit Position	Bit Name	Function	R/W
b31	Reserved	Read as 0.	R
b30 to b26	bLcd_RED1	Red Data palette: 5 bits	R/W
b25 to b21	bLcd_GREEN1	Green Data palette: 5 bits	R/W
b20 to b16	bLcd_BLUE1	Blue Data palette: 5 bits	R/W
b15	Reserved	Read as 0.	R
b14 to b10	bLcd_RED0	Red Data palette: 5 bits	R/W
b9 to b5	bLcd_GREEN0	Green Data palette: 5 bits	R/W
b4 to b0	bLcd_BLUE0	Blue Data palette: 5 bits	R/W

8.4.21.2 rLcd_PAL_RGB_565 — Coding Palette when RGB 5:6:5 Mode

Coding Palette setting, bLcd_RGB = 0, bLcd_OPS0 = 0 in rLcd_CR1 register

Address: 5300 4200h - 5300 43FCh

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bLcd_RED1					bLcd_GREEN1						bLcd_BLUE1				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_RED0					bLcd_GREEN0						bLcd_BLUE0				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.24 rLcd_PAL_RGB_565 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b27	bLcd_RED1	Red Data palette: 5 bits	R/W
b26 to b21	bLcd_GREEN1	Green Data palette: 6 bits	R/W
b20 to b16	bLcd_BLUE1	Blue Data palette: 5 bits	R/W
b15 to b11	bLcd_RED0	Red Data palette: 5 bits	R/W
b10 to b5	bLcd_GREEN0	Green Data palette: 6 bits	R/W
b4 to b0	bLcd_BLUE0	Blue Data palette: 5 bits	R/W

8.4.21.3 rLcd_PAL_BGR_555 — Coding Palette when BGR 5:5:5 Mode

Coding Palette setting, bLcd_RGB = 1, bLcd_OPS0 = 1 in rLcd_CR1 register

Address: 5300 4200h - 5300 43FCh

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	bLcd_BLUE1					bLcd_GREEN1					bLcd_RED1				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	bLcd_BLUE0					bLcd_GREEN0					bLcd_RED0				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.25 rLcd_PAL_BGR_555 Register Contents

Bit Position	Bit Name	Function	R/W
b31	Reserved	Read as 0.	R
b30 to b26	bLcd_BLUE1	Blue Data palette: 5 bits	R/W
b25 to b21	bLcd_GREEN1	Green Data palette: 5 bits	R/W
b20 to b16	bLcd_RED1	Red Data palette: 5 bits	R/W
b15	Reserved	Read as 0.	R
b14 to b10	bLcd_BLUE0	Blue Data palette: 5 bits	R/W
b9 to b5	bLcd_GREEN0	Green Data palette: 5 bits	R/W
b4 to b0	bLcd_RED0	Red Data palette: 5bits	R/W

8.4.21.4 rLcd_PAL_BGR_565 — Coding Palette when BGR 5:6:5 Mode

Coding Palette setting, bLcd_RGB = 1, bLcd_OPS0 = 0 in rLcd_CR1 register

Address: 5300 4200h - 5300 43FCh

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bLcd_BLUE1					bLcd_GREEN1						bLcd_RED1				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bLcd_BLUE0					bLcd_GREEN0						bLcd_RED0				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8.26 rLcd_PAL_BGR_565 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b27	bLcd_BLUE1	Blue Data palette: 5 bits	R/W
b26 to b21	bLcd_GREEN1	Green Data palette: 6 bits	R/W
b20 to b16	bLcd_RED1	Red Data palette: 5 bits	R/W
b15 to b11	bLcd_BLUE0	Blue Data palette: 5 bits	R/W
b10 to b5	bLcd_GREEN0	Green Data palette: 6 bits	R/W
b4 to b0	bLcd_RED0	Red Data palette: 5 bits	R/W

8.5 Operation

8.5.1 Main Features Description

The LCD Controller Synoptic figure depicts the LCD Controller architecture.

The LCD Controller is first initialized by the processor via the Slave Bus interface. This interface is a read/write interface in which the LCD Controller can “only respond and not initiate bus transactions”.

Minimal setup of the Control and Status Registers are:

- The timing registers for horizontal and vertical timing signals (rLcd_HTR, rLcd_VTR1 and rLcd_VTR2)
- The DMA Base Address Registers (rLcd_DBAR)
- The Pixel Clock Timing Register (rLcd_PCTR)

After that, bLcd_LCE in the control register (rLcd_CR1) must be set and the LCD Controller runs, accessing frame buffer memory and processing and piping the data through to the display.

If the Palette is used, it must first be loaded. The Palette can be loaded statically by the processor via the Slave Interface.

The start of each frame begins with an internal start sync pulse from the Timing & Control Unit, coincident with the vertical synchronization signal. This start sync pulse initiates the DMA controller to start accessing data from frame buffer memory via the DMA Master Interface. The start sync pulse also initiates the Pixel Unpack to start accepting data from the read side of the input FIFO.

The Master Interface initiates read transactions with the master AHB bus. There are programmable options for 4, 8, 16 words (64 bits word) bursts read lengths to improve bus utilization. Received frame data is written into the input FIFO. The FIFO bridges the two clock domains, with the LCD panel usually running at LCD_ECLK with tolerances, and the processor system bus running at LCD_HCLK.

The Pixel Unpack will unpack 1, 2, 4, 8, 16, 18, or 24 bits per pixel (bpp) words from the 64-bit frame buffer word. Depending on the bpp programming and whether the Palette is used, the pixel data is sent to the Output FIFO either via the Palette (where it is transformed via a color lookup table) or directly from Pixel Unpack.

The Output FIFO queues ready pixels for synchronization with the LCD panel timing signals. The output of the FIFO drives the Output Formatter, which formats the RGB data for 18/24-bit panel interfaces.

The remaining sections present more detailed information of each block processing of the pixel data flow.

8.5.2 Bandwidth Limitation

The principal limitation on the LCD Panel supported is the available memory bandwidth.

The LCD Controller supports a variety of user programmable LCD panel resolutions and encoded pixel size which determine the frame buffer memory size and system interconnect bandwidth requirements. Table LCD Frame Buffer Memory Size lists various LCD panel resolutions versus frame buffer bpp selection, and the required memory size. Table LCD Bandwidth lists the corresponding required frame buffer bandwidth.

Table 8.27 LCD Bandwidth

	QVGA	WQVGA	VGA	WVGA	SVGA	WSVGA	XGA
Horizontal	320	480	640	800	800	1024	1024
Vertical	240	272	480	480	600	600	768
BitPerPixel	Bandwidth (MB/sec @ 60 Hz)						
1	0.6	1.0	2.3	2.9	3.6	4.6	5.9
2	1.2	2.0	4.6	5.8	7.2	9.2	11.8
4	2.3	3.9	9.2	11.5	14.4	18.4	23.6
8	4.6	7.8	18.4	23.0	28.8	36.8	47.2
16	9.2	15.7	36.9	46.1	57.6	73.7	94.4*1
24 (Packed)	13.8	23.5	55.3	69.1	86.4*1	110.5*1	141.6*1,*2
18 / 24 (Non-Packed)	18.4	31.3	73.7	92.2*1	115.2*1	147.4*1,*2	188.7*1,*2

Note 1. Underflow can be occurred at 4-burst transfer.

Note 2. Underflow can be occurred at 8-burst transfer.

Table 8.28 LCD Frame Buffer Memory Size

	QVGA	WQVGA	VGA	WVGA	SVGA	WSVGA	XGA
Horizontal	320	480	640	800	800	1024	1024
Vertical	240	272	480	480	600	600	768
BitPerPixel	Frame Buffer Memory Requirement (KiB)						
1	9.4	15.9	37.5	46.9	58.6	75.0	96.0
2	18.8	31.9	75.0	93.8	117.2	150.0	192.0
4	37.5	63.8	150.0	187.5	234.4	300.0	384.0
8	75.0	127.5	300.0	375.0	468.8	600.0	768.0
16	150.0	255.0	600.0	750.0	937.5	1200.0	1536.0
24 (Packed)	225.0	382.5	900.0	1125.0	1406.3	1800.0	2304.0
18 / 24 (Non-Packed)	300.0	510.0	1200.0	1500.0	1875.0	2400.0	3072.0

8.5.3 Timing and Control

The Timing and Control Unit utilizes the Horizontal Timing Register (rLcd_HTR) and Vertical Timing Registers 1 & 2 (rLcd_VTR1, rLcd_VTR2) and optionally Hor. & Vert. Timing Extension Register (rLcd_HVTER) and Horizontal Timing Override Register (rLcd_HPPLOR) to generate timing signals LCD_VSYNC, LCD_HSYNC, and LCD_DE to the LCD panel. The Timing Unit remains inactive till bLcd_LCE in Control Register 1 (rLcd_CR1) goes active. At that point, the Timing & Control Unit runs till bLcd_LCE is de-asserted. At that time, the timing unit will keep running till the end of the current frame, and then orderly shutdown. The Timing Unit can be re-activated with bLcd_LCE re-asserted, but often power to the display must be re-cycled. This can be accomplished by bLcd_LPE in Control Register 1 (rLcd_CR1) connected as an enable to an external power source for the LCD panel.

bLcd_LCE also plays a role in Power Sequencing. On startup, while bLcd_LCE is inactive, timing signals LCD_PCLK, LCD_HSYNC, LCD_VSYNC, LCD_DE and data signals LCD_R[7:0], LCD_G[7:0], LCD_B[7:0] are held to logic zero. On bLcd_LCE shutdown, after the current frame being displayed completes and the Timing Unit halt these same signals are forced to logic zero. At that point power can safely be removed from the LCD panel.

The Timing Unit provides interrupt bLcd_VCT which triggers on one of four timing trigger points during the vertical scan period. The point of triggering is programmable via register Interrupt Scan Compare Register (rLcd_ISCR).

The figure below gives the definition of different parameters of LCD.

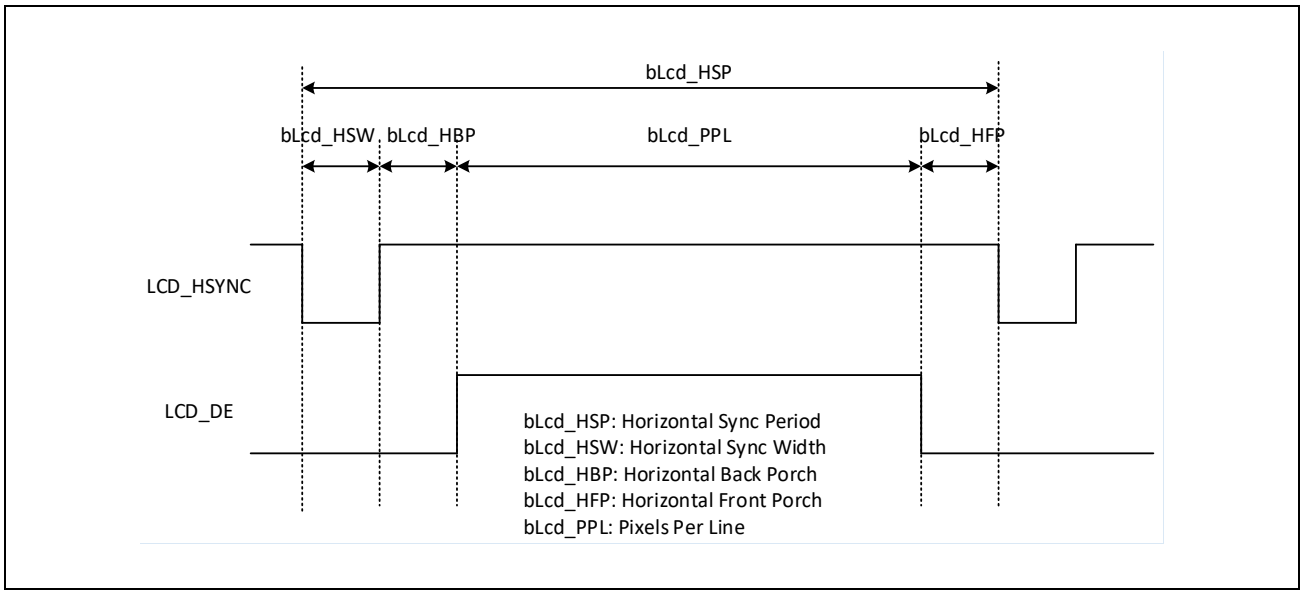

Figure 8.2 LCD Horizontal Timing

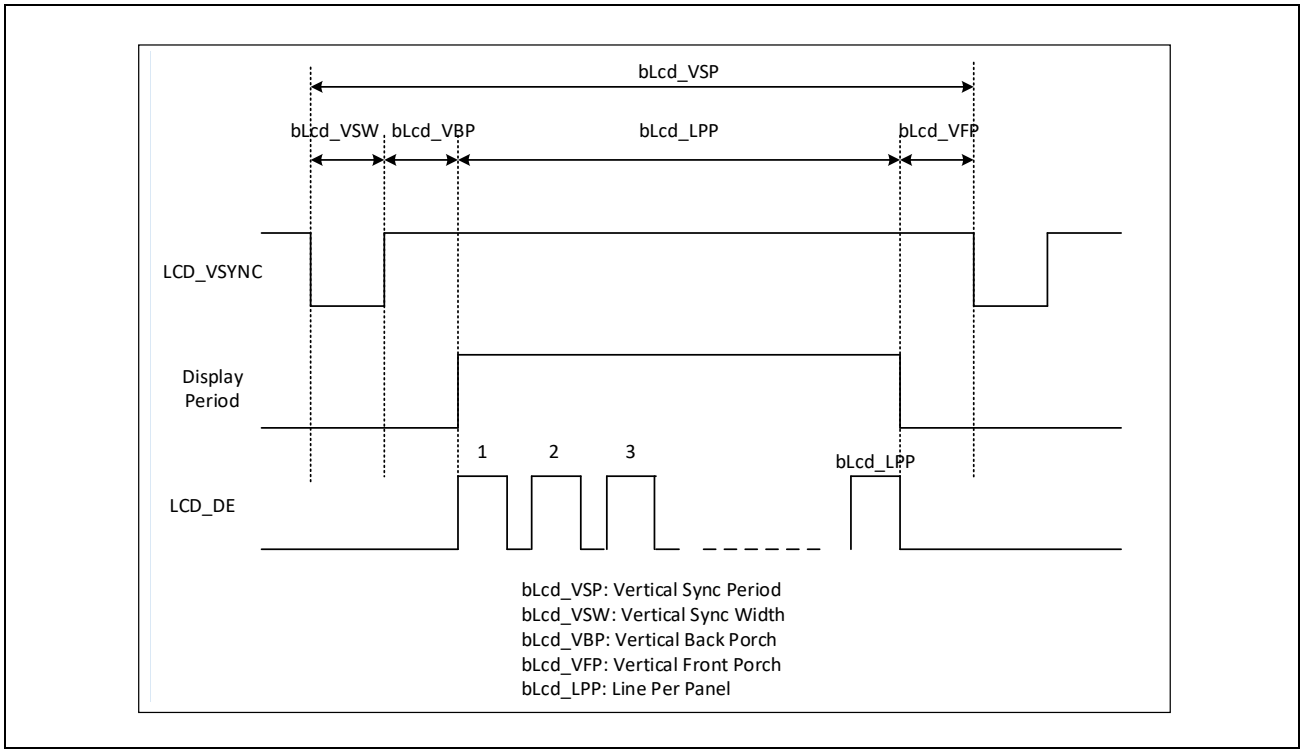

Figure 8.3 LCD Vertical Timing

8.5.4 DMA Controller and Memory Interface

The DMA Controller initializes via the internal frame start pulse with the transfer of the DMA Base Address Register (rLcd_DBAR) to the DMA Current Address Register (rLcd_DCAR) and the commencement of the first memory transfer transaction. The numbers of words in a burst are programmed by bLcd_FDW in Control Register 1 (rLcd_CR1). Based on bLcd_FDW and the number of empty words in the FIFO, a service request by the DMA Controller to the Master Interface initiates a frame buffer read.

Software must program the rLcd_DEAR register with the frame buffer end address. The DMA Controller will keep reading frame buffer words until the current address in rLcd_DCAR equals rLcd_DEAR. At that point, the DMA Controller will halt until the next frame, where upon it will read from frame buffer starting with the address in rLcd_DBAR.

8.5.5 Frame Buffer Organization

The frame buffer memory is located in external memory (DDR or embedded SRAM). The frame buffer attached to the DMA Master Interface provides encoded or un-encoded pixels for display on the LCD panel.

CAUTION

- DMA Master interface connection has data bus width of 64 bits.
- See **Figure 8.1, LCD Controller Synoptic**.

8.5.6 Input FIFO

The size of input FIFO is 1K word depth by 64-bit width memory, see **Figure 8.1, LCD Controller Synoptic**.

The DMA Controller and Master Interface control the write side on the Bus Clock (LCD_HCLK) domain, while the Pixel Unpack controls the read side on the Pixel Clock (LCD_ECLK) domain. Address pointers are used for FIFO empty and full flag calculations as well as when there are 4, 8 or 16 empty word locations.

Based on the bLcd_FDW in Control Register 1 (rLcd_CR1) and the number of empty FIFO locations, a service request for 4-, 8-, or 16-word bursts from memory is issued by the DMA Controller to the Master Interface.

Interrupts bLcd_IFO (Input FIFO - Overrun) and bLcd_IFU (Input FIFO - Underrun) trigger whenever there is a FIFO write with no empty locations or read with no valid data. The write side by design cannot overrun the FIFO. The read side unpack logic can cause an underrun, but this is due to insufficient Master Bus bandwidth or frame buffer memory response, causing the input FIFO to go empty while there is a request for data by the unpack logic.

The first indication of a FIFO underrun due to insufficient Master Bus bandwidth or frame buffer memory response is the bLcd_OFU (Output FIFO - Underrun).

8.5.7 Pixel Unpack

The Pixel Unpack reads 32-bit data from the input FIFO and extracts 1, 2, 4, 8, 16, 18, or 24 bits per pixel data depending on the bLcd_BPP in Control Register 1 (rLcd_CR1). Note that 1, 2, 4, 8 bpp are encoded pixels that index an entry into the palette while 16, 18, 24 bpp are un-encoded pixels that directly drive the LCD panel via the Output Formatter. The LCD Controller supports big endian, little endian data formats.

With each frame, the internal start sync pulse from the Timing & Control Unit initializes the Pixel Unpack to start de queuing words from the Input FIFO as they appear on the read side.

The table below lists the structure of the data in each Frame Buffer Word in the input FIFO corresponding to the endian and bLcd_BPP programming combinations. For each of the three supported data formats, the Pixel Unpack extracts the appropriate display pixel from the data word.

The following list the three data types:

- LEB_LEP: Little Endian Frame Buffer Byte, placed in Little Endian Pixel Byte
  - bLcd_EBO = 0 and bLcd_EPO = 0
- BEB_BEP: Big Endian Frame Buffer Byte, placed in Big Endian Pixel Byte
  - bLcd_EBO = 1 and bLcd_EPO = x
  - There is a difference in 1, 2, 4, 8, 16 bpp only
- LEB_BEP: Little Endian Frame Buffer Byte, placed in Big Endian Pixel Byte
  - bLcd_EBO = 0 and bLcd_EPO = 1
  - There is a difference in 1, 2, 4 bpp only

LCD Pixel Unpack and Data Structure
Little Endian Frame Buffer Byte(LEB) placed in Little Endian Pixel Byte(LEP)
bLcd_EBO=0, bLcd_EPO=0

BPP	Input FIFO Read Side Output Bits															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	p31	p30	p29	p28	p27	p26	p25	p24	p23	p22	p21	p20	p19	p18	p17	p16
2	p15		p14		p13		p12		p11		p10		p9		p8	
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
4	p7				p6				p5				p4			
	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
8	p3								p2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
16	p1															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18	p0															
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	17	16
24	p0															
	-	-	-	-	-	-	-	-	23	22	21	20	19	18	17	16

BPP	Input FIFO Read Side Output Bits															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	p15	p14	p13	p12	p11	p10	p9	p8	p7	p6	p5	p4	p3	p2	p1	p0
2	p7		p6		p5		p4		p3		p2		p1		p0	
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
4	p3				p2				p1				p0			
	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
8	p1								p0							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
16	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
24	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 8.4 Pixel Unpack, Little Endian Frame Buffer Byte placed in Little Endian Pixel Byte

LCD Pixel Unpack and Data Structure
Big Endian Frame Buffer Byte(BEB) placed in Big Endian Pixel Byte(BEP)
bLcd_EBO=1, bLcd_EPO=x

BPP	Input FIFO Read Side Output Bits															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15
2	p0		p1		p2		p3		p4		p5		p6		p7	
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
4	p0				p1				p2				p3			
	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
8	p0								p1							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
16	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18	p0															
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	17	16
24	p0															
	-	-	-	-	-	-	-	-	23	22	21	20	19	18	17	16

BPP	Input FIFO Read Side Output Bits															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	p16	p17	p18	p19	p20	p21	p22	p23	p24	p25	p26	p27	p28	p29	p30	p31
2	p8		p9		p10		p11		p12		p13		p14		p15	
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
4	p4				p5				p6				p7			
	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
8	p2								p3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
16	p1															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
24	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 8.5 Pixel Unpack, Big Endian Frame Buffer Byte placed in Big Endian Pixel Byte

LCD Pixel Unpack and Data Structure
Little Endian Frame Buffer Byte(LEB) placed in Big Endian Pixel Byte(BEP)
bLcd_EBO=0, bLcd_EPO=1

BPP	Input FIFO Read Side Output Bits															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	p24	p25	p26	p27	p28	p29	p30	p31	p16	p17	p18	p19	p20	p21	p22	p23
2	p12		p13		p14		p15		p8		p9		p10		p11	
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
4	p6				p7				p4				p5			
	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
8	p3								p2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
16	p1															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18	p0															
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	17	16
24	p0															
	-	-	-	-	-	-	-	-	23	22	21	20	19	18	17	16

BPP	Input FIFO Read Side Output Bits															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	p8	p9	p10	p11	p12	p13	p14	p15	p0	p1	p2	p3	p4	p5	p6	p7
2	p4		p5		p6		p7		p0		p1		p2		p3	
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
4	p2				p3				p0				p1			
	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
8	p1								p0							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
16	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
24	p0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 8.6 Pixel Unpack, Little Endian Frame Buffer Byte placed in Big Endian Pixel Byte

8.5.8 Palette Lookup Table

The Palette is a 256 entry by 16-bit lookup table implemented as a two port 128 entry by 32-bit RAM. One port ties in with the Bus Clock (LCD_HCLK) domain, the Slave Interface via the processor can fill the Palette. The second port ties in with the Pixel Clock (LCD_ECLK) domain, enabling the Palette RAMs contents to be indexed by the Pixel Unpack encoded pixel output. The Palette's output flows to the Output Formatter.

CAUTION

- DMA Master interface connection has data bus width of 64 bits.
- See **Figure 8.1, LCD Controller Synoptic**.

Selection of which 16-bit half of the 32-bit Palette entry is determined by the endian setting and the least significant bit of the indexing encoded pixel input. bLcd_EBO in Control Register 1 (rLcd_CR1) determines the endian setting.

- In little endian mode, when the input index encoded pixel least significant bit is zero, the lower 16-bit palette entry is selected.
  - See **Figure 8.6, Pixel Unpack, Little Endian Frame Buffer Byte placed in Big Endian Pixel Byte** table.

- In big endian mode, when the input index encoded pixel least significant bit is zero, the upper 16-bit palette entry is selected.
  - See **Figure 8.5, Pixel Unpack, Big Endian Frame Buffer Byte placed in Big Endian Pixel Byte** table.

The internal format of the Palette Data Words (rLcd_PAL_BGR_555, rLcd_PAL_BGR_565, rLcd_PAL_RGB_555, rLcd_PAL_RGB_565 registers) is shown **8.4.21, Coding Palette (Palette registers) Description**. There are four formats, interpreted according to LCD Palette Data Word Interpretation by bLcd_OPS and bLcd_RGB table below.

Note that bLcd_OPS and bLcd_RGB are in Control Register 1 (rLcd_CR1).

- bLcd_OPS[0] controls the Green bit selection of 5 or 6 bits.
- bLcd_RGB controls the RGB or BGR swapping of the Red or Blue fields.

Table 8.29 LCD Palette Data Word Interpretation by bLcd_OPS and bLcd_RGB

bLcd_OPS[0], bLcd_RGB	RGB Format	Red/Blue, Swap/NoSwap
2'b00	RGB 5:6:5	No Swap
2'b01	BGR 5:6:5	Swap
2'b10	RGB 5:5:5	No Swap
2'b11	BGR 5:5:5	Swap

8.5.9 Output FIFO and Formatter

The Output Formatter contains an output FIFO which comprises of a 16 word by 24-bit memory. Depending on the bits per pixel programming, the incoming selection is either the Pixel Unpack (16, 18, 24 bpp) or the Palette (1, 2, 4, 8 bpp).

The output FIFO is slave to the Pixel Unpack which drives pixels to it either directly or through the Palette. The output FIFO provides back pressure pipeline freeze capability when it cannot accept another pixel for queuing. This allows the LCD Controller to pre-fetch frame buffer data at the start of a frame, filling up both input and output FIFOs and then freezing till the first line is ready to display. Once the first Data Enable (LCD_DE) signal from the Timing & Control Unit is active, the output FIFO read side continuously reads for the remainder of each active horizontal line period. These reads in turn re-activate the Pixel Unpack and subsequently the DMA Controller to access frame buffer data on a demand basis.

The Output Formatter interprets the pixel read from the output FIFO according to bLcd_BPP, bLcd_OPS and bLcd_RGB defined in Control Register 1 (rLcd_CR1) and listed in tables below.

BPP: bLcd_BPP
OPS: bLcd_OPS
RGB: bLcd_RGB

LCD Output Formatting
BPP: 1, 2, 4, 8 (Palette)

BPP	OPS[1:0]	RGB	RGB Format	BGR Format	Pins	Note
1	00	0	5:6:5	-	LCD_R[5:1] LCD_G[5:0] LCD_B[5:1]	RGB defined in Palette
2	00	0		-		RGB defined in Palette
4	00	0		-		RGB defined in Palette
8	00	0		-		RGB defined in Palette
1	10	0	5:6:5	-	LCD_R[7:3] LCD_G[7:2] LCD_B[7:3]	RGB defined in Palette
2	10	0		-		RGB defined in Palette
4	10	0		-		RGB defined in Palette
8	10	0		-		RGB defined in Palette
1	01	0	5:5:5	-	LCD_R[5:1] LCD_G[5:1] LCD_B[5:1]	RGB defined in Palette
2	01	0		-		RGB defined in Palette
4	01	0		-		RGB defined in Palette
8	01	0		-		RGB defined in Palette
1	11	0	5:5:5	-	LCD_R[7:3] LCD_G[7:3] LCD_B[7:3]	RGB defined in Palette
2	11	0		-		RGB defined in Palette
4	11	0		-		RGB defined in Palette
8	11	0		-		RGB defined in Palette
1	00	1	-	5:6:5	LCD_R[5:1] LCD_G[5:1] LCD_B[5:1]	Palette B,R swapped
2	00	1	-			Palette B,R swapped
4	00	1	-			Palette B,R swapped
8	00	1	-			Palette B,R swapped
1	10	1	-	5:6:5	LCD_R[7:3] LCD_G[7:3] LCD_B[7:3]	Palette B,R swapped
2	10	1	-			Palette B,R swapped
4	10	1	-			Palette B,R swapped
8	10	1	-			Palette B,R swapped
1	01	1	-	5:5:5	LCD_R[5:1] LCD_G[5:1] LCD_B[5:1]	Palette B,R swapped
2	01	1	-			Palette B,R swapped
4	01	1	-			Palette B,R swapped
8	01	1	-			Palette B,R swapped
1	11	1	-	5:5:5	LCD_R[7:3] LCD_G[7:3] LCD_B[7:3]	Palette B,R swapped
2	11	1	-			Palette B,R swapped
4	11	1	-			Palette B,R swapped
8	11	1	-			Palette B,R swapped

Figure 8.7 LCD Output Formatting, bLcd_BPP: 1, 2, 4, 8

BPP: bLcd_BPP
OPS: bLcd_OPS
RGB: bLcd_RGB

LCD Output Formatting
BPP: 16, 18, 24 (Frame Buffer)

BPP	OPS[1:0]	RGB	RGB Format	BGR Format	Pins	Note
16	00	-	5:6:5	-	LCD_R[5:1] LCD_G[5:0] LCD_B[5:1]	Frame Buffer Direct
16	01	-	5:5:5	-	LCD_R[5:1] LCD_G[5:1] LCD_B[5:1]	Frame Buffer Direct
16	10	-	5:6:5	-	LCD_R[7:3] LCD_G[7:2] LCD_B[7:3]	Frame Buffer Direct
16	11	-	5:5:5	-	LCD_R[7:3] LCD_G[7:3] LCD_B[7:3]	Frame Buffer Direct
18	0-	-	6:6:6	-	LCD_R[5:0] LCD_G[5:0] LCD_B[5:0]	Frame Buffer Direct
18	1-	-	6:6:6	-	LCD_R[7:2] LCD_G[7:2] LCD_B[7:2]	Frame Buffer Direct
24	-	-	8:8:8	-	LCD_R[7:0] LCD_G[7:0] LCD_B[7:0]	Frame Buffer Direct

Figure 8.8 LCD Output Formatting, bLcd_BPP: 16, 18, 24

CAUTION

- Map onto most significant bits of 18-bit RGB interface to LCD panel when bpp < 18 and bLcd_OPS[1] =0. Unused output pins driven to zero.
- Map onto most significant bits of 24-bit RGB interface to LCD panel when bpp < 18 and bLcd_OPS[1] =1. Unused output pins driven to zero.

Both the write side and the read side of the output FIFO are on the Pixel Clock (LCD_ECLK) domain. Grey encoded address pointers are used for FIFO empty and full flag calculations as well as the look-ahead pipeline freeze signal.

Interrupts bLcd_OFO (Output FIFO - Overrun) and bLcd_OFU (Output FIFO - Underrun) trigger whenever there is a FIFO write with no empty locations or read with no valid data. The write side logic by design cannot overrun (because of the back pressure pipeline freeze capability). While tested, this protection remains for potential error analysis. The read side can cause an bLcd_OFU interrupt, and the cause of this could be inadequate bus bandwidth in accessing frame buffer data.

8.5.10 Initializing Configuration Registers

To change or initialize configuration registers, the LCD Controller provides the following sequencing support:

(1) Clear bLcd_LCE (LCD Disable). LCD timing unit waits for current frame display to finish before forcing panel signals to logic 0. Internally the LCD Controller core holds the following signals to logic zero:

- LCD_VSYNC
- LCD_HSYNC
- LCD_DE
- LCD_PCLK
- LCD_R[7:0]
- LCD_B[7:0]
- LCD_G[7:0]

(2) All configuration registers must be programmed during this step to avoid LCD malfunctions. The registers configurations lists are:

- rLcd_CR1 register except bLcd_LCE (Already cleared to 0, LCD Disable) and bLcd_LPE (depending on power mode sequence required)
- rLcd_HTR, rLcd_VTR1 and rLcd_VTR2 registers (Horizontal and vertical timing)
- rLcd_HPPLOR and rLcd_HVTER registers (Extended horizontal and vertical timing)
- Pixel Clock RESET released to inactive (no reset state) in rLcd_PCTR register
- rLcd_PWMDCR_0, rLcd_PWMDCR_1, rLcd_PWMFR_0, rLcd_PWMFR_1 PWM0 & 1 registers (PWM control)
- rLcd_DBAR register (DMA base address)
- Palette registers initialization

(3) After initializing configuration registers, bLcd_LCE in Control Register 1 (rLcd_CR1) is set to on. With bLcd_LCE on, the signals to the LCD panel listed in (1) are free to drive to their programmed active levels and first frame fetch and display commences.

8.5.11 Interrupts

There are three coordinated interrupt registers; The Interrupt Status Register (rLcd_ISR), the Interrupt Mask Register (rLcd_IMR), and the Interrupt Vector Register (rLcd_IVR). The rLcd_ISR and rLcd_IMR are both read/write registers while the rLcd_IVR is read only.

Any of the internally generated interrupts set a corresponding bit in the rLcd_ISR. If the error's corresponding mask bit is set in the rLcd_IMR, then the corresponding error bit in the rLcd_IVR register will set, generating an interrupt to the processor. The processor interrupt handler can respond by reading the rLcd_IVR to determine the particular interrupt to process. At the end of an interrupt response, the programming can reset the interrupt in the rLcd_ISR by writing logic 1 to the corresponding interrupt bit in the rLcd_ISR. Through the rLcd_IMR register the programming has full control over which interrupts to enable.

8.5.12 Power Sequencing

The LCD Controller provides the following power up sequencing support:

(4) Power is applied to the VLSI device containing the LCD Controller and the LCD panel. Internally the LCD Controller holds the following signals to logic zero:
- LCD_VSYNC
- LCD_HSYNC
- LCD_DE
- LCD_PCLK
- LCD_R[7:0]
- LCD_B[7:0]
- LCD_G[7:0]

(5) Optionally, all configuration registers must be programmed during this step to avoid LCD malfunctions. The registers configurations lists are:
- rLcd_CR1 register
- rLcd_HTR, rLcd_VTR1 and rLcd_VTR2 registers (Horizontal and vertical timing)
- rLcd_HPPLOR and rLcd_HVTER registers (Extended horizontal and vertical timing)
- Pixel Clock RESET released to inactive (no reset state) in rLcd_PCTR register
- rLcd_PWMDCR_0, rLcd_PWMDCR_1, rLcd_PWMFR_0, rLcd_PWMFR_1 PWM0 & 1 registers (PWM control)
- rLcd_DBAR register (DMA base address)
- Palette registers initialization

(6) After a pre-determined amount of time specified by the LCD panel and controlled by a processor timer, bLcd_LPE in Control Register 1 (rLcd_CR1) is set to on. With bLcd_LPE on, the signals to the LCD panel listed in (1) are free to drive to their programmed active levels.

The LCD Controller provides the following power down sequencing support:

(1) bLcd_LPE in Control Register 1 (rLcd_CR1) is set to off.

(2) After the current frame being displayed completes, the signals to the LCD panel listed above are forced to zero.

(3) At the time, the signals to the LCD panel are forced to zero, interrupt bLcd_LDD is generated, signaling frame completion. After a pre-determined amount of time specified by the LCD panel, power to the display can be removed.

Note that bLcd_LPE in Control Register 1 (rLcd_CR1), connected as an enable to an external power source for the LCD panel, can be used for enabling and disabling power to the LCD panel.

8.5.13 Frame Buffer 24 bpp Packed Word

The LCD Controller can drive a 24 bpp LCD Panel with an unpacked or packed 24-bit pixel within a 32-bit Frame Buffer Word. For an unpacked 24-bit pixel within a 32-bit word, the frame buffer words are organized as follows:

Table 8.30 LCD Frame Buffer Organization, bLcd_FBP = 0 and bLcd_BPP = 3'b110

	Frame Buffer Contents: Unpacked data 24 bpp mode: ●bLcd_FBP = 0 and bLcd_BPP = 3'b110	
Frame Buffer Base Address	Bit 31:24	Bit 23:0
32'h0	Unused	Pixel 0 (24-bits Pixel Data: RGB only)
32'h4	Unused	Pixel 1 (24-bits Pixel Data: RGB only)
....	....	....

Thus, for each 32-bit Frame Buffer Word, the most significant byte is unused.

No Swap mode (only RGB) available in this configuration.

The LCD Controller contains a programming mode whereby when bLcd_FBP in Control Register 1 (rLcd_CR1) is set, the 24 bpp Pixel in a 32-bit Frame Buffer Word can be packed according to table below:

Table 8.31 LCD Frame Buffer Organization, bLcd_FBP = 1 and bLcd_BPP = 3'b110

	Frame Buffer Contents: Packed data 24 bpp mode: ● bLcd_FBP = 1 and bLcd_BPP = 3'b110, see Warning below			
Frame Buffer Base Address	Bit 31:24	Bit 23:0	Bit 15:8	Bit 7:0
32'h0	P1-Byte0: B	P0-Byte2: R	P0-Byte1: G	P0-Byte0: B
32'h4	P2-Byte1: G	P2-Byte0: B	P1-Byte2: R	P1-Byte1: G
32'h8	P3-Byte2: R	P3-Byte1: G	P3-Byte0: B	P2-Byte2: R
....	....	....		

No Swap mode (only RGB) available in this configuration.

Note that Pn = Pixel n, Bytek = Byte k (within the pixel). Thus, for each successive three 32-bit Frame Buffer Words, four 24-bit pixels can be unpacked by the LCD Controller.

CAUTION

The packed mode (bLcd_FBP = 1) must be used with 24 bpp mode only (bLcd_BPP = 3'b110).

If firmware does not respect this restriction, the LCD Controller function is not guaranteed.

8.5.14 Pulse Width Modulation

With the advent of LED for backlighting of TFT LCD Panels, two additional Pulse Width Modulation (PWM0 & 1) modules are added to the LCD Controller. Typically, a DC-DC converter provides the constant current to the LEDs, and the converter contains a brightness input. Modulating the brightness input with a PWM signal trades off power consumed by the panels versus brightness.

The reference clock of PWM0 & 1 module is LCD_ECLK. The desired PWM frequency is configured with PWM Frequency Clock Divider Register (rLcd_PWMFR_0 and rLcd_PWMFR_1), which in turn is modulated in pulse width by the PWM Duty Cycle Register (rLcd_PWMDCR_0 and rLcd_PWMDCR_1).

Since the Control / Status Registers are clocked by the Slave Bus LCD_HCLK while the PWM modules are clocked by LCD_ECLK, Clock Domain Crossing logic first transfers CSR signals to the selected PWM_CLK domain, and then on the start of the PWM first PWM_CLK cycle, transferred to Holding Registers. In this way, the PWM can be dynamically changed in frequency and duty cycle.

8.5.15 Blink Function

The main objective of blink function is the flashing part of picture.

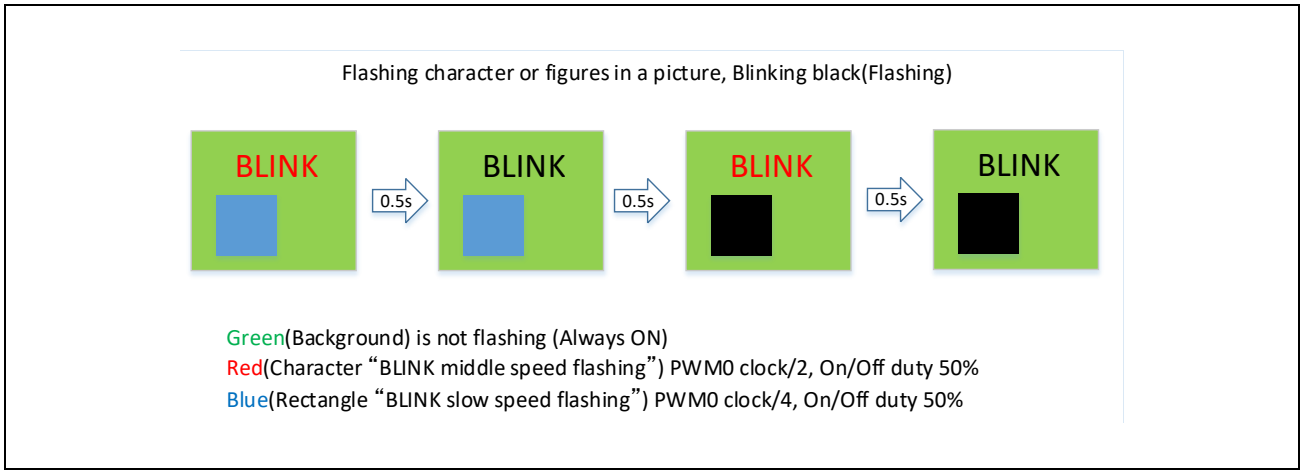

Figure 8.9 LCD Blink Main Principle

In figure Blink BL[1:0] Attribute Management, it can see in memory frame buffer (inside DDR/SRAM), the mapping of each pixel (R,G,B on 24 bits) with or without blink function active.

The blink function is controlled by:

- Blink Enable or Disable controlled by bLcd_BlinkOn in rLcd_GPIOR register.
  - The Pixel data presented on LCD_R[7:0], LCD_G[7:0], LCD_B[7:0] depends on whether the blink function is disabled or enabled.
  - See **Figure 8.10, Blink BL[1:0] Attribute Management**.
- Blink Brightness Mode is controlled by bLcd_BlinkMode in rLcd_GPIOR register.
  - The blink frequency depends on bLcd_PWMFCD_0 and bLcd_PWMFCE_0 inside rLcd_PWMFR_0 register.
- In this case, the output of this PWM0 unit drives the BLINK logic.
- The blink function can be used in 24 bpp mode only.

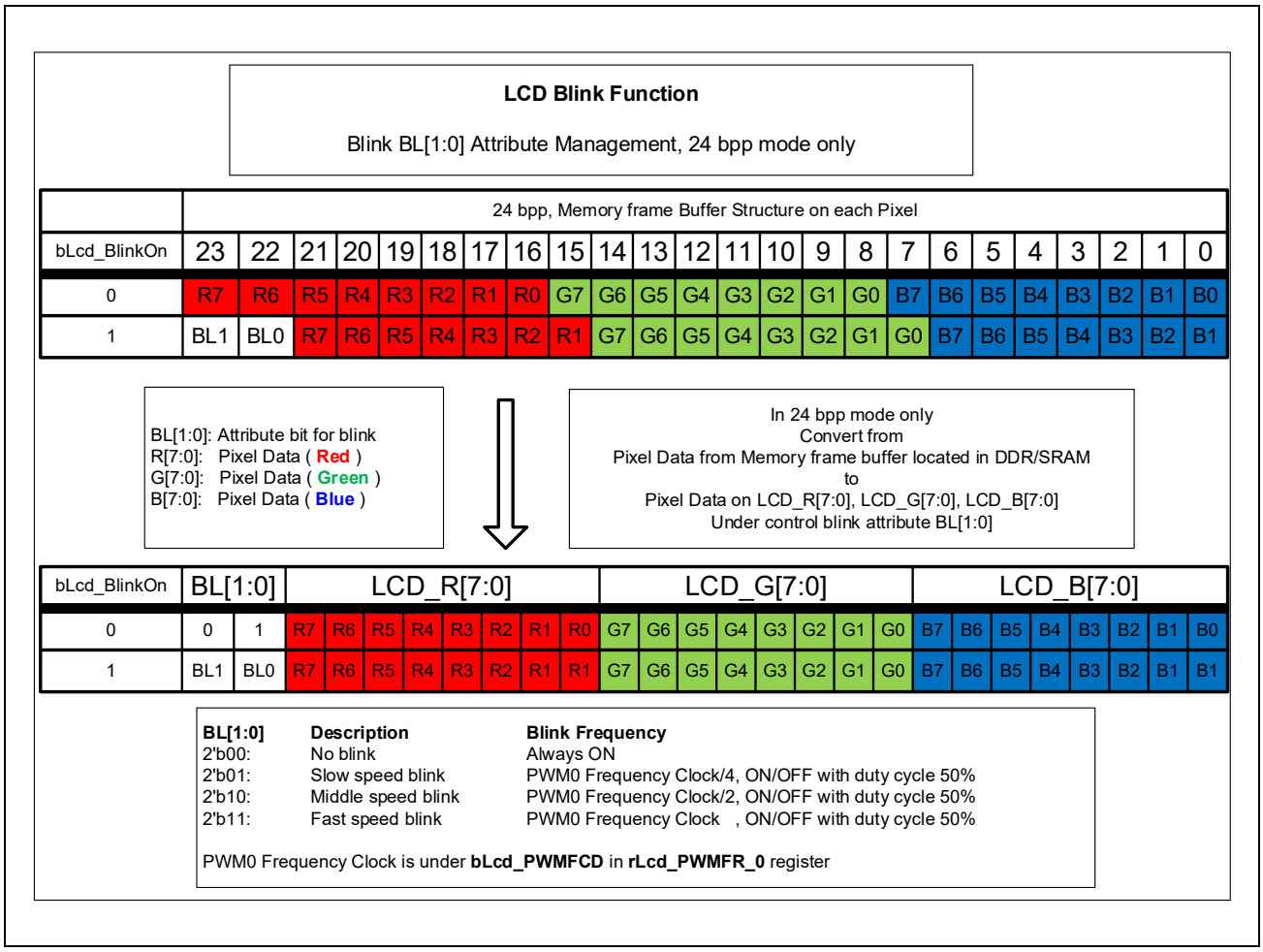

Figure 8.10 Blink BL[1:0] Attribute Management

8.5.16 Limitation

LCD Controller has following limitation:

- Odd number of pixels per line should not be used (strictly forbidden).
- Due to bandwidth limitation between LCD Controller & DDR2/3, an underrun input or output FIFO can be detected, to solve it, please:
  - Increase bLcd_FDW value (burst size configuration of DMA read request in word).
  - Verify if LCD resolution targeted is compatible with bandwidth available on DDR2/3 or SRAM.
    1) Depends on bpp (bits per pixel) and Refresh rate
    2) Frequency limitation max 83.3 MHz
    3) Bandwidth limitation between LCD & DDR2/3

    See **Section 8.5.2, Bandwidth Limitation**.
- Concerning configuration of bLcd_FDW parameter recommended to avoid bandwidth issue:
  - bLcd_FDW = 4 must be used only for low resolution
  - bLcd_FDW = 16 must be used only for high resolution
- Pixel Packed Mode
  - The packed mode (bLcd_FBP = 1) must be used with 24 bpp mode only (bLcd_BPP = 3’b110).
  - If firmware does not respect this restriction, the Packed Mode function is not guaranteed, RGB outputs may be wrong.
  - Swap mode is not available in this mode.
- Blink Function
  - The blink function must be used with 24 bpp mode only (bLcd_BPP = 3’b110).
  - If firmware does not respect this restriction, the blink function is not guaranteed, RGB outputs may be wrong.
  - Blink function reuse all resources of PWM0 unit.
    PWM0 unit drives the BLINK frequency.

Section 9 Semaphore

9.1 Overview

The semaphores are a group of R/W registers used to configure the hardware lock mechanism which regulates the exclusivity access of all internal shared resources (Buffers pools, memory regions and peripherals).

Before to use any shared resource, a processor must become the owner of this with a hardware lock semaphore.

64 individual hardware lock semaphores are available and resources are shared between 4 CPUs (includes external CPU).

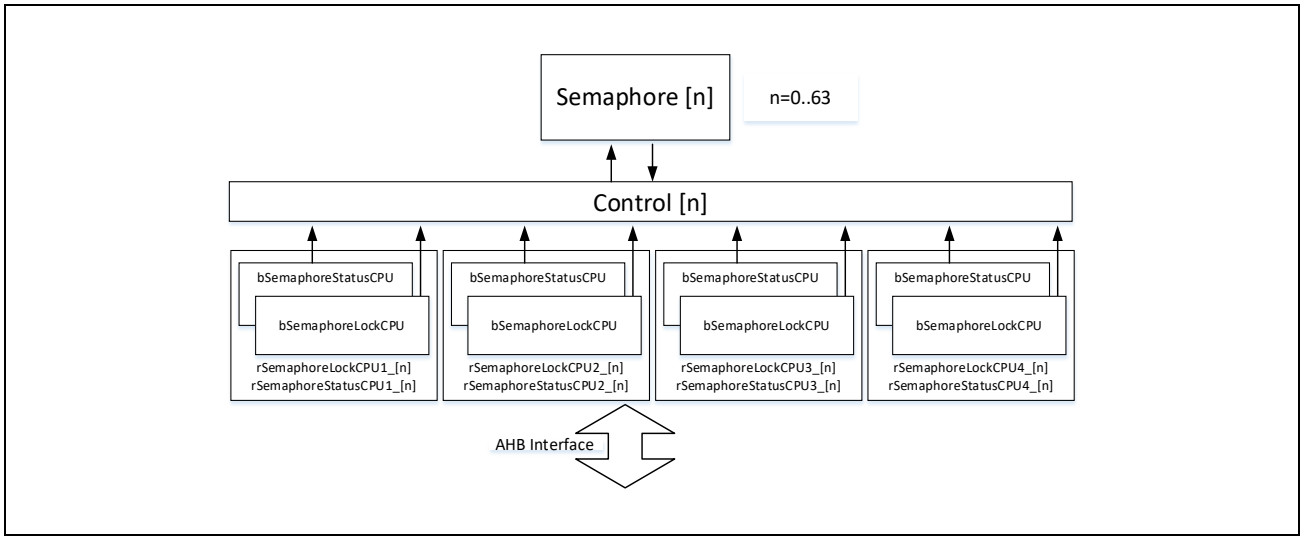

Figure 9.1 Semaphore Synoptic

9.2 Signal Interfaces

Table 9.1 Signal Interface

Signal Name	Input Output	Description
Clock		
SEMAP_HCLK	Input	Internal bus clock (AHB)

9.3 Register Map

Table 9.2 Register Map

Address	Register Symbol	Register Name
5300 0000h + 10h × n	rSemaphoreLockCPU1_[n] (n = 0..63)	Semaphore Lock CPU1 Register [n]
5300 0004h + 10h × n	rSemaphoreStatusCPU1_[n] (n = 0..63)	Semaphore Status CPU1 Register [n]
5300 1000h + 10h × n	rSemaphoreLockCPU2_[n] (n = 0..63)	Semaphore Lock CPU2 Register [n]
5300 1004h + 10h × n	rSemaphoreStatusCPU2_[n] (n = 0..63)	Semaphore Status CPU2 Register [n]
5300 2000h + 10h × n	rSemaphoreLockCPU3_[n] (n = 0..63)	Semaphore Lock CPU3 Register [n]
5300 2004h + 10h × n	rSemaphoreStatusCPU3_[n] (n = 0..63)	Semaphore Status CPU3 Register [n]
5300 3000h + 10h × n	rSemaphoreLockCPU4_[n] (n = 0..63)	Semaphore Lock CPU4 Register [n]
5300 3004h + 10h × n	rSemaphoreStatusCPU4_[n] (n = 0..63)	Semaphore Status CPU4 Register [n]

9.4 Register Description

9.4.1 rSemaphoreLockCPU[m]_[n] — Semaphore Lock CPU[m] Register [n]

With CPU[m] (m = 1..4) and Semaphore[n] (n = 0..63).

CPU[m] is distinguished not by actual CPU but address only.

The semaphore in free state will be reserved for the CPU which reads.

Only the reserved CPU for the semaphore can release it by writing with the value 3’b000.

Address: rSemaphoreLockCPU1_[n]: 5300 0000h + 10h × n
rSemaphoreLockCPU2_[n]: 5300 1000h + 10h × n
rSemaphoreLockCPU3_[n]: 5300 2000h + 10h × n
rSemaphoreLockCPU4_[n]: 5300 3000h + 10h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	bSemaphoreLockCPU		
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9.3 rSemaphoreLockCPU[m]_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b3	Reserved		R
b2 to b0	bSemaphoreLockCPU	For each Semaphore [n] Read by CPU 3’b000 Semaphore is free Hardware reservation for CPU 3’b100 Semaphore is already reserved by CPU1 3’b101 Semaphore is already reserved by CPU2 3’b110 Semaphore is already reserved by CPU3 3’b111 Semaphore is already reserved by CPU4 Write by CPU 3’b000 Release semaphore Not 3’b000 No action	R/W

9.4.2 rSemaphoreStatusCPU[m]_[n] — Semaphore Status CPU[m] Register [n]

With CPU[m] (m = 1..4) and Semaphore[n] (n = 0..63).

Address: rSemaphoreStatusCPU1_[n]: 5300 0004h + 10h × n
rSemaphoreStatusCPU2_[n]: 5300 1004h + 10h × n
rSemaphoreStatusCPU3_[n]: 5300 2004h + 10h × n
rSemaphoreStatusCPU4_[n]: 5300 3004h + 10h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	bSemaphoreStatusCPU		
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9.4 rSemaphoreStatusCPU[m]_[n] Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b3	Reserved		R
b2 to b0	bSemaphoreStatusCPU	For each Semaphore [n] Read a current value of semaphore 3'b000 Semaphore is free 3'b001, 3'b010, 3'b011 Reserved 3'b100 Semaphore is reserved by CPU1 3'b101 Semaphore is reserved by CPU2 3'b110 Semaphore is reserved by CPU3 3'b111 Semaphore is reserved by CPU4	R

9.5 Operation

9.5.1 Semaphore [n] (n = 0..63)

This state machine manages the reservation and the release of semaphore.

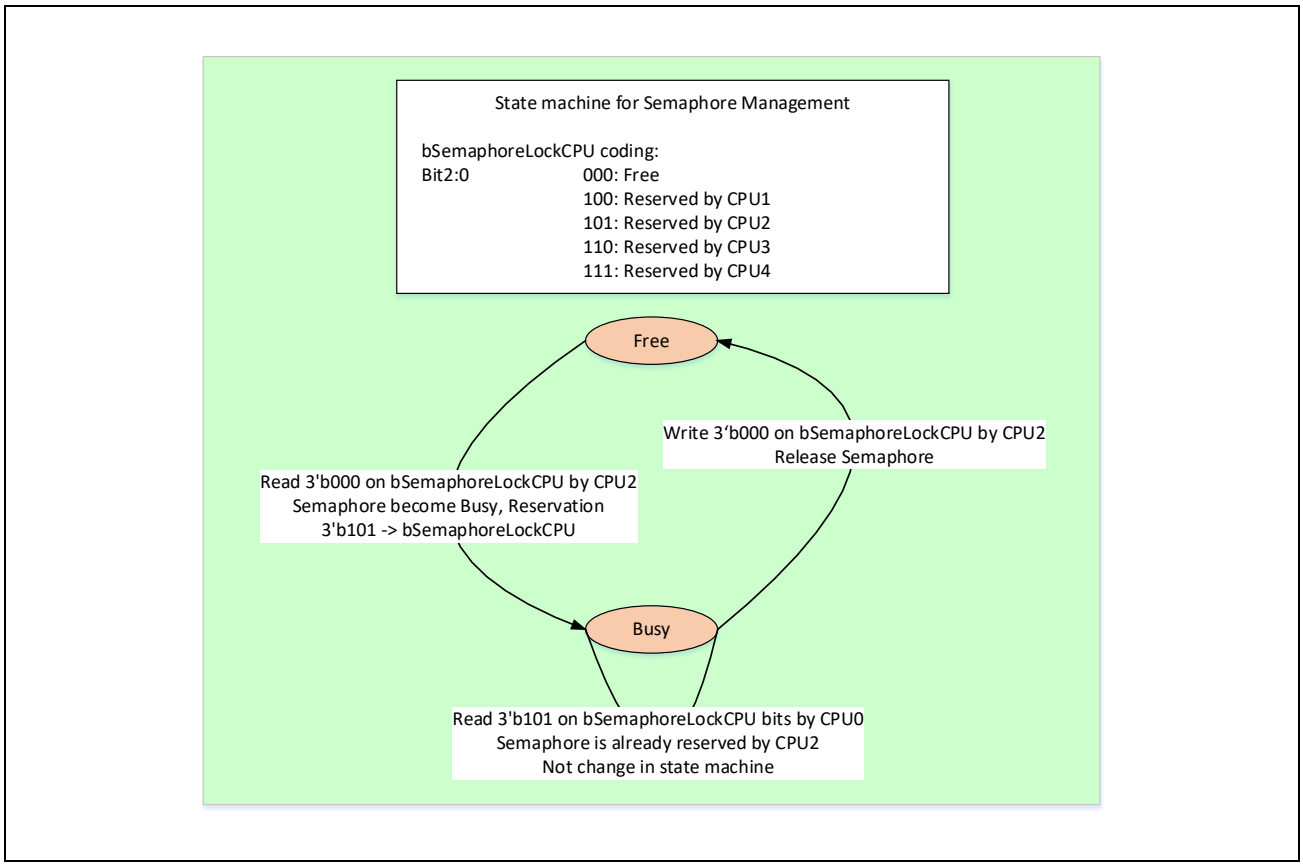

Figure 9.2 Semaphore State Machine

9.5.2 CPU Identify and Address Decoding

A same Semaphore [n] have 4 separate addresses to be access.

The Semaphore [n] can be accessed by the registers.

- rSemaphoreLockCPU1_[n]
- rSemaphoreLockCPU2_[n]
- rSemaphoreLockCPU3_[n]
- rSemaphoreLockCPU4_[n]

The semaphore state machine uses this address to write a semaphore identify in bSemaphoreLockCPU

- 3’b100: Semaphore reserved by CPU1
- 3’b101: Semaphore reserved by CPU2
- 3’b110: Semaphore reserved by CPU3
- 3’b111: Semaphore reserved by CPU4

To have a good management of semaphore identify, each CPU must use a different address to access in Semaphore.

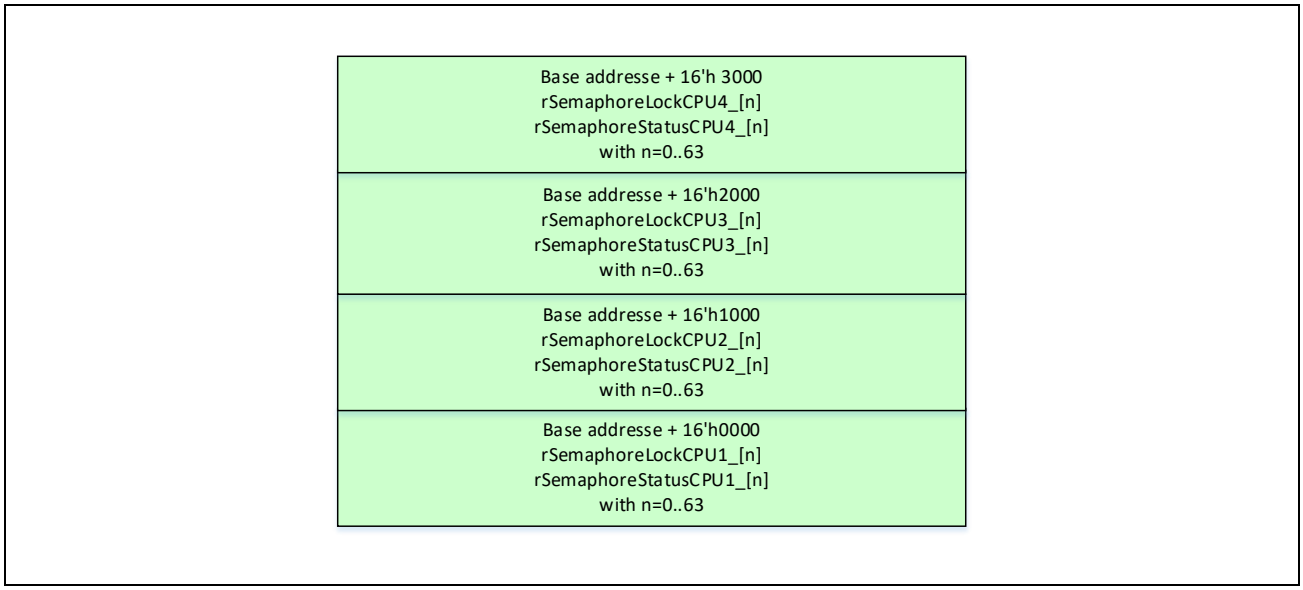

Figure 9.3 Semaphore Address Block for Identify CPU

9.6 Usage Notes

To use a Semaphore in several CPUs, the basic procedures are following below:

Example of Semaphore55 by CPU1 and CPU2:

(1) CPU1 reads bSemaphoreLockCPU of rSemaphoreLockCPU1_55 register.

- Test if 3'b000
  - If the read value is 3'b000, Semaphore55 is reserved for CPU1. CPU1 can access exclusivity of all internal shared resources (Buffer pools, memory regions and peripherals).
  - If the read value is 3'b101, Semaphore55 is already reserved for CPU2, CPU1 must wait.

(2) CPU2 reads bSemaphoreLockCPU of rSemaphoreLockCPU2_55 register.

- Test if 3'b000
  - If the read value is 3'b000, Semaphore55 is reserved for CPU2. CPU2 can access exclusivity of all internal shared resources.
  - If the read value is 3'b100, Semaphore55 is already reserved for CPU1, CPU2 must wait.

In this example, CPU1 and CPU2 use the Semaphore55 to share a resource:

- CPU1 manages the Semaphore55 with rSemaphoreLockCPU1_55 register
- CPU2 manages the Semaphore55 with rSemaphoreLockCPU2_55 register

Semaphore55 provides different addresses to CPU1 and 2, the hardware uses the semaphore addresses to identify the CPU.

The semaphore function must be used with following allocation:

- CPU1 must use only rSemaphoreLockCPU1_[n] to manage the Semaphore [n]
- CPU2 must use only rSemaphoreLockCPU2_[n] to manage the Semaphore [n]
- CPU3 must use only rSemaphoreLockCPU3_[n] to manage the Semaphore [n]
- CPU4 must use only rSemaphoreLockCPU4_[n] to manage the Semaphore [n]

CAUTION

Different CPUs should not use the same address to access Semaphore55.

Section 10 Medium Speed External Bus Interface (MSEBI)

10.1 Overview

The MSEBI is fully programmable. It has 4 chip selects, address/data/control-data are multiplexed. The 32-bit data bus can be configured to interface 8, 16 and 32 bits external devices.

MSEBI is composed of 2 independents blocks:

- Master
- Slave

Master Mode feature:

The Medium Speed External Bus Interface (MSEBI) manages the signals which control the access to external asynchronous and synchronous peripheral devices. Separate read and write control signals allow asynchronous direct memory and peripheral interfacing. It also provides an external wait request reception (from slave) capability and DMA coupling. On the same board, synchronous and asynchronous mode, data bus width (32, 16, or 8 bits) can be mixed.

Slave Mode feature:

The Medium Speed External Bus Interface (MSEBI) manages all signals allowing the decode access from the bus in synchronous mode only. It also provides an external wait request transmission (to master) capability and external DMA request transmission (to master) capability.

3 basics modes are available with different configurations:

- Multiplexed 32 bits on data bus (MSEBI Mode32)
- Multiplexed 16 bits on data bus (MSEBI Mode16)
- Multiplexed 8 bits on data bus (MSEBI Mode8)

Chip selects CS0_N to CS3_N can be configured to have a programmable address capability until 4 GB, depending on configuration.

For each of slave and master mode, the CPU can disconnect all pins not used by MSEBI interface. When these pins are not used, this feature allows using these pins as general purpose inputs or outputs.

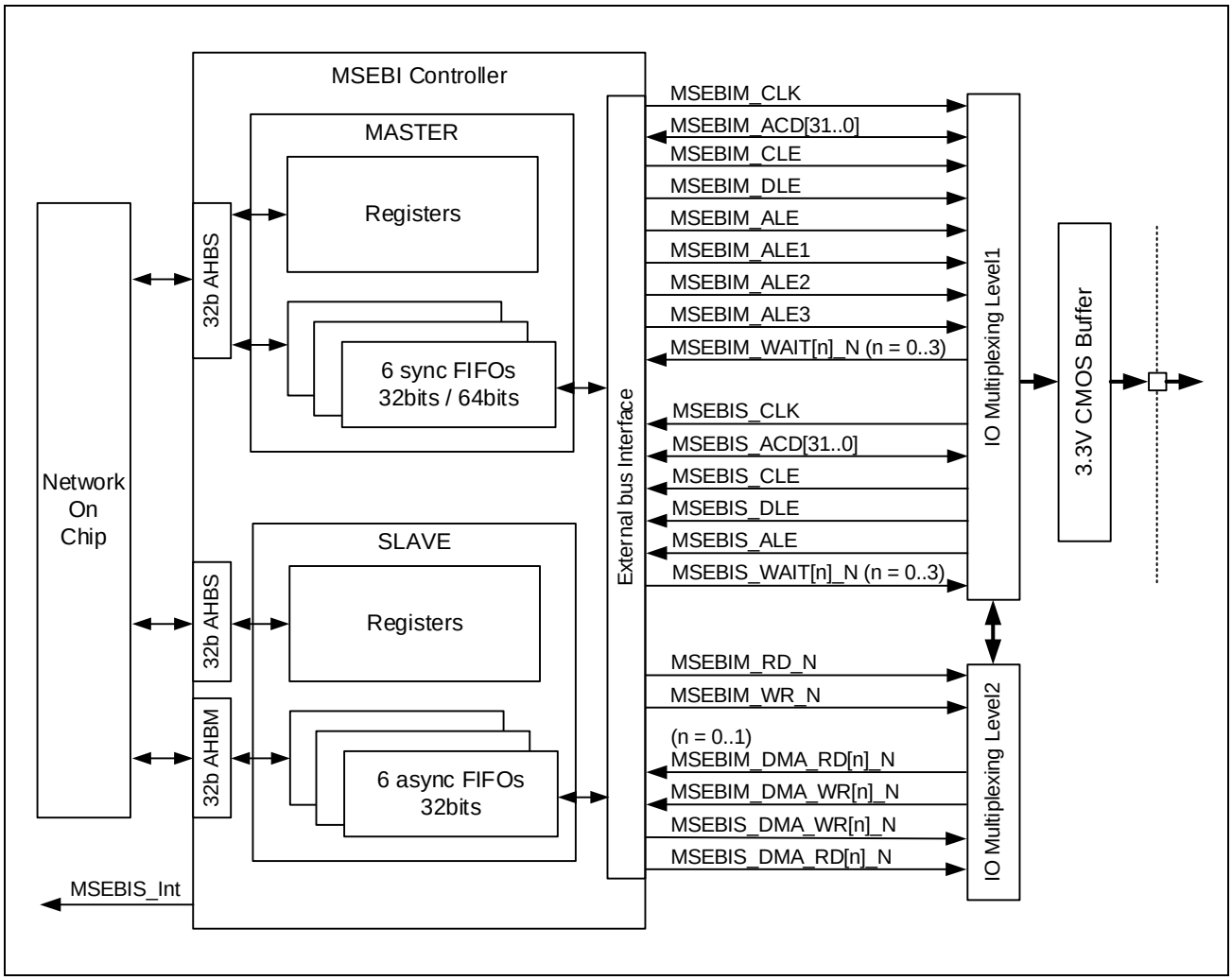

Figure 10.1 MSEBI Synoptic

The main features are:

- Data bus width selectable from 8, 16 and 32 bits
- Separate enable control of the master and slave part
- Asynchronous and synchronous mode
- Multi DLE mode
- Burst mode
- DMA coupling
  - Peripheral flow controller mode
  - 4 DMA channel available (external request reception capability)
- Concerning FIFO size in master mode, we have
  - CPU transmit and receive FIFO for master: 2 × 32 Words × 32bits
  - DMA transmit and receive FIFO for master: 4 × 32 Words × 64bits
- Concerning FIFO size in slave mode, we have
  - CPU transmit and receive FIFO for slave: 2 × 32 Words × 32bits
  - DMA transmit and receive FIFO for slave: 4 × 32 Words × 32bits
- Up to 4 chip select Lines
- Programmable address capability from 2 B...4 GB
- Programmable setup time
- Programmable hold time
- External wait request (can be enabled or disabled)

10.1.1 Signal Interfaces

Table 10.1 Signal Interface

Signal Name	Input Output	Description
Clock		
MSEBIM_HCLK	Input	Internal AHB Bus Clock (Master)
MSEBIS_HCLK	Input	Internal AHB Bus Clock (Slave)
Interrupt		
MSEBIS_Int	Output	Level sensitive interrupt output, Active High
External Signal (Master Mode)		
MSEBIM_ACD[31:0]	I/O	Address, Control and Data multiplexed
MSEBIM_CLK	Output	Global configurable clock, reference for all timings.
MSEBIM_ALE	Output	Address Latch Enable (active high)
MSEBIM_ALE1	Output	Address Latch Enable (active high)
MSEBIM_ALE2	Output	Address Latch Enable (active high)
MSEBIM_ALE3	Output	Address Latch Enable (active high)
MSEBIM_CLE	Output	Address and Control Latch Enable (active high)
MSEBIM_DLE	Output	Data Latch Enable (active high)
MSEBIM_RD_N	Output	Read enable (active low)
MSEBIM_WR_N	Output	Write enable (active low)
MSEBIM_WAIT[n]_N	Input	Wait insertion in current cycle with n = 0..3 (active low)
MSEBIM_DMA_RD[n]_N	Input	DMA request dedicated to MSEBI_CS[n]_N in read mode with n = 0..1
MSEBIM_DMA_WR[n]_N	Input	DMA request dedicated to MSEBI_CS[n]_N in write mode with n = 0..1
External Signal (Slave Mode)		
MSEBIS_ACD[31:0]	I/O	Address, Control and Data multiplexed
MSEBIS_CLK	Input	Clock given by the master of the bus
MSEBIS_ALE	Input	Address Latch Enable (active high)
MSEBIS_CLE	Input	Address and Control Latch Enable (active high)
MSEBIS_DLE	Input	Data Latch Enable (active high)
MSEBIS_WAIT[n]_N	Output	Wait insertion in current cycle with n = 0..3 (active low)
MSEBIS_DMA_RD[n]_N	Output	DMA request dedicated to MSEBI_CS[n]_N in read mode with n = 0..1
MSEBIS_DMA_WR[n]_N	Output	DMA request dedicated to MSEBI_CS[n]_N in write mode with n = 0..1

Note: In case of GPIO Multiplexed Pin Name, index[n] is added at the end.
Ex) MSEBIM_WAIT_N[n], MSEBIS_DMA_RD_N[n]

10.1.2 MSEBI Master Address Mapping of CS[n] from CPU

Writing or reading data to the area described in the following table pushes a command to the CPU transmit FIFO and generates an MSEBI access for each MSEBI_CS[n]_N (n = 0..3) according to the address.

Table 10.2 Address Mapping of CS[n] from CPU

CS[n]	Base Address
CS0	6000 0000h
CS1	6800 0000h
CS2	7000 0000h
CS3	7800 0000h

10.1.3 Multiplexed Signal Interface

Data multiplexer on MSEBIM_ACD is controlled by MSEBIM_ALE, MSEBIM_CLE, MSEBIM_DLE and latched on rising edge of MSEBIM_CLK. Details to the associated signals are described in the following tables.

Note that the access is depending on the bus size. Please refer to the following table for details:

- **Table 10.4, MSEBI Mode32, Multiplexer Function on ACD31..0**
- **Table 10.6, MSEBI Mode16, Multiplexer Function on ACD15..0**
- **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**

Table 10.3 Multiplexed Signal Interface (1/2)

Signal Name	Description
MSEBIM_D[31:16]	Data (Used only in Mode32)
MSEBIM_D[15:8]	Data (Used only in Mode32 or Mode16)
MSEBIM_D[7:0]	Data
MSEBIM_A[31:0]	Address
MSEBI_BE[n]_N	Byte Enable (active low) • Mode32 with n = 0..3: BE3_N BE2_N BE1_N BE0_N 0 0 0 0 Word 32 bits 1 1 0 0 Lower halfword 16 bits 0 0 1 1 Upper halfword 16 bits 1 1 1 0 Lower byte 1 1 0 1 Lower byte +1 1 0 1 1 Lower byte +2 0 1 1 1 Upper byte, Lower byte+3 1 1 1 1 Reserved • Mode16 with n = 0..1: BE1_N BE0_N 0 0 HalfWord 16 bits 1 0 Lower byte 0 1 Upper byte 1 1 Reserved • Mode8: No Byte enable in this mode
MSEBI_CS[n]_N	Chip select, 4GB programmable address capability for each (active low) • Concerning address capability All chip selects (n = 0..3) can be configured to have a programmable address capability depending on its configuration. Extended address capability modes are important when user wants reduce the optional number of ALE phases to increase the bandwidth and reduce the latency.
MSEBI_CSREG_N	Access to the global registers to manage interruptions (from Master to Slave) and status (active low) ***Caution*)** In both modes, master and slave, when the master is accessing the slave's shared registers (using MSEBI_CSREG_N), no prefetch allowed. Following decoding tables describe how to access to the shared registers of a specific chip select with n = 0..3 MSEBI_CS[n]_N MSEBI_CSREG_N CS[n]_N Access Type 0 0 Access CS[n]_N shared registers 0 1 Access CS[n]_N memory space 1 0 Reserved 1 1 No access

Table 10.3 Multiplexed Signal Interface (2/2)

Signal Name	Description
MSEBI_DMA_N	Give identification of initiator of the current request (DMA or CPU) and must be associated with CS[n]_N (n = 0..3) See details in **Table 10.47, Slave Detection of Request Initiator**.
MSEBI_R/W_N	Read Write control 1: Read Access 0: Write Access

10.1.3.1 Mode32 Multiplexer

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface

Table 10.4 MSEBI Mode32, Multiplexer Function on ACD31..0

MSEBI_ACD Bus Multiplexed	Stage ADDRESS Controlled by ALE (Optional)	Stage CONTROL Controlled by CLE	Stage DATA Controlled by DLE
MSEBI(x)_ALE Serial mode only	1'b1	1'b0	1'b0
MSEBIM_ALE Parallel mode only	1'b1	1'b0	1'b0
MSEBIM_ALE1 Parallel mode only	1'b0	1'b0	1'b0
MSEBIM_ALE2 Parallel mode only	1'b0	1'b0	1'b0
MSEBIM_ALE3 Parallel mode only	1'b0	1'b0	1'b0
MSEBI(x)_CLE	1'b0	1'b1	1'b0
MSEBI(x)_DLE	1'b0	1'b0	1'b1
Assignment on ACD			
MSEBI(x)_ACD[31:11]	1'b0	MSEBI_A[22:2]	MSEBI_D[31:11]
MSEBI(x)_ACD10	1'b0	MSEBI_BE3_N	MSEBI_D10
MSEBI(x)_ACD9	1'b0	MSEBI_BE2_N	MSEBI_D9
MSEBI(x)_ACD8	MSEBI_A31	MSEBI_BE1_N	MSEBI_D8
MSEBI(x)_ACD7	MSEBI_A30	MSEBI_BE0_N	MSEBI_D7
MSEBI(x)_ACD6	MSEBI_A29	MSEBI_R/W_N	MSEBI_D6
MSEBI(x)_ACD5	MSEBI_A28	MSEBI_DMA_N	MSEBI_D5
MSEBI(x)_ACD4	MSEBI_A27	MSEBI_CSREG_N	MSEBI_D4
MSEBI(x)_ACD3	MSEBI_A26	MSEBI_CS3_N	MSEBI_D3
MSEBI(x)_ACD2	MSEBI_A25	MSEBI_CS2_N	MSEBI_D2
MSEBI(x)_ACD1	MSEBI_A24	MSEBI_CS1_N	MSEBI_D1
MSEBI(x)_ACD0	MSEBI_A23	MSEBI_CS0_N	MSEBI_D0

Table 10.5 MSEBI Mode32, Chip Selects Management

Extended Address Capability Available with 0 ALE phase			
4 Chip Selects 8 MB	3 Chip Selects 16 MB	2 Chip Selects 32 MB	1 Chip Select 64 MB
MSEBI_CS3_N	MSEBI_A23	MSEBI_A23	MSEBI_A23
MSEBI_CS2_N	MSEBI_CS2_N	MSEBI_A24	MSEBI_A24
MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_A25
MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N

ALE phase is optional to increase bandwidth and reduce latency for each chip select CS[n]_N (n = 0..3).

For Master Mode

CPU (n = 0..3)

- MSEBI_A26 .. MSEBI_A2 are directly driven by CPU (128 MB for each CS[n]_N)
- MSEBI_A31 .. MSEBI_A27 are driven by following registers:
  - rMSEBIM_CONFIG_CS[n]_N

DMA (n = 0..1)

- MSEBI_A31 .. MSEBI_A2 are driven by following registers:
  - rMSEBIM_ADDRDMA_CURRENTREAD_CS[n]_N
  - rMSEBIM_ADDRDMA_CURRENTWRITE_CS[n]_N

10.1.3.2 Mode16 Multiplexer

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface

On ALE columns:

- If multiple ALE stages are used, the symbol #k in the column represents the order in which the master will send the ALE phases to the slave, i.e. #1 is the first ALE, #2 the second, etc.

Table 10.6 MSEBI Mode16, Multiplexer Function on ACD15..0

MSEBI_ACD Bus Multiplexed	Stage ADDRESS Controlled by ALE (Optional, #2)	Stage ADDRESS Controlled by ALE (Optional, #1)	Stage CONTROL Controlled by CLE	Stage DATA Controlled by DLE
MSEBI(x)_ALE Serial mode only	1'b1	1'b1	1'b0	1'b0
MSEBIM_ALE Parallel mode only	1'b0	1'b1	1'b0	1'b0
MSEBIM_ALE1 Parallel mode only	1'b1	1'b0	1'b0	1'b0
MSEBIM_ALE2 Parallel mode only	1'b0	1'b0	1'b0	1'b0
MSEBIM_ALE3 Parallel mode only	1'b0	1'b0	1'b0	1'b0
MSEBI(x)_CLE	1'b0	1'b0	1'b1	1'b0
MSEBI(x)_DLE	1'b0	1'b0	1'b0	1'b1
Assignment on ACD				
MSEBI(x)_ACD[15:9]	1'b0	MSEBI_A[23:17]	MSEBI_A[7:1]	MSEBI_D[15:9]
MSEBI(x)_ACD8	1'b0	MSEBI_A16	MSEBI_BE1_N	MSEBI_D8
MSEBI(x)_ACD7	MSEBI_A31	MSEBI_A15	MSEBI_BE0_N	MSEBI_D7
MSEBI(x)_ACD6	MSEBI_A30	MSEBI_A14	MSEBI_R/W_N	MSEBI_D6
MSEBI(x)_ACD5	MSEBI_A29	MSEBI_A13	MSEBI_DMA_N	MSEBI_D5
MSEBI(x)_ACD4	MSEBI_A28	MSEBI_A12	MSEBI_CSREG_N	MSEBI_D4
MSEBI(x)_ACD3	MSEBI_A27	MSEBI_A11	MSEBI_CS3_N	MSEBI_D3
MSEBI(x)_ACD2	MSEBI_A26	MSEBI_A10	MSEBI_CS2_N	MSEBI_D2
MSEBI(x)_ACD1	MSEBI_A25	MSEBI_A9	MSEBI_CS1_N	MSEBI_D1
MSEBI(x)_ACD0	MSEBI_A24	MSEBI_A8	MSEBI_CS0_N	MSEBI_D0

Table 10.7 MSEBI Mode16, Chip Selects Management

Extended Address Capability Available with 0 ALE phase			
4 Chip Selects 256 B	3 Chip Selects 512 B	2 Chip Selects 1 KB	1 Chip Select 2 KB
MSEBI_CS3_N	MSEBI_A8	MSEBI_A8	MSEBI_A8
MSEBI_CS2_N	MSEBI_CS2_N	MSEBI_A9	MSEBI_A9
MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_A10
MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N

Extended Address Capability Available with 1 ALE phase			
4 Chip Selects 16 MB	3 Chip Selects 32 MB	2 Chip Selects 64 MB	1 Chip Select 128 MB
MSEBI_CS3_N	MSEBI_A24	MSEBI_A24	MSEBI_A24
MSEBI_CS2_N	MSEBI_CS2_N	MSEBI_A25	MSEBI_A25
MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_A26
MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N

ALE phase is optional to increase bandwidth and reduce latency for each chip select CS[n]_N (n = 0..3).

For Master Mode

CPU (n = 0..3)

- MSEBI_A26... MSEBI_A1 are directly driven by CPU (128 MB for each CS[n]_N)
- MSEBI_A31... MSEBI_A27 are driven by following registers:
  - rMSEBIM_CONFIG_CS[n]_N

DMA (n = 0..1)

- MSEBI_A31... MSEBI_A1 are driven by following registers:
  - rMSEBIM_ADDRDMA_CURRENTREAD_CS[n]_N (n = 0..1)
  - rMSEBIM_ADDRDMA_CURRENTWRITE_CS[n]_N (n = 0..1)

10.1.3.3 Mode8 Multiplexer

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface

On ALE columns:

- If multiple ALE stages are used, the symbol #k in the column represents the order in which the master will send the ALE phases to the slave, i.e. #1 is the first ALE, #2 the second, etc.

Table 10.8 MSEBI Mode8, Multiplexer Function on ACD7..0

MSEBI_ACD Bus Multiplexed	Stage ADDRESS Controlled by ALE (Optional, #4)	Stage ADDRESS Controlled by ALE (Optional, #3)	Stage ADDRESS Controlled by ALE (Optional, #2)	Stage ADDRESS Controlled by ALE (Optional, #1)	Stage CONTROL Controlled by CLE	Stage DATA Controlled by DLE
MSEBI(x)_ALE Serial mode only	1'b1	1'b1	1'b1	1'b1	1'b0	1'b0
MSEBIM_ALE Parallel mode only	1'b0	1'b0	1'b0	1'b1	1'b0	1'b0
MSEBIM_ALE1 Parallel mode only	1'b0	1'b0	1'b1	1'b0	1'b0	1'b0
MSEBIM_ALE2 Parallel mode only	1'b0	1'b1	1'b0	1'b0	1'b0	1'b0
MSEBIM_ALE3 Parallel mode only	1'b1	1'b0	1'b0	1'b0	1'b0	1'b0
MSEBI(x)_CLE	1'b0	1'b0	1'b0	1'b0	1'b1	1'b0
MSEBI(x)_DLE	1'b0	1'b0	1'b0	1'b0	1'b0	1'b1
Assignment on ACD						
MSEBI(x)_ACD7	1'b0	MSEBI_A24	MSEBI_A16	MSEBI_A8	MSEBI_A0	MSEBI_D7
MSEBI(x)_ACD6	MSEBI_A31	MSEBI_A23	MSEBI_A15	MSEBI_A7	MSEBI_R/W_N	MSEBI_D6
MSEBI(x)_ACD5	MSEBI_A30	MSEBI_A22	MSEBI_A14	MSEBI_A6	MSEBI_DMA_N	MSEBI_D5
MSEBI(x)_ACD4	MSEBI_A29	MSEBI_A21	MSEBI_A13	MSEBI_A5	MSEBI_CSREG_N	MSEBI_D4
MSEBI(x)_ACD3	MSEBI_A28	MSEBI_A20	MSEBI_A12	MSEBI_A4	MSEBI_CS3_N	MSEBI_D3
MSEBI(x)_ACD2	MSEBI_A27	MSEBI_A19	MSEBI_A11	MSEBI_A3	MSEBI_CS2_N	MSEBI_D2
MSEBI(x)_ACD1	MSEBI_A26	MSEBI_A18	MSEBI_A10	MSEBI_A2	MSEBI_CS1_N	MSEBI_D1
MSEBI(x)_ACD0	MSEBI_A25	MSEBI_A17	MSEBI_A9	MSEBI_A1	MSEBI_CS0_N	MSEBI_D0

Table 10.9 MSEBI Mode8, Chip Selects Management

Extended Address Capability Available with 0 ALE phase			
4 Chip Selects 2 B	3 Chip Selects 4 B	2 Chip Selects 8 B	1 Chip Select 16 B
MSEBI_CS3_N	MSEBI_A1	MSEBI_A1	MSEBI_A1
MSEBI_CS2_N	MSEBI_CS2_N	MSEBI_A2	MSEBI_A2
MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_A3
MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N

Extended Address Capability with 1 ALE phase			
4 Chip Selects 512 B	3 Chip Selects 1 KB	2 Chip Selects 2 KB	1 Chip Select 4 KB
MSEBI_CS3_N	MSEBI_A9	MSEBI_A9	MSEBI_A9
MSEBI_CS2_N	MSEBI_CS2_N	MSEBI_A10	MSEBI_A10
MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_A11
MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N

Extended Address Capability Available with 2 ALE phases			
4 Chip Selects 128 KB	3 Chip Selects 256 KB	2 Chip Selects 512 KB	1 Chip Select 1 MB
MSEBI_CS3_N	MSEBI_A17	MSEBI_A17	MSEBI_A17
MSEBI_CS2_N	MSEBI_CS2_N	MSEBI_A18	MSEBI_A18
MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_A19
MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N

Extended Address Capability Available with 3 ALE phases			
4 Chip Selects 32 MB	3 Chip Selects 64 MB	2 Chip Selects 128 MB	1 Chip Select 256 MB
MSEBI_CS3_N	MSEBI_A25	MSEBI_A25	MSEBI_A25
MSEBI_CS2_N	MSEBI_CS2_N	MSEBI_A26	MSEBI_A26
MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_CS1_N	MSEBI_A27
MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N	MSEBI_CS0_N

ALE phase is optional to increase bandwidth and reduce latency for Each chip select CS[n]_N (n = 0..3).

For Master Mode

CPU (n = 0..3)

- MSEBI_A26 .. MSEBI_A0 are directly driven by CPU (128 MB for each CS[n]_N)
- MSEBI_A31 .. MSEBI_A27 are driven by following registers:
  - rMSEBIM_CONFIG_CS[n]_N

DMA (n = 0..1)

- MSEBI_A31 .. MSEBI_A0 are driven by following registers:
  - rMSEBIM_ADDRDMA_CURRENTREAD_CS[n]_N (n = 0..1)
  - rMSEBIM_ADDRDMA_CURRENTWRITE_CS[n]_N (n = 0..1)

10.2 Register Map

10.2.1 Register Map MSEBI Master from CPU

Table 10.10 Register Map MSEBI Master from CPU

Address	Register Symbol	Register Name
400C 0000h + 100h × n	rMSEBIM_CYCLESIZE_CS[n]_N (n = 0..3)	Chip Select CycleSize Register
400C 0004h + 100h × n	rMSEBIM_SETUPHOLD_CS[n]_N (n = 0..3)	Chip Select SetupHold Register
400C 0008h + 100h × n	rMSEBIM_TDMACR_CS[n]_N (n = 0..1)	DMA Transmit Control and Status Register
400C 000Ch + 100h × n	rMSEBIM_RDMACR_CS[n]_N (n = 0..1)	DMA Receive Control and Status Register
400C 0010h + 100h × n	rMSEBIM_ADDRDMA_READ_CS[n]_N (n = 0..1)	DMA Read Address Register
400C 0014h + 100h × n	rMSEBIM_ADDRDMA_CURRENTREAD_CS[n]_N (n = 0..1)	DMA Current Read Address Register
400C 0018h + 100h × n	rMSEBIM_ADDRDMA_WRITE_CS[n]_N (n = 0..1)	DMA Write Address Register
400C 001Ch + 100h × n	rMSEBIM_ADDRDMA_CURRENTWRITE_CS[n]_N (n = 0..1)	DMA Current Write Address Register
400C 0020h + 100h × n	rMSEBIM_DMATDLR_CS[n]_N (n = 0..1)	DMA Transmit Data Level Register
400C 0024h + 100h × n	rMSEBIM_DMARDLR_CS[n]_N (n = 0..1)	DMA Receive Data Level Register
400C 0060h + 100h × n	rMSEBIM_CONFIG_CS[n]_N (n = 0..3)	Chip Select Config Register
400C 0800h	rMSEBIM_CONFIG	Common Config Register
400C 0808h	rMSEBIM_CPU_FIFOREAD_FLUSH	Flush Receive FIFO Register

10.2.2 Register Map MSEBI Master from DMA

Table 10.11 Register Map MSEBI Master from DMA

Address	Register Symbol	Register Name
4008 0000h + 20000h × n	rMSEBIM_DMA_FIFOREAD_CS[n]_N (n = 0..1)	DMA Receive FIFO (32 KB)
4009 0000h + 20000h × n	rMSEBIM_DMA_FIFOWRITE_CS[n]_N (n = 0..1)	DMA Transmit FIFO (64 KB)

10.2.3 Register Map MSEBI Slave from CPU

Table 10.12 Register Map MSEBI Slave from CPU

Address	Register Symbol	Register Name
400C 2000h + 100h × n	rMSEBIS_CYCLESIZE_CS[n]_N (n = 0..3)	Chip Select CycleSize Register
400C 2004h + 100h × n	rMSEBIS_SETUPHOLD_CS[n]_N (n = 0..3)	Chip Select SetupHold Register
400C 2008h + 100h × n	rMSEBIS_MMU_ADDR_CS[n]_N (n = 0..3)	MMU Base Address Register
400C 200Ch + 100h × n	rMSEBIS_MMU_ADDR_MASK_CS[n]_N (n = 0..3)	MMU Address Mask Register
400C 2010h + 100h × n	rMSEBIS_DMATX_REQ_CS[n]_N (n = 0..1)	DMA Transmit Request Register
400C 2014h + 100h × n	rMSEBIS_DMARX_REQ_CS[n]_N (n = 0..1)	DMA Receive Request Register
400C 2018h + 100h × n	rMSEBIS_DMATDLR_CS[n]_N (n = 0..1)	DMA Transmit Data Level Register
400C 201Ch + 100h × n	rMSEBIS_DMARDLR_CS[n]_N (n = 0..1)	DMA Receive Data Level Register
400C 2060h + 100h × n	rMSEBIS_CONFIG_CS[n]_N (n = 0..3)	Chip Select Config Register
400C 2800h	rMSEBIS_CONFIG	Common Config Register
400C 2804h	rMSEBIS_STATUS_INT0	Interrupt Status Register
400C 2808h	rMSEBIS_STATUS_INT1	Masked Interrupt Status Register
400C 280Ch	rMSEBIS_MASK_INT	Interrupt Mask Register
400C 2810h	rMSEBIS_CLR_INT	Interrupt Clear Register
400C 2814h	rMSEBIS_EOB_ADDR	End Of Block Address Register

10.2.4 Register Map MSEBI Slave from MSEBI

Table 10.13 Register Map MSEBI Slave from MSEBI

CS[n]*1	Address	Register Symbol	Register Name
CS0	400C 1000h	rMSEBIS_INT	Slave Interrupt Register
	400C 1004h	rMSEBIS_STATUS	Slave Status Register
	400C 1008h + 4h × n	rMSEBIS_ID_CS[n]_N (n: 0..3)	Slave ID Register
CS1	400C 1400h	rMSEBIS_INT	Slave Interrupt Register
	400C 1404h	rMSEBIS_STATUS	Slave Status Register
	400C 1408h + 4h × n	rMSEBIS_ID_CS[n]_N (n: 0..3)	Slave ID Register
CS2	400C 1800h	rMSEBIS_INT	Slave Interrupt Register
	400C 1804h	rMSEBIS_STATUS	Slave Status Register
	400C 1808h + 4h × n	rMSEBIS_ID_CS[n]_N (n: 0..3)	Slave ID Register
CS3	400C 1C00h	rMSEBIS_INT	Slave Interrupt Register
	400C 1C04h	rMSEBIS_STATUS	Slave Status Register
	400C 1C08h + 4h × n	rMSEBIS_ID_CS[n]_N (n: 0..3)	Slave ID Register

Note 1. MSEBI Master can access the register of each MSEBI Slave using MSEBI_CS[n]_N (n = 0..3) according to this table.

10.3 Register Description

10.3.1 Register Description MSEBI Master from CPU

10.3.1.1 rMSEBIM_CYCLESIZE_CS[n]_N — Chip Select CycleSize Register (n = 0..3)

CAUTION

Before switching configuration, the user must ensure that no previous accesses are pending by flushing the CPU receive FIFO (CPU write at rMSEBIM_CPU_FIFOREAD_FLUSH with the pattern 32'h0808). All FIFOs must be empty, and DMA controller must be stopped.

Address: 400C 0000h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_WRDLEDATA_NB								bMSEBIM_RDDLEDATA_NB							
Value after reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIM_WRDLEDATA_B		—	—	bMSEBIM_RDDLEDATA_B		—	—	—	—	—	—	bMSEBIM_CLEDATA	bMSEBIM_ALEDATA
Value after reset	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	1

Table 10.14 rMSEBIM_CYCLESIZE_CS[n]_N Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b24	bMSEBIM_WRDLEDATA_NB	Size of latch data phase in no burst mode (WRDLEDATA_NB) Used only: • On write cycle • Single access or first access of burst cycle (synchronous mode only). Time duration (MSEBIM_CLK) of MSEBIM_DLE high 8'h00: 1 MSEBIM_CLK 8'h01: 2 MSEBIM_CLK … … … 8'hFE: 255 MSEBIM_CLK 8'hFF: 256 MSEBIM_CLK See **Section 10.4.4, MSEBI Timing**.	R/W
b23 to b16	bMSEBIM_RDDLEDATA_NB	Size of latch data phase in no burst mode (RDDLEDATA_NB) Used only: • On read cycle • Single access or first access of burst cycle (synchronous mode only). Time duration (MSEBIM_CLK) of MSEBIM_DLE high 8'h00: 1 MSEBIM_CLK 8'h01: 2 MSEBIM_CLK … … … 8'hFE: 255 MSEBIM_CLK 8'hFF: 256 MSEBIM_CLK	R/W
b15, b14	Reserved	Read as 0.	R

Table 10.14 rMSEBIM_CYCLESIZE_CS[n]_N Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b13, b12	bMSEBIM_WRDLEDATA_B	Size of latch data phase in burst mode (WRDLEDATA_B) Used only: • On write cycle • Synchronous mode only with burst enable Time duration (MSEBIM_CLK) of MSEBIM_DLE high 2'b00: 1 MSEBIM_CLK 2'b01: 2 MSEBIM_CLK 2'b10: 3 MSEBIM_CLK 2'b11: 4 MSEBIM_CLK See **Section 10.4.4, MSEBI Timing**.	R/W
b11, b10	Reserved	Read as 0.	R
b9, b8	bMSEBIM_RDDLEDATA_B	Size of latch data phase in burst mode (RDDLEDATA_B) Use only: • On read cycle • Synchronous mode only with burst enable Time duration (MSEBIM_CLK) of MSEBIM_DLE high 2'b00: 1 MSEBIM_CLK 2'b01: 2 MSEBIM_CLK 2'b10: 3 MSEBIM_CLK 2'b11: 4 MSEBIM_CLK	R/W
b7 to b2	Reserved	Read as 0.	R
b1	bMSEBIM_CLEDATA	Size of control latch phase (CLEDATA) Time duration (MSEBIM_CLK) of MSEBIM_CLE phase: 0: 1 MSEBIM_CLK (high during 1 MSEBIM_CLK) 1: 2 MSEBIM_CLK (high during 1 MSEBIM_CLK then low during 1 MSEBIM_CLK) ***Caution*)** On asynchronous mode, use an MSEBI_CLE phase length of 2 to guarantee hold time of external latch.	R/W
b0	bMSEBIM_ALEDATA	Size of address latch phase (ALEDATA) Time duration (MSEBIM_CLK) of MSEBIM_ALE phase: 0: 1 MSEBIM_CLK (high during 1 MSEBIM_CLK) 1: 2 MSEBIM_CLK (high during 1 MSEBIM_CLK then low during 1 MSEBIM_CLK) ***Caution*)** If no ALE phase, this parameter is not used. On asynchronous mode, use an MSEBI_ALE phase length of 2 to guarantee hold time of external latch. See **Section 10.4.4, MSEBI Timing**.	R/W

10.3.1.2 rMSEBIM_SETUPHOLD_CS[n]_N — Chip Select SetupHold Register (n = 0..3)

CAUTION

Before switching configuration, the user must ensure that no previous accesses are pending by flushing the CPU receive FIFO (CPU write at rMSEBIM_CPU_FIFOREAD_FLUSH with the pattern 32'h0808). All FIFOs must be empty, and DMA controller must be stopped.

Address: 400C 0004h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	bMSEBIM_WRDLEHOLD						—	—	bMSEBIM_RDDLEHOLD					
Value after reset	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIM_WRDLESETUP						—	—	bMSEBIM_RDDLESETUP					
Value after reset	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1

Table 10.15 rMSEBIM_SETUPHOLD_CS[n]_N Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31, b30	Reserved	Read as 0.	R
b29 to b24	bMSEBIM_WRDLEHOLD	Size of hold data phase (WRDLEHOLD) Used only: • On write cycle Time duration (MSEBIM_CLK) of hold phase 6'h00: 0 MSEBIM_CLK 6'h01: 1 MSEBIM_CLK … … … 6'h3E: 62 MSEBIM_CLK 6'h3F: 63 MSEBIM_CLK See **Section 10.4.4, MSEBI Timing**.	R/W
b23, b22	Reserved	Read as 0.	R
b21 to b16	bMSEBIM_RDDLEHOLD	Size of hold data phase (RDDLEHOLD) Used only: • On read cycle Time duration (MSEBIM_CLK) of hold phase 6'h00: 0 MSEBIM_CLK 6'h01: 1 MSEBIM_CLK … … … 6'h3E: 62 MSEBIM_CLK 6'h3F: 63 MSEBIM_CLK ***Caution*)** Length "0" (to avoid bus conflict) should not be used for read access to peripheral.	R/W
b15, b14	Reserved	Read as 0.	R

Table 10.15 rMSEBIM_SETUPHOLD_CS[n]_N Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b13 to b8	bMSEBIM_WRDLESETUP	Size of setup data phase (WRDLESETUP) Used only: • On write cycle Time duration (MSEBIM_CLK) of setup phase 6'h00: 0 MSEBIM_CLK 6'h01: 1 MSEBIM_CLK ... ... 6'h3E: 62 MSEBIM_CLK 6'h3F: 63 MSEBIM_CLK	R/W
b7, b6	Reserved	Read as 0.	R
b5 to b0	bMSEBIM_RDDLESETUP	Size of setup data phase (RDDLESETUP) Used only: • On read cycle Time duration (MSEBIM_CLK) of setup phase 6'h00: 0 MSEBIM_CLK 6'h01: 1 MSEBIM_CLK ... ... 6'h3E: 62 MSEBIM_CLK 6'h3F: 63 MSEBIM_CLK ***Caution*)** Length "0" (to avoid bus conflict) should not be used for read access to peripheral. See **Section 10.4.4, MSEBI Timing**.	R/W

10.3.1.3 rMSEBIM_TDMACR_CS[n]_N — DMA Transmit Control and Status Register (n = 0..1)

Address: 400C 0008h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	bMSEBIM_SINGLE_DEST_WIDTH	bMSEBIM_CURRENT_DEST_BLOCK_SIZE													bMSEBIM_DEST_BLOCK_SIZE
Value after reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_DEST_BLOCK_SIZE												bMSEBIM_DEST_BURST_SIZE			bMSEBIM_TDMAE1
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.16 rMSEBIM_TDMACR_CS[n]_N Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31	Reserved	Read as 0.	R
b30	bMSEBIM_SINGLE_DEST_WIDTH	Size of single transaction DMAC.CTL[ch].DST_TR_WIDTH Register (ch = 0..7) must be programmed according to this value. 1'b1: Width is 64 bits ***Caution*)** All other values (8, 16, 32 bits for example) programmed in DMAC.CTL[ch].DST_TR_WIDTH fields are forbidden, to use the MSEBI interface	R
b29 to b17	bMSEBIM_CURRENT_DEST_BLOCK_SIZE	Current value of DEST_BLOCK_SIZE Once the transfer starts, the read of bMSEBIM_CURRENT_DEST_BLOCK_SIZE bits gives the total number of single transactions to be written in the DMA Transmit FIFO in order to end the current block transfer. bMSEBIM_CURRENT_DEST_BLOCK_SIZE is reloaded with bMSEBIM_DEST_BLOCK_SIZE value, when the firmware: Set "1'b1" on bMSEBIM_TDMAE1 (rising edge)	R
b16 to b4	bMSEBIM_DEST_BLOCK_SIZE	DEST_BLOCK_SIZE Destination Block Transfer Size in DMA Transmit FIFO. MSEBI is the flow controller. The user must write this field before or at the same time the DMA mode is enabled. The number programmed into DEST_BLOCK_SIZE indicates the total number of single transactions to perform for every block transfer. The size of single transaction depends on bMSEBIM_SINGLE_DEST_WIDTH bit. 0: 0 single transaction to transfer or end of block transfer 1: 1 single transaction to transfer 2: 2 single transactions to transfer … … … 8191: 8191 single transactions to transfer ***Note*)** To be able to transfer a max number of element (max DEST_BLOCK_SIZE = 8191 single elements), the destination pointer of the DMAC register DMAC.DAR[ch] (ch = 0..7) must be set to the base address of the FIFO before the beginning of the transfer.	R/W

Table 10.16 rMSEBIM_TDMACR_CS[n]_N Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3 to b1	bMSEBIM_DEST_BURST_SIZE	DEST_BURST_SIZE Destination Burst Transaction Size in DMA Transmit FIFO. The MSEBI is the flow controller. The user must write this field before or at the same time the DMA mode is enabled. Number of single transactions, to be written to the DMA Transmit FIFO every time a transmit burst transaction request is made 3'b000: 1 single transaction 3'b001: 4 single transactions, recommended value 3'b010: 8 single transactions 3'b011: 16 single transactions 3'b100: 32 single transactions 3'b101: Reserved, not used 3'b11x: Reserved, not used	R/W
b0	bMSEBIM_TDMAE1	Transmit DMA Enables/Disables 0: Disable the DMA in Transmit mode 1: Enable the DMA in Transmit mode A rising edge on bMSEBIM_TDMAE1 flushes the DMA Transmit FIFO. The bMSEBIM_TDMAE1 bit is automatically cleared by hardware to disable the DMA in Transmit mode after the last transaction in Transmit FIFO has completed (DEST_BLOCK_SIZE single transactions are written in DMA Transmit FIFO) and also all data in DMA Transmit FIFO have been output (FIFO Empty) to MSEBI bus. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. ***Caution*)** • When the DMA is controlled by MSEBIM_DMA_WR0_N or MSEBIM_DMA_WR1_N external pin (depends on MSEBI_CS_N used). Prior to enable this bit, Software must enable the bMSEBIM_USE_EXT_WRDMA_REQ in the rMSEBIM_DMATDLR_CS0_N (or rMSEBIM_DMATDLR_CS1_N, depends on MSEBI_CS_N used) register to enable the DMA mode by external control pins. After that, the DMA mode is controlled with the bMSEBIM_TDMAE1 and MSEBIM_DMA_WR0_N or MSEBIM_DMA_WR1_N external pin (depends on MSEBI_CS_N used). See **Figure 10.53, MSEBI Master: External DMA Request**. • When the DMA is not controlled by MSEBIM_DMA_WR0_N or MSEBIM_DMA_WR1_N external pin. Prior to enable this bit, Software must disable the bMSEBIM_USE_EXT_WRDMA_REQ in the rMSEBIM_DMATDLR_CS0_N (or rMSEBIM_DMATDLR_CS1_N, depends on MSEBI_CS_N used) register. After that, the DMA mode is controlled only with the bMSEBIM_TDMAE1. • If this bit is cleared during a DMA transfer, the current transfer (Burst or Single) is finished before the stop of DMA mode. The bMSEBIM_CURRENT_DEST_BLOCK_SIZE value will be consistent only when the current transfer is finished. To complete the DMA Block transfer: – Check the DMA Transmit FIFO is empty (bMSEBIM_DMA_TRANSMIT_FIFOLEVEL) – Reload the bMSEBIM_ADDRDMA_WRITE with the bMSEBIM_ADDRDMA_CURRENTWRITE – Reload the bMSEBIM_DEST_BLOCK_SIZE with the bMSEBIM_CURRENT_DEST_BLOCK_SIZE. – Update bMSEBIM_DEST_BURST_SIZE if necessary. – Set bMSEBIM_TDMAE1 to 1.	R/W

10.3.1.4 rMSEBIM_RDMACR_CS[n]_N — DMA Receive Control and Status Register (n = 0..1)

Address: 400C 000Ch + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	bMSEBIM_SINGLE_SRC_WIDTH	bMSEBIM_CURRENT_SRC_BLOCK_SIZE													bMSEBIM_SRC_BLOCK_SIZE
Value after reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_SRC_BLOCK_SIZE												bMSEBIM_SRC_BURST_SIZE			bMSEBIM_RDMAE1
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.17 rMSEBIM_RDMACR_CS[n]_N Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31	Reserved	Read as 0.	R
b30	bMSEBIM_SINGLE_SRC_WIDTH	Size of single transaction DMAC.CTL[ch].SRC_TR_WIDTH Register (ch = 0..7) must be programmed according to this value. 1'b1: Width is 64 bits ***Caution*)** All other values (8, 16, 32 bits for example) programmed in DMAC.CTL[ch].SRC_TR_WIDTH fields are forbidden, to use the MSEBI interface.	R
b29 to b17	bMSEBIM_CURRENT_SRC_BLOCK_SIZE	Current value of SRC_BLOCK_SIZE Once the transfer starts, the read of bMSEBIM_CURRENT_SRC_BLOCK_SIZE bits gives the total number of single transactions to be read in the DMA Receive FIFO in order to end the current block transfer. bMSEBIM_CURRENT_SRC_BLOCK_SIZE is reloaded with bMSEBIM_SRC_BLOCK_SIZE value, when the firmware: Set "1" on bMSEBIM_RDMAE1 (rising edge)	R
b16 to b4	bMSEBIM_SRC_BLOCK_SIZE	SRC_BLOCK_SIZE Source Block Transfer Size in DMA Receive FIFO. MSEBI is the flow controller. The user must write this field before or at the same time the DMA mode is enabled. The number programmed into SRC_BLOCK_SIZE indicates the total number of single transactions to perform for every block transfer. The size of single transaction depends on bMSEBIM_SINGLE_SRC_WIDTH bit. 0: 0 single transaction to transfer or end of block transfer 1: 1 single transaction to transfer 2: 2 single transactions to transfer … … … 8191: 8191 single transactions to transfer ***Note*)** To be able to transfer a maximum number of element (max DEST_BLOCK_SIZE = 8191 single elements), the source pointer of the DMAC register DMAC.SAR[ch] (ch = 0..7) must be set to the base address of the FIFO before the beginning of the transfer.	R/W

Table 10.17 rMSEBIM_RDMACR_CS[n]_N Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3 to b1	bMSEBIM_SRC_BURST_SIZE	SRC_BURST_SIZE Source Burst Transaction Size in DMA Receive FIFO. The MSEBI is the flow controller. The user must write this field before or at the same time the DMA mode is enabled. Number of single transactions, to be read in the DMA Receive FIFO every time a receive burst transaction request is made with n = 0 or n = 2 for CS0_N or CS1_N. 3'b000: 1 single transaction 3'b001: 4 single transactions, recommended value 3'b010: 8 single transactions 3'b011: 16 single transactions 3'b100: 32 single transactions 3'b101: Reserved, not used 3'b11x: Reserved, not used	R/W
b0	bMSEBIM_RDMAE1	Receive DMA Enables/Disables 0: Disable the DMA in Receive mode 1: Enable the DMA in Receive mode A rising edge on bMSEBIM_RDMAE1 flushes the DMA Receive FIFO. The bMSEBIM_RDMAE1 bit is automatically cleared by hardware to disable the DMA in Receive mode after the last transaction from DMA Receive FIFO has completed (SRC_BLOCK_SIZE single transactions read in DMA Receive FIFO). The DMA controller is stopped and DMA Receive FIFO is flushed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. ***Caution*)** • When the DMA is controlled by MSEBIM_DMA_RD0_N or MSEBIM_DMA_RD1_N external pin (depends on MSEBI_CS_N used). Prior to enable this bit, Software must enable the bMSEBIM_USE_EXT_RDDMA_REQ in the rMSEBIM_DMARDLR_CS0_N (or rMSEBIM_DMARDLR_CS1_N, depends on MSEBI_CS_N used) register to enable the DMA mode by external control pins. After that, the DMA mode is controlled with the bMSEBIM_RDMAE1 and MSEBIM_DMA_RD0_N or MSEBIM_DMA_RD1_N external pin (depends on MSEBI_CS_N used). See **Figure 10.53, MSEBI Master: External DMA Request**. • When the DMA is not controlled by MSEBIM_DMA_RD0_N or MSEBIM_DMA_RD1_N external pin. Prior to enable this bit, Software must disable the bMSEBIM_USE_EXT_RDDMA_REQ in the rMSEBIM_DMARDLR_CS0_N (or rMSEBIM_DMARDLR_CS1_N, depends on MSEBI_CS_N used) register. After that, the DMA mode is controlled only with the bMSEBIM_RDMAE1. • If this bit is cleared during a DMA transfer, the current transfer (Burst or Single) is finished before the stop of DMA mode. The bMSEBIM_CURRENT_SRC_BLOCK_SIZE value will be consistent only when the current transfer is finished. To complete the DMA Block transfer: – Check the DMA Receive FIFO is empty (bMSEBIM_DMA_RECEIVE_FIFOLEVEL) – Reload the bMSEBIM_ADDRDMA_READ with the bMSEBIM_ADDRDMA_CURRENTREAD – Reload the bMSEBIM_SRC_BLOCK_SIZE with the bMSEBIM_CURRENT_SRC_BLOCK_SIZE – Update bMSEBIM_SRC_BURST_SIZE if necessary. – Set bMSEBIM_RDMAE1 to 1.	R/W

10.3.1.5 rMSEBIM_ADDRDMA_READ_CS[n]_N — DMA Read Address Register (n = 0..1)

Address: 400C 0010h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_ADDRDMA_READ															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_ADDRDMA_READ													bMSEBIM_ADDRDMA_READ_2	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.18 rMSEBIM_ADDRDMA_READ_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b3	bMSEBIM_ADDRDMA_READ	Address DMA Read Access First block address used by DMA controller to start a DMA transfer from MSEBI bus to DMA Receive FIFO when the firmware set “1” on bMSEBIM_RDMAE1 (rising edge). See **Figure 10.49, MSEBI: Burst Mode, DMA Receive FIFO and Bus Interface Coupling**.	R/W
b2	bMSEBIM_ADDRDMA_READ_2	This bit is ignored to define first block address.	R/W
b1, b0	Reserved	These bits [1:0] are reset to 0.	R

10.3.1.6 rMSEBIM_ADDRDMA_CURRENTREAD_CS[n]_N — DMA Current Read Address Register (n = 0..1)

Address: 400C 0014h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_ADDRDMA_CURRENTREAD															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_ADDRDMA_CURRENTREAD															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.19 rMSEBIM_ADDRDMA_CURRENTREAD_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bMSEBIM_ADDRDMA_CURRENTREAD	Current Address DMA Read Access Address used by DMA controller to read data from MSEBI bus to DMA Receive FIFO. At the end of MSEBI access, this value is updated with the size of packed data reading on the bus. See **Figure 10.49, MSEBI: Burst Mode, DMA Receive FIFO and Bus Interface Coupling**. bMSEBIM_ADDRDMA_CURRENTREAD is reloaded with bMSEBIM_ADDRDMA_READ value, when the firmware set “1” on bMSEBIM_RDMAE1 (rising edge). On reload: • The bit [2:0] are reset to 0 if bMSEBIM_SINGLE_SRC_WIDTH = 1 (alignment 64 bits).	R

10.3.1.7 rMSEBIM_ADDRDMA_WRITE_CS[n]_N — DMA Write Address Register (n = 0..1)

Address: 400C 0018h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_ADDRDMA_WRITE															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_ADDRDMA_WRITE													bMSEBIM_ADDRDMA_WRITE_2	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.20 rMSEBIM_ADDRDMA_WRITE_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b3	bMSEBIM_ADDRDMA_WRITE	Address DMA Write Access First block address used by DMA controller to start a DMA transfer from DMA Transmit FIFO to MSEBI bus when the firmware set “1” on bMSEBIM_TDMAE1 (rising edge). The bit [1:0] are reset to 0 (alignment 32 bits). See **Figure 10.48, MSEBI: Burst Mode, DMA Transmit FIFO and Bus Interface Coupling**.	R/W
b2	bMSEBIM_ADDRDMA_WRITE_2	Depending on bMSEBIM_SINGLE_DEST_WIDTH bit • bMSEBIM_SINGLE_DEST_WIDTH: 1 (single transaction 64bits). This bit is ignored to define first block address (alignment 64 bits).	R/W
b1, b0	Reserved	These bits [1:0] are reset to 0 (Alignment 32 bits).	R

10.3.1.8 rMSEBIM_ADDRDMA_CURRENTWRITE_CS[n]_N — DMA Current Write Address Register (n = 0..1)

Address: 400C 001Ch + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_ADDRDMA_CURRENTWRITE															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_ADDRDMA_CURRENTWRITE															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.21 rMSEBIM_ADDRDMA_CURRENTWRITE_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bMSEBIM_ADDRDMA_CURRENTWRITE	Current Address DMA Write Access Address used by DMA controller to write data from DMA Transmit FIFO to MSEBI bus. At the end of MSEBI access, this value is updated with the size of packed data written on the bus. See **Figure 10.48, MSEBI: Burst Mode, DMA Transmit FIFO and Bus Interface Coupling**. bMSEBIM_ADDRDMA_CURRENTWRITE is reload with bMSEBIM_ADDRDMA_WRITE value, when the firmware set “1” on bMSEBIM_TDMAE1 (rising edge). On reload: • The bit [2:0] are reset to 0 if bMSEBIM_SINGLE_DEST_WIDTH = 1 (alignment 64 bits).	R

10.3.1.9 rMSEBIM_DMATDLR_CS[n]_N — DMA Transmit Data Level Register (n = 0..1)

CAUTION

Before switching configuration, the user must ensure that no previous accesses are pending by flushing the CPU receive FIFO (CPU write at rMSEBIM_CPU_FIFOREAD_FLUSH with the pattern 32'h0808). All FIFOs must be empty, and DMA controller must be stopped.

Address: 400C 0020h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bMSEBIM_USE_EXT_WRDMA_REQ
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_BURST_SIZEMAX_DMAWRITE			—	bMSEBIM_DMA_TRANSMIT_FIFOLEVEL						—	bMSEBIM_DMATDLR				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.22 rMSEBIM_DMATDLR_CS[n]_N Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31	Reserved	Keep initial value.	R/W
b30 to b17	Reserved	Read as 0.	R
b16	bMSEBIM_USE_EXT_WRDMA_REQ	In transmit mode, Enable the control of DMA channel by MSEBIM_DMA_WR0_N or MSEBIM_DMA_WR1_N external pin (depends on CS_N used). In this mode, the DMA mode is also controlled with the bMSEBIM_TDMAE1. Transmit DMA mode Enables/Disables 1'b0: Disable DMA control by external pins. The DMA transfer starts immediately on the rising edge of bMSEBIM_TDMAE1 bit. 1'b1: Enable DMA control by MSEBIM_DMA_WR0_N or MSEBIM_DMA_WR1_N external pin (depends on CS_N used). The DMA transfer starts when an external DMA request becomes 0 after a rising edge of bMSEBIM_TDMAE1 bit is detected. See **Figure 10.53, MSEBI Master: External DMA Request**.	R/W
b15 to b13	bMSEBIM_BURST_SIZEMAX_DMAWRITE	For all CS[n]_N (n = 0..1) Burst Size Max Allowed on Write access from DMA Transmit FIFO to MSEBI bus. Used by Round Robin Priority arbiter. See **Figure 10.45, MSEBI: Round Robin Priority**. 3'b000: 1 word 3'b001: 2 words 3'b010: 4 words 3'b011: 8 words 3'b100: 16 words 3'b101: Not limited 3'b11x: Reserved A word is 32 bits width for Mode32 device, 16 bits width for Mode16 and 8 bits width for Mode8 device.	R/W
b12	Reserved	Read as 0.	R

Table 10.22 rMSEBIM_DMATDLR_CS[n]_N Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b11 to b6	bMSEBIM_DMA_TRANSMIT_FIFOLEVEL	DMA Transmit FIFO Level Contains the number of valid data entries in the DMA transmit FIFO. 6'd0: 0 data entry, DMA Transmit FIFO empty 6'd1: 1 data entry or an activity is present on handshaking and/or MSEBI bus 6'd2: 2 data entries ... ... 6'd32: 32 data entries, DMA Transmit FIFO full ***Note*)** One data entry: 1 word 64 bits	R
b5	Reserved	Read as 0.	R
b4 to b0	bMSEBIM_DMATDLR	DMA Transmit FIFO 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 request signal is generated when the number of valid data entries in the DMA Transmit FIFO is equal to or below this field value, and bMSEBIM_TDMAE1 = 1. Refer below for the field decode. 6'd0: DMA request is asserted when 0 data entries are present in the DMA Transmit FIFO 6'd1: DMA request is asserted when 1 or less data entries are present in the DMA Transmit FIFO ... ... 6'd31: DMA request is asserted, when 31 or less data entries are present in the DMA Transmit FIFO Recommended value for optimal operation (ch = 0..7): DMAC.CTL[ch].DST_TR_WIDTH = 3 (64 bits) DMAC.CTL[ch].DEST_MSIZE = 1 (4 single transactions) bMSEBIM_SINGLE_DEST_WIDTH = 1 (64 bits) bMSEBIM_DEST_BURST_SIZE = 1 (4 single transactions) bMSEBIM_DMATDLR = 28 See **Figure 10.48, MSEBI: Burst Mode, DMA Transmit FIFO and Bus Interface Coupling**.	R/W

10.3.1.10 rMSEBIM_DMARDLR_CS[n]_N — DMA Receive Data Level Register (n = 0..1)

CAUTION

Before switching configuration, the user must ensure that no previous accesses are pending by flushing the CPU receive FIFO (CPU write at rMSEBIM_CPU_FIFOREAD_FLUSH with the pattern 32'h0808). All FIFOs must be empty, and DMA controller must be stopped.

Address: 400C 0024h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bMSEBIM_USE_EXT_RDDMA_REQ
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_BURST_SIZEMAX_DMAREAD			—	bMSEBIM_DMA_RECEIVE_FIFOLEVEL						—	bMSEBIM_DMARDLR				
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.23 rMSEBIM_DMARDLR_CS[n]_N Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31, b30	Reserved	Keep initial value.	R/W
b29 to b17	Reserved	Read as 0.	R
b16	bMSEBIM_USE_EXT_RDDMA_REQ	In receive mode, Enable the control of DMA channel by MSEBIM_DMA_RD0_N or MSEBIM_DMA_RD1_N external pin (depends on CS_N used). In this mode, the DMA mode is also controlled with the bMSEBIM_RDMAE1. Receive DMA mode Enables/Disables 1'b0: Disable DMA control by external pins. The DMA transfer starts immediately on a rising edge of bMSEBIM_RDMAE1 bit. 1'b1: Enable DMA control by MSEBIM_DMA_RD0_N or MSEBIM_DMA_RD1_N external pin (depends on CS_N used). The DMA transfer starts when an external DMA request becomes 0 after a rising edge of bMSEBIM_RDMAE1 bit is detected. See **Figure 10.53, MSEBI Master: External DMA Request**.	R/W
b15 to b13	bMSEBIM_BURST_SIZEMAX_DMAREAD	For all CS[n]_N (n = 0..1) Burst Size Max Allowed on Read access from MSEBI bus to DMA Receive FIFO. Used by Round Robin Priority arbiter. See **Figure 10.45, MSEBI: Round Robin Priority**. 3'b000: 1 word 3'b001: 2 words 3'b010: 4 words 3'b011: 8 words 3'b100: 16 words 3'b101: Not limited 3'b11x: Reserved A word is 32 bits width for Mode32 device, 16 bits width for Mode16 and 8 bits width for Mode8 device.	R/W
b12	Reserved	Read as 0.	R

Table 10.23 rMSEBIM_DMARDLR_CS[n]_N Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b11 to b6	bMSEBIM_DMA_RECEIVE_FIFOLEVEL	DMA Receive FIFO Level Contains the number of valid data entries in the DMA Receive FIFO. 6'd0: 0 data entry, DMA Receive FIFO empty 6'd1: 1 data entry or an activity is present on handshaking and/or MSEBI bus 6'd2: 2 data entries ... ... 6'd32: 32 data entries, DMA Receive FIFO full With one data entry: 1 word 64 bits	R
b5	Reserved	Read as 0.	R
b4 to b0	bMSEBIM_DMARDLR	DMA Receive FIFO Data Level This bit field controls the level at which a DMA request is made by the receive logic. The watermark level = bMSEBIM_DMARDLR+1, that is, DMA request is generated when the number of valid data entries in the DMA receive FIFO is equal to or above this field value + 1, and bMSEBIM_RDMAE1 =1. Refer below for the field decode. 6'd0: DMA request is asserted, when 1 or more data entries are present in the receive FIFO 6'd1: DMA request is asserted when 2 or more data entries are present in the receive FIFO ... ... 6'd31: DMA request is asserted, when 32 or more data entries are present in the receive FIFO Recommended value for optimal operation (ch = 0..7): DMAC.CTL[ch].SRC_TR_WIDTH = 3 (64 bits) DMAC.CTL[ch].SRC_MSIZE = 1 (4 single transactions) bMSEBIM_SINGLE_SRC_WIDTH = 1 (64 bits) bMSEBIM_SRC_BURST_SIZE = 1 (4 single transactions) bMSEBIM_DMARDLR = 3 See **Figure 10.49, MSEBI: Burst Mode, DMA Receive FIFO and Bus Interface Coupling**.	R/W

10.3.1.11 rMSEBIM_CONFIG_CS[n]_N — Chip Select Config Register (n = 0..3)

CAUTION

Before switching configuration, the user must ensure that no previous accesses are pending by flushing the CPU receive FIFO (CPU write at rMSEBIM_CPU_FIFOREAD_FLUSH with the pattern 32'h0808). All FIFOs must be empty, and DMA controller must be stopped.

Address: 400C 0060h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_EXTEND_ADDR					—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_MULTI_DLE	bMSEBIM_CS[n]N_ROUTING_CS3_N	bMSEBIM_CS[n]N_ROUTING_CS2_N	bMSEBIM_CS0N_ROUTING_CS1_N	bMSEBIM_ALE_MODE	bMSEBIM_ALE_NUMBER			bMSEBIM_BURST_ENABLE	bMSEBIM_MODE_WAIT		—	—	bMSEBIM_CONFIG		
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.24 rMSEBIM_CONFIG_CS[n]_N Register Contents (1/4)

Bit Position	Bit Name	Function	R/W
b31 to b27	bMSEBIM_EXTEND_ADDR	Extend address capability from MSEBI_A27 until MSEBI_A31 depending on user use case by register. With following allocation bits: Bit31: MSEBI_A31 Bit30: MSEBI_A30 Bit29: MSEBI_A29 Bit28: MSEBI_A28 Bit27: MSEBI_A27 See **Section 10.1.3, Multiplexed Signal Interface**.	R/W
b26 to b16	Reserved	Read as 0.	R
b15	bMSEBIM_MULTI_DLE	Multi DLE mode is used to give a fast and low pin consumption way to configure FPGA. During a write burst access, MSEBI_DLE present a rising edge on each valid signal of data to allow external FPGA configuration method. 1'b0: Multi DLE mode is disabled. 1'b1: Multi DLE mode is enabled. ***Caution*)** In Multi DLE mode, only write access are supported. Wait signals are ignored. See **Section 10.4.6.2(3), Master Multi DLE**.	R/W

Table 10.24 rMSEBIM_CONFIG_CS[n]_N Register Contents (2/4)

Bit Position	Bit Name	Function	R/W
b14	bMSEBIM_CS[n]N_ROUTING_CS3_N	During an access on: MSEBI_CS0_N MSEBI_CS1_N MSEBI_CS2_N MSEBI_CS3_N can be used as address bit to extend address capability depending on user use case. The assignment of address bit depends on the number of ALE phases. These bits are not used and read as 0 for: rMSEBIM_CONFIG_CS3_N Bit is managed as follows: 1'b0: No address routing on this line 1'b1: Address routing enabled on this line See **Section 10.1.3, Multiplexed Signal Interface**.	R/W
b13	bMSEBIM_CS[n]N_ROUTING_CS2_N	During an access on: MSEBI_CS0_N MSEBI_CS1_N MSEBI_CS2_N can be used as address bit to extend address capability depending on user use case. The assignment of address bit depends on the number of ALE phases. These bits are not used and read as 0 for: rMSEBIM_CONFIG_CS2_N rMSEBIM_CONFIG_CS3_N Bit is managed as follows: 1'b0: No address routing on this line 1'b1: Address routing enabled on this line See **Section 10.1.3, Multiplexed Signal Interface**.	R/W
b12	bMSEBIM_CS0N_ROUTING_CS1_N	During an access on: MSEBI_CS0_N MSEBI_CS1_N can be used as address bit to extend address capability depending on user use case. The assignment of address bit depends on the number of ALE phases. These bits are not used and read as 0 for: rMSEBIM_CONFIG_CS1_N rMSEBIM_CONFIG_CS2_N rMSEBIM_CONFIG_CS3_N Bit is managed as follows: 1'b0: No address routing on this line 1'b1: Address routing enabled on this line See **Section 10.1.3, Multiplexed Signal Interface**.	R/W

Table 10.24 rMSEBIM_CONFIG_CS[n]_N Register Contents (3/4)

Bit Position	Bit Name	Function	R/W
b11	bMSEBIM_ALE_MODE	MSEBI_ALE is managed in serial or parallel mode • 1'b0: Serial mode – Only MSEBIM_ALE used for all ALE cycles – Serial mode is recommended for synchronous interface • 1'b1: Parallel mode – Parallel mode is recommended for asynchronous interface, allowing direct connection of external latches (74x16373 type), thus reducing the cost of discrete components on the board – Mode32 MSEBIM_ALE used for ALE cycle – Mode16 MSEBIM_ALE used for first ALE cycle MSEBIM_ALE1 used for second ALE cycle – Mode8 MSEBIM_ALE used for first ALE cycle MSEBIM_ALE1 used for second ALE cycle MSEBIM_ALE2 used for third ALE cycle MSEBIM_ALE3 used for fourth ALE cycle See **Section 10.1.3, Multiplexed Signal Interface**, **Section 10.4.4, MSEBI Timing**.	R/W
b10 to b8	bMSEBIM_ALE_NUMBER	Number of phase MSEBI_ALE used to address the peripheral 3'b000: 0 MSEBI_ALE used 3'b001: 1 MSEBI_ALE used 3'b010: 2 MSEBI_ALE used 3'b011: 3 MSEBI_ALE used 3'b100: 4 MSEBI_ALE used 3'b101: Reserved 3'b11x: Reserved See **Section 10.1.3, Multiplexed Signal Interface**, **Section 10.4.4, MSEBI Timing**.	R/W
b7	bMSEBIM_BURST_ENABLE	Enable the burst mode on read or write access. The burst mode is only available in synchronous mode. In asynchronous mode, this bit is ignored. When burst is enable, 1'b0: Burst disable, on single access. 1'b1: Burst enable, single and burst access. ***Caution*)** When the master is accessing the slave's shared registers (using MSEBI_CSREG_N), no prefetch allowed. See **Section 10.4.4, MSEBI Timing**.	R/W
b6, b5	bMSEBIM_MODE_WAIT	For each MSEBI_CS[n]_N (n = 0..3) MSEBI interface can be configured in 3 basics function: 2'b00: No wait management on MSEBIM_WAIT[n]_N pin. The dedicated MSEBIM_WAIT[n]_N external pins are not used. 2'b01: Wait management on MSEBIM_WAIT[n]_N pin. The dedicated MSEBIM_WAIT[n]_N external pins are monitored and managed. 2'b10: Wait management on MSEBIM_WAIT0_N pin. Only one common MSEBIM_WAIT0_N external pin is monitored and managed for selected MSEBI_CS[n]_N. 2'b11: Reserved. For each MSEBI_CS[n]_N, the pins MSEBIM_WAIT[n]_N are managed in synchronous or asynchronous mode, depends on configuration of bMSEBIM_CONFIG. The mode 2'b10 allows to use only one MSEBIM_WAIT0_N external pin for all MSEBI_CS[n]_N. In this case, we have a reduced number of external pins used. See **Section 10.4.4, MSEBI Timing**.	R/W

Table 10.24 rMSEBIM_CONFIG_CS[n]_N Register Contents (4/4)

Bit Position	Bit Name	Function	R/W
b4, b3	Reserved	Keep initial value.	R/W
b2 to b0	bMSEBIM_CONFIG	MSEBI interface can be configured in 6 basics function: 3'b000: Asynchronous, 16 bits, multiplexed, Mode16, No Burst 3'b001: Synchronous, 16 bits, multiplexed, Mode16, Burst available 3'b010: Asynchronous, 32 bits, multiplexed, Mode32, No Burst 3'b011: Synchronous, 32 bits, multiplexed, Mode32, Burst available 3'b100: Asynchronous, 8 bits, multiplexed, Mode8, No Burst 3'b101: Synchronous, 8 bits, multiplexed, Mode8, Burst available 3'b110: Reserved 3'b111: Reserved See **Section 10.1.3, Multiplexed Signal Interface**, **Section 10.4.4, MSEBI Timing**.	R/W

10.3.1.12 rMSEBIM_CONFIG — Common Config Register

CAUTION

Before switching configuration, the user must ensure that no previous accesses are pending by flushing the CPU receive FIFO (CPU write at rMSEBIM_CPU_FIFOREAD_FLUSH with the pattern 32'h0808). All FIFOs must be empty, and DMA controller must be stopped.

Address: 400C 0800h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	bMSEBIM_CPU_RECEIVE_FIFOLEVEL						bMSEBIM_CPU_TRANSMIT_FIFOLEVEL
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_CPU_TRANSMIT_FIFOLEVEL					bMSEBIM_BURST_SIZEMAX_CPUREAD			bMSEBIM_BURST_SIZEMAX_CPUWRITE			bMSEBIM_CLKENABLE	bMSEBIM_CLKH		bMSEBIM_CLKL	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Table 10.25 rMSEBIM_CONFIG Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31, b30	Reserved	Keep initial value.	R/W
b29 to b23	Reserved	Read as 0.	R
b22 to b17	bMSEBIM_CPU_RECEIVE_FIFOLEVEL	CPU receive FIFO Level Contains the number of valid data entries in the CPU receive FIFO. 6'd0: 0 Data entry, CPU receive FIFO empty 6'd1: 1 Data entry or an activity is present on MSEBI bus 6'd2: 2 Data entries ... ... 6'd31: 31 Data entries 6'd32: 32 Data entries, CPU receive FIFO full With one data entry: 1 word 32 bits	R
b16 to b11	bMSEBIM_CPU_TRANSMIT_FIFOLEVEL	CPU transmit FIFO Level Contains the number of valid data entries in the CPU transmit FIFO. 6'd0: 0 Data entry, CPU transmit FIFO empty 6'd1: 1 Data entry or an activity is present on MSEBI bus 6'd2: 2 Data entries ... ... 6'd31: 31 Data entries 6'd32: 32 Data entries, CPU transmit FIFO full With one data entry: 1 word 32 bits	R

Table 10.25 rMSEBIM_CONFIG Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b10 to b8	bMSEBIM_BURST_SIZEMAX_CPUREAD	Burst Size Max Allowed on Read access from MSEBI bus to CPU receive FIFO. Used by Round Robin Priority arbiter. See **Figure 10.45, MSEBI: Round Robin Priority**. 3'b000: 1 word. 3'b001: 2 words. 3'b010: 4 words. 3'b011: 8 words. 3'b100: 16 words. 3'b101: Not limited. 3'b11x: Reserved. A word is 32 bits width for Mode32, 16 bits width for Mode16 and 8bits width for Mode8 device. ***Note*)** This bit is also used to control the maximum number of words to be read during a prefetch operation on the CPU receive FIFO.	R/W
b7 to b5	bMSEBIM_BURST_SIZEMAX_CPUWRITE	Burst Size Max Allowed on Write access from CPU transmit FIFO to MSEBI bus. Used by Round Robin Priority arbiter. See **Figure 10.45, MSEBI: Round Robin Priority**. 3'b000: 1 word. 3'b001: 2 words. 3'b010: 4 words. 3'b011: 8 words. 3'b100: 16 words. 3'b101: Not limited. 3'b11x: Reserved. A word is 32 bits width for Mode32, 16 bits width for Mode16 and 8bits width for Mode8 device.	R/W
b4	bMSEBIM_CLKENABLE	Enable MSEBIM_CLK clock generation 1'b0: Disable clock generation, MSEBIM_CLK is reset to 0 1'b1: Enable clock generation. (IO multiplexing setting is required) No glitch is generated when clock generation is enabled/disabled.	R/W
b3, b2	bMSEBIM_CLKH	MSEBIM_CLK clock period configuration Time duration (MSEBIM_HCLK) of level high of MSEBIM_CLK 2'b00: 1 MSEBIM_HCLK 2'b01: 2 MSEBIM_HCLK 2'b10: 3 MSEBIM_HCLK 2'b11: 4 MSEBIM_HCLK	R/W
b1, b0	bMSEBIM_CLKL	MSEBIM_CLK clock period configuration Time duration (MSEBIM_HCLK) of level low of MSEBIM_CLK 2'b00: 1 MSEBIM_HCLK 2'b01: 2 MSEBIM_HCLK 2'b10: 3 MSEBIM_HCLK 2'b11: 4 MSEBIM_HCLK	R/W

10.3.1.13 rMSEBIM_CPU_FIFOREAD_FLUSH — Flush Receive FIFO Register

Address: 400C 0808h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_CPU_FIFOREAD_FLUSH															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_CPU_FIFOREAD_FLUSH															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.26 rMSEBIM_CPU_FIFOREAD_FLUSH Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bMSEBIM_CPU_FIFOREAD_FLUSH	A write by CPU with the pattern 32'h0808 flushes the CPU receive FIFO. No Action when the pattern written is different of 32'h0808. ***Note*)** Always read as 0. See **Section 10.4.6, MSEBI Master Mode**.	W

10.3.2 Register Description MSEBI Master from DMA

10.3.2.1 rMSEBIM_DMA_FIFOREAD_CS[n]_N — DMA Receive FIFO (64 KB) (n = 0..1)

CAUTION

No CPU access allowed, may produce CPU hangup. This memory region is intended for DMA Controller use only.

Address: 4008 0000h + 20000h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_DMA_FIFOREAD															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_DMA_FIFOREAD															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.27 rMSEBIM_DMA_FIFOREAD_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bMSEBIM_DMA_FIFOREAD	DMA Receive FIFO Reading this register gives the data at the top of the receive FIFO. Each consecutive read pops the Receive FIFO and gives next data that is currently at the top of the DMA Receive FIFO. See **Figure 10.49, MSEBI: Burst Mode, DMA Receive FIFO and Bus Interface Coupling**.	R

10.3.2.2 rMSEBIM_DMA_FIFOWRITE_CS[n]_N — DMA Transmit FIFO (64 KB) (n = 0..1)

CAUTION

No CPU access allowed, may produce CPU hangup. This memory region is intended for DMA Controller use only.

Address: 4009 0000h + 20000h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIM_DMA_FIFOWRITE															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIM_DMA_FIFOWRITE															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.28 rMSEBIM_DMA_FIFOWRITE_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bMSEBIM_DMA_FIFOWRITE	DMA Transmit FIFO Writing this register pushes the data in the Transmit FIFO. Each consecutive write pushes the data to the next write location in the DMA Transmit FIFO. See **Figure 10.48, MSEBI: Burst Mode, DMA Transmit FIFO and Bus Interface Coupling**.	W

10.3.3 Register Description MSEBI Slave from CPU

10.3.3.1 rMSEBIS_CYCLESIZE_CS[n]_N — Chip Select CycleSize Register (n = 0..3)

CAUTION

Before switching configuration, user must ensure that no accesses are ongoing by reading bMSEBIS_BUSY = 0 (of the corresponding chip select).

Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE (of the corresponding chip select).

Address: 400C 2000h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIS_WRDLEDATA_NB								bMSEBIS_RDDLEDATA_NB							
Value after reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_WRDLEDATA_B		—	—	bMSEBIS_RDDLEDATA_B		—	—	—	—	—	—	bMSEBIS_CLEDATA	—
Value after reset	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	1

Table 10.29 rMSEBIS_CYCLESIZE_CS[n]_N Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b24	bMSEBIS_WRDLEDATA_NB	Size of latch data phase in no burst mode (WRDLEDATA_NB) Used only: • On write cycle • Single access or first access of burst cycle Time duration (MSEBIS_CLK) of MSEBIS_DLE high 8'h00: 1 MSEBIS_CLK 8'h01: 2 MSEBIS_CLK … … … 8'hFE: 255 MSEBIS_CLK 8'hFF: 256 MSEBIS_CLK See **Section 10.4.4, MSEBI Timing**.	R/W
b23 to b16	bMSEBIS_RDDLEDATA_NB	Size of latch data phase in no burst mode (RDDLEDATA_NB) Used only: • On read cycle • Single access or first access of burst cycle Time duration (MSEBIS_CLK) of MSEBIS_DLE high 8'h00: 1 MSEBIS_CLK 8'h01: 2 MSEBIS_CLK … … … 8'hFE: 255 MSEBIS_CLK 8'hFF: 256 MSEBIS_CLK	R/W
b15, b14	Reserved	Read as 0.	R

Table 10.29 rMSEBIS_CYCLESIZE_CS[n]_N Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b13, b12	bMSEBIS_WRDLEDATA_B	Size of latch data phase in burst mode (WRDLEDATA_B) Used only: • On write cycle • Burst enable Time duration (MSEBIS_CLK) of MSEBIS_DLE high 2'b00: 1 MSEBIS_CLK 2'b01: 2 MSEBIS_CLK 2'b10: 3 MSEBIS_CLK 2'b11: 4 MSEBIS_CLK See **Section 10.4.4, MSEBI Timing**.	R/W
b11, b10	Reserved	Read as 0.	R
b9, b8	bMSEBIS_RDDLEDATA_B	Size of latch data phase in burst mode (RDDLEDATA_B) Use only: • On read cycle • Burst enable Time duration (MSEBIS_CLK) of MSEBIS_DLE high 2'b00: 1 MSEBIS_CLK 2'b01: 2 MSEBIS_CLK 2'b10: 3 MSEBIS_CLK 2'b11: 4 MSEBIS_CLK	R/W
b7 to b2	Reserved	Read as 0.	R
b1	bMSEBIS_CLEDATA	Size of control latch phase (CLEDATA) Time duration (MSEBIS_CLK) of MSEBIS_CLE: 0: high during 1 MSEBIS_CLK 1: high during 1 MSEBIS_CLK then low during 1 MSEBIS_CLK	R/W
b0	Reserved	Keep initial value.	R/W

10.3.3.2 rMSEBIS_SETUPHOLD_CS[n]_N — Chip Select SetupHold Register (n = 0..3)

CAUTION

Before switching configuration, user must ensure that no accesses are ongoing by reading bMSEBIS_BUSY = 0 (of the corresponding chip select).

Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE (of the corresponding chip select).

Address: 400C 2004h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_WRDLESETUP						—	—	bMSEBIS_RDDLESETUP					
Value after reset	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1

Table 10.30 rMSEBIS_SETUPHOLD_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved	Read as 0.	R
b13 to b8	bMSEBIS_WRDLESETUP	Size of setup data phase (WRDLESETUP) Used only: • On write cycle Time duration (MSEBIS_CLK) of setup phase 6'h00: Reserved 6'h01: 1 MSEBIS_CLK … … … 6'h3E: 62 MSEBIS_CLK 6'h3F: 63 MSEBIS_CLK	R/W
b7, b6	Reserved	Read as 0.	R
b5 to b0	bMSEBIS_RDDLESETUP	Size of setup data phase (RDDLESETUP) Used only: • On read cycle Time duration (MSEBIS_CLK) of setup phase 6'h00: Reserved 6'h01: 1 MSEBIS_CLK … … … 6'h3E: 62 MSEBIS_CLK 6'h3F: 63 MSEBIS_CLK See **Section 10.4.4, MSEBI Timing**.	R/W

10.3.3.3 rMSEBIS_MMU_ADDR_CS[n]_N — MMU Base Address Register (n = 0..3)

CAUTION

Before switching configuration, user must ensure that no accesses are ongoing by reading bMSEBIS_BUSY = 0 (of the corresponding chip select).

Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE (of the corresponding chip select).

Address: 400C 2008h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIS_MMU_ADDR															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIS_MMU_ADDR				—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.31 rMSEBIS_MMU_ADDR_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b12	bMSEBIS_MMU_ADDR	MSEBI Slave controller uses this parameter for address translation when bMSEBIS_ADDR_MODE is set to MMU mode. In this configuration, bMSEBIS_MMU_ADDR is the base address for the conversion. See **Section 10.4.7.6, MSEBI Slave: Addressing Mode**.	R/W
b11 to b0	Reserved	Read as 0.	R

10.3.3.4 rMSEBIS_MMU_ADDR_MASK_CS[n]_N — MMU Address Mask Register (n = 0..3)

CAUTION

Before switching configuration, user must ensure that no accesses are ongoing by reading bMSEBIS_BUSY = 0 (of the corresponding chip select).

Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE (of the corresponding chip select).

Address: 400C 200Ch + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIS_MMU_ADDR_MASK															
Value after reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIS_MMU_ADDR_MASK				—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 10.32 rMSEBIS_MMU_ADDR_MASK_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b12	bMSEBIS_MMU_ADDR_MASK	MSEBI Slave controller uses this parameter for address translation when bMSEBIS_ADDR_MODE is set to MMU mode. In this configuration, bMSEBIS_MMU_ADDR is the mask of address used for the conversion. See **Section 10.4.7.6, MSEBI Slave: Addressing Mode**.	R/W
b11 to b0	Reserved	Read as 12'hFFF.	R

10.3.3.5 rMSEBIS_DMATX_REQ_CS[n]_N — DMA Transmit Request Register (n = 0..1)

Address: 400C 2010h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bMSEBIS_DMATX_ENABLE	bMSEBIS_DMATX_FORCE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 10.33 rMSEBIS_DMATX_REQ_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved	Read as 0.	R
b1	bMSEBIS_DMATX_ENABLE	For each MSEBI_CS[n]_N (n = 0..1) MSEBI Slave controller configuration status of DMA TX channel. This field must be set at the end of the configuration of the chip select. This bit is used to: • Set the chip select to active/inactive when MSEBI_CS[n]_N is coupled with MSEBI_DMA_N. If not empty, the DMA TX FIFO will be read until it becomes empty. See **Section 10.4.7.3, MSEBI Slave: Detection of Request Initiator**. Enable bit is managed as follows: 1'b0: DMA TX channel of MSEBI Slave controller is not ready to receive request from the master. If MSEBI_CS[n]_N and MSEBI_DMA_N are set to 0 during CLE phase, the current access is ignored. 1'b1: DMA TX channel of MSEBI Slave controller is ready to receive request from the Master. If MSEBI_CS[n]_N and MSEBI_DMA_N are set to 0 during CLE phase, the current access is executed. ***Caution*)** Chip select MSEBI_CS[n]_N must be configured to active state (Set dedicated bMSEBIS_CS_ENABLE bit to 1), if not (cleared to 0), the current access is ignored.	R/W
b0	bMSEBIS_DMATX_FORCE	For each MSEBI_CS[n]_N (n = 0..1) If DMA flow control is enabled (bMSEBIS_DMATX_FLOW_CTRL bit is set to 1), bMSEBIS_DMATX_FORCE drives DMA flow control pins: MSEBIS_DMA_WR[n]_N (n = 0..1) 1'b0: MSEBIS_DMA_WR[n]_N is set to 0. 1'b1: MSEBIS_DMA_WR[n]_N is set to 1.	R/W

10.3.3.6 rMSEBIS_DMARX_REQ_CS[n]_N — DMA Receive Request Register (n = 0..1)

Address: 400C 2014h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	bMSEBIS_DMARX_ENABLE	bMSEBIS_DMARX_FORCE
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 10.34 rMSEBIS_DMARX_REQ_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	Reserved	Read as 0.	R
b1	bMSEBIS_DMARX_ENABLE	For each MSEBI_CS[n]_N (n = 0..1) MSEBI Slave controller configuration status of DMA RX channel. This field must be set at the end of the configuration of the chip select. This bit is used to: • Flush corresponding FIFO • Set the chip select to active/inactive when MSEBI_CS[n]_N is coupled with MSEBI_DMA_N See **Section 10.4.7.3, MSEBI Slave: Detection of Request Initiator**. Enable bit is managed as follows: 1'b0: DMA RX channel of MSEBI Slave controller is not ready to receive request from the master. If MSEBI_CS[n]_N and MSEBI_DMA_N are set to 0 during CLE phase, the current access is ignored. 1'b1: DMA RX channel of MSEBI Slave controller is ready to receive request from the master. If MSEBI_CS[n]_N and MSEBI_DMA_N are set to 0 during CLE phase, the current access is executed. ***Caution*)** Chip select MSEBI_CS[n]_N must be configured to active state (Set dedicated bMSEBIS_CS_ENABLE bit to 1), if not (cleared to 0), the current access is ignored.	R/W
b0	bMSEBIS_DMARX_FORCE	For each MSEBI_CS[n]_N (n = 0..1) If DMA flow control is enabled (bMSEBIS_DMARX_FLOW_CTRL bit set to 1), bMSEBIS_DMARX_FORCE drives the DMA flow control signals: MSEBIS_DMA_RD[n]_N (n = 0..1). 1'b0: MSEBIS_DMA_RD[n]_N is set to 0. 1'b1: MSEBIS_DMA_RD[n]_N is set to 1.	R/W

10.3.3.7 rMSEBIS_DMATDLR_CS[n]_N — DMA Transmit Data Level Register (n = 0..1)

CAUTION

Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE (of the corresponding chip select).

Address: 400C 2018h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_DMATX_FIFO_LVL						—	—	—	—	bMSEBIS_DMATX_FLOW_CTRL	bMSEBIS_DMATX_OPT_BURST	bMSEBIS_DMATX_MAX_BURST	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.35 rMSEBIS_DMATDLR_CS[n]_N Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31	Reserved	Keep initial value	R/W
b30 to b14	Reserved	Read as 0.	R
b13 to b8	bMSEBIS_DMATX_FIFO_LVL	DMA TX[n] (n = 0..1) FIFO Level This FIFO contains the data to write on memory with AHB bus after a write request from MSEBI bus on MSEBI_CS[n]_N with MSEBI_DMA_N = DMA mode See **Table 10.47, Slave Detection of Request Initiator**. bMSEBIS_DMATX_FIFO_LVL contains the number of valid data entries in the DMA TX[n] (n = 0..1) FIFO. 6'd0: 0 Data entry, DMA TX[n] FIFO empty 6'd1: 1 Data entry or an activity is present on AHB bus 6'd2: 2 Data entries ... ... 6'd31: 31 Data entries 6'd32: 32 Data entries, DMA TX[n] FIFO full With one data entry: 1 word 32 bits See **Section 10.4.7.2(2)(a), Slave DMA FIFOs for Requests from Master MSEBI DMA TX FIFOs**.	R
b7 to b4	Reserved	Read as 0.	R
b3	bMSEBIS_DMATX_FLOW_CTRL	Enable the DMA flow control for signal MSEBIS_DMA_WR[n]_N (n = 0..1) See **Section 10.4.7.2(3), Slave DMA Flow Control Signals**. 1'b0: DMA flow control is disabled. 1'b1: DMA flow control is enabled.	R/W
b2	bMSEBIS_DMATX_OPT_BURST	MSEBI Slave controller waits to have enough data on the slave's DMA TX[n] (n = 0..1) FIFO to prepare a burst before sending write request to the NoC (until the reception of "end of block event"). This option optimizes bandwidth of the NoC AHB. Optimization on DMA TX[n] (n = 0..1) FIFO is enabled as follows: 1'b0: disable burst size optimization 1'b1: enable burst size optimization See **Section 10.4.7.2, MSEBI Slave: Burst Mode**, **Section 10.4.7.2(2)(a), Slave DMA FIFOs for Requests from Master MSEBI DMA TX FIFOs**.	R/W

Table 10.35 rMSEBIS_DMATDLR_CS[n]_N Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b1, b0	bMSEBIS_DMATX_MAX_BURST	MSEBI Slave controller can optimize access from slave's DMA TX[n] (n = 0..1) FIFO to AHB master port by grouping single write access into burst. The size maximum of a burst can be configured. Size of burst is control by following values: 2'b00: 1 word max 2'b01: 4 words max 2'b10: 8 words max 2'b11: 16 words max See **Section 10.4.7.2, MSEBI Slave: Burst Mode**, **Section 10.4.7.2(2)(a), Slave DMA FIFOs for Requests from Master MSEBI DMA TX FIFOs**.	R/W

10.3.3.8 rMSEBIS_DMARDLR_CS[n]_N — DMA Receive Data Level Register (n = 0..1)

CAUTION

Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE (of the corresponding chip select).

Address: 400C 201Ch + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_DMARX_FIFO_L						—	—	—	—	—	bMSEBIS_DMARX_FLOW_CTRL	bMSEBIS_DMARX_MAX_BURST	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.36 rMSEBIS_DMARDLR_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31	Reserved	Keep initial value	R/W
b30 to b14	Reserved	Read as 0.	R
b13 to b8	bMSEBIS_DMARX_FIFO_L	DMA RX[n] (n = 0..1) FIFO Level This FIFO contains the data read on AHB bus after a read request from MSEBI bus on MSEBI_CS[n]_N (n = 0..1) with MSEBI_DMA_N = DMA mode. bMSEBIS_DMARX_FIFO_LVL contains the number of valid data entries in the DMA RX[n] (n = 0..1) FIFO. 6'd0: 0 Data entry, DMA RX[n] FIFO empty 6'd1: 1 Data entry or an activity is present on MSEBI bus 6'd2: 2 Data entries ... ... 6'd31: 31 Data entries 6'd32: 32 Data entries, DMA RX[n] FIFO full With one data entry: 1 word 32 bits See **Table 10.47, Slave Detection of Request Initiator**, **Figure 10.62, Slave DMA FIFOs for Requests from Master MSEBI DMA RX FIFOs**.	R
b7 to b3	Reserved	Read as 0.	R
b2	bMSEBIS_DMARX_FLOW_CTRL	Enable the DMA flow control for signal MSEBIS_DMA_RD[n]_N (n = 0..1) 1'b0: DMA flow control is disabled. 1'b1: DMA flow control is enabled.	R/W
b1 to b0	bMSEBIS_DMARX_MAX_BURST	MSEBI Slave controller can optimize latency of read accesses coming from the DMA RX[n] (n = 0..1) FIFO of the master of the MSEBI bus with a prefetch mechanism on AHB master port. Size of burst prefetch can be configured: 2'b00: 1 word max 2'b01: 4 words max 2'b10: 8 words max 2'b11: 16 words max See **Section 10.4.7.2, MSEBI Slave: Burst Mode**, **Section 10.4.7.2(2)(b), Slave DMA FIFOs for Requests from Master MSEBI DMA RX FIFOs**.	R/W

10.3.3.9 rMSEBIS_CONFIG_CS[n]_N — Chip Select Config Register (n = 0..3)

CAUTION

Before switching configuration, user must ensure that no accesses are ongoing by reading bMSEBIS_BUSY = 0 (of the corresponding chip select).

Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE (of the corresponding chip select).

Address: 400C 2060h + 100h × n

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	bMSEBIS_CS[n]N_ROUTING_CS3_N	bMSEBIS_CS[n]N_ROUTING_CS2_N	bMSEBIS_CS[n]N_ROUTING_CS1_N	—	—	bMSEBIS_CS_ENABLE	bMSEBIS_ADDR_MODE	bMSEBIS_BURST_ENABLE	bMSEBIS_MODE_WAIT		—	bMSEBIS_WEN	bMSEBIS_BUSY	bMSEBIS_CONFIG	
Value after reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Table 10.37 rMSEBIS_CONFIG_CS[n]_N Register Contents (1/4)

Bit Position	Bit Name	Function	R/W
b31 to b15	Reserved	Read as 0.	R
b14	bMSEBIS_CS[n]N_ROUTING_CS3_N	During an access on: MSEBI_CS0_N MSEBI_CS1_N MSEBI_CS2_N MSEBI_CS3_N can be used as address bit to extend address capability depending on user use case. The assignment of address bit depends on the number of ALE phases. This bit is not used and read as 0: rMSEBIS_CONFIG_CS3_N Bit is managed as follows: 1'b0: No address routing on this line 1'b1: Address routing enabled on this line See **Section 10.1.3, Multiplexed Signal Interface**.	R/W
b13	bMSEBIS_CS[n]N_ROUTING_CS2_N	During an access on: MSEBI_CS0_N MSEBI_CS1_N MSEBI_CS2_N can be used as address bit to extend address capability depending on user use case. The assignment of address bit depends on the number of ALE phases. These bits are not used and read as 0 for: rMSEBIS_CONFIG_CS2_N rMSEBIS_CONFIG_CS3_N Bit is managed as follows: 1'b0: No address routing on this line 1'b1: Address routing enabled on this line See **Section 10.1.3, Multiplexed Signal Interface**.	R/W

Table 10.37 rMSEBIS_CONFIG_CS[n]_N Register Contents (2/4)

Bit Position	Bit Name	Function	R/W
b12	bMSEBIS_CS[n]N_ROUTING_CS1_N	During an access on: MSEBI_CS0_N MSEBI_CS1_N can be used as address bit to extend address capability depending on user use case. The assignment of address bit depends on the number of ALE phases. These bits are not used and read as 0 for: rMSEBIS_CONFIG_CS1_N rMSEBIS_CONFIG_CS2_N rMSEBIS_CONFIG_CS3_N Bit is managed as follows: 1'b0: No address routing on this line 1'b1: Address routing enabled on this line See **Section 10.1.3, Multiplexed Signal Interface**.	R/W
b11, b10	Reserved	Read as 0.	R
b9	bMSEBIS_CS_ENABLE	MSEBI Slave controller configuration status of channel CPU This field is set by the slave CPU when it completes the configuration on configuration registers group dedicated to MSEBI_CS[n]_N (n = 0..3). Enable bit is managed as follow: • 1'b0: CPU channel of MSEBI Slave controller is not ready to receive request from the Master – Set the chip select (MSEBI_CS[n]_N) to inactive state. – Controls the status value reported in rMSEBIS_ID_CS[n]_N register – Configuration registers group dedicated to MSEBI_CS[n]_N are unlocked (can be written by CPU) • 1'b1: CPU channel of MSEBI Slave controller is ready to receive request from the master – Set the chip select (MSEBI_CS[n]_N) to active state. – Controls the status value reported in rMSEBIS_ID_CS[n]_N register – Configuration registers group dedicated to MSEBI_CS[n]_N are locked (cannot be written by CPU). If detection of falling edge on bMSEBIS_CS_ENABLE, the dedicated bMSEBIS_ERROR_CS_CONFIGURATION bit is set inside rMSEBIS_ID_CS register. Lock/unlock access to configuration registers: • rMSEBIS_CYCLESIZE_CS[n]_N (n = 0..3) • rMSEBIS_SETUPHOLD_CS[n]_N (n = 0..3) • rMSEBIS_MMU_ADDR_CS[n]_N (n = 0..3) • rMSEBIS_MMU_MASK_ADDR_CS[n]_N (n = 0..3) • rMSEBIS_DMATDLR_CS[n]_N (n = 0..1) • rMSEBIS_DMARDLR_CS[n]_N (n = 0..1) • rMSEBIS_CONFIG_CS[n]_N (n = 0..3) • rMSEBIS_CONFIG • rMSEBIS_EOB_ADDR ***Note*)** If bMSEBIS_CS_ENABLE is cleared on MSEBI_CS[n]_N (n = 0..3). • The dedicated request is not taken into account on CLE and DLE phases decoding. • The dedicated request inside CPU and DMA FIFO is not executed. • If access on MSEBI_CS[n]_N is ongoing, we can lose data and synchronization between MSEBI master and slave. See **Section 10.4.7.8, MSEBI Slave: Configuration Registers & Synchronization**.	R/W

Table 10.37 rMSEBIS_CONFIG_CS[n]_N Register Contents (3/4)

Bit Position	Bit Name	Function	R/W
b8	bMSEBIS_ADDR_MODE	MSEBI Slave interface can be configured in 2 basics function: 1'b0: address management by direct access 1'b1: address management by MMU mode See **Section 10.4.7.6, MSEBI Slave: Addressing Mode**.	R/W
b7	bMSEBIS_BURST_ENABLE	For each MSEBI_CS[n]_N (n = 0..3), enable the burst mode on read or write access on AHB master port. The size of a burst is limited depending on the configuration parameters for max burst size on rMSEBIS_CONFIG. If burst is disable, no prefetch allowed for this chip select. 1'b0: Burst disable, only single access. 1'b1: Burst enable, single and burst access See **Section 10.4.7.2, MSEBI Slave: Burst Mode**, **Section 10.4.4, MSEBI Timing**. ***Caution*)** When the master is accessing the slave's shared registers (using MSEBI_CSREG_N), no prefetch allowed.	R/W
b6, b5	bMSEBIS_MODE_WAIT	For each MSEBI_CS[n]_N (n = 0..3), MSEBI Slave interface can be configured in 3 basic functions: 2'b00: Reserved 2'b01: Wait management on MSEBIS_WAIT[n]_N pin. The dedicated MSEBIS_WAIT[n]_N external pins are driven. 2'b10: Wait management on MSEBIS_WAIT0_N pin. Only one common MSEBIS_WAIT0_N external pin is driven for the selected MSEBI_CS[n]_N. 2'b11: Reserved The mode 2'b10 allows to use only one MSEBIS_WAIT0_N external pin for all MSEBI_CS[n]_N. In this case we reduce the number of external pins needed. See **Section 10.4.4, MSEBI Timing**.	R/W
b4	Reserved	Keep initial value	R/W
b3	bMSEBIS_WEN	This bit permits to manage write access right to the device. If a write access occurs when bMSEBIS_WEN is set to 0, an error is flag on rMSEBIS_STATUS. Rights access are set as follow: 1'b0: Write on device is disabled. 1'b1: Write on device is enabled. See **Section 10.4.7.7, MSEBI Slave: Write Protect**.	R/W

Table 10.37 rMSEBIS_CONFIG_CS[n]_N Register Contents (4/4)

Bit Position	Bit Name	Function	R/W
b2	bMSEBIS_BUSY	For each MSEBI_CS[n]_N (n = 0..3), gives MSEBI slave controller communication status. bMSEBIS_BUSY is set to 1 until all following conditions are met: • No MSEBI bus access is ongoing on chip select (MSEBI_CS[n]_N) in CPU or DMA mode • No AHB bus access is ongoing • DMA Mode: Read from AHB bus to DMA RX FIFO – If DMA disabled (bMSEBIS_DMARX_ENABLE bit is cleared), FIFO flushed and empty. – If DMA enabled (bMSEBIS_DMARX_ENABLE bit is set), DMA RX FIFO empty and no AHB access is ongoing. • DMA Mode: Write from DMA TX FIFO to AHB bus – DMA TX FIFO empty (if not wait “end of block event” required) and no AHB access is ongoing. • CPU Mode: Write from CPU receive FIFO to AHB bus – No request pending inside CPU receive FIFO – No AHB access ongoing • CPU Mode: From AHB Bus to CPU transmit FIFO – No request pending inside CPU transmit FIFO – No AHB access ongoing 1’b0: MSEBI slave controller is idle for this chip select (MSEBI_CS[n]_N) 1’b1: A request is still ongoing on this chip select (MSEBI_CS[n]_N) See figures: **Figure 10.62, Slave DMA FIFOs for Requests from Master MSEBI DMA RX FIFOs** **Figure 10.61, Slave DMA FIFOs for Requests from Master MSEBI DMA TX FIFOs** **Figure 10.58, MSEBI Slave CPU FIFOs Example 1** **Figure 10.59, MSEBI Slave CPU FIFOs Example 1 Mode8** **Figure 10.60, MSEBI Slave CPU FIFOs Example 2**	R
b1, b0	bMSEBIS_CONFIG	MSEBI Slave interface can be configured in 3 basics function: 2’b00: Synchronous, 16 bits, multiplexed, Mode16, Burst available 2’b01: Synchronous, 32 bits, multiplexed, Mode32, Burst available 2’b10: Synchronous, 8 bits, multiplexed, Mode8, Burst available 2’b11: Reserved See Sections: **Section 10.1.3, Multiplexed Signal Interface** **Section 10.4.4, MSEBI Timing**	R/W

10.3.3.10 rMSEBIS_CONFIG — Common Config Register

CAUTION

Before switching configuration, user must ensure that no accesses are ongoing by reading bMSEBIS_BUSY = 0 (of the corresponding chip select).

Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE of all the chip selects MSEBI_CS[n]_N. (Except bMSEBIS_TIMEOUT_REG_ACCESS that can be accessed at any time)

Address: 400C 2800h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	bMSEBIS_WAIT_CONF	—	bMSEBIS_TIMEOUT_REG_ACCESS	bMSEBIS_TIMEOUT_REG_ACCESS_DELAY				bMSEBIS_AHB_MASTER_CACHE	bMSEBIS_AHB_MASTER_BUF	bMSEBIS_CPUTX_FIFO_LVL					
Value after reset	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_CPURX_FIFO_LVL						—	—	—	—	bMSEBIS_BURST_SIZEMAX_CPUWRITE		bMSEBIS_BURST_SIZEMAX_CPUREAD	
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.38 rMSEBIS_CONFIG Register Contents (1/3)

Bit Position	Bit Name	Function	R/W
b31	Reserved	Keep initial value	R/W
b30	bMSEBIS_WAIT_CONF	Should be set to 1.	R/W
b29	Reserved	Keep initial value	R/W
b28	bMSEBIS_TIMEOUT_REG_ACCESS	Flag a timeout during the synchronization mechanism for write access on slave's configuration registers. For read access: 1'b0: no timeout detected 1'b1: timeout detected For write access (can be cleared at any time): 1'b0: write 0 has no effect 1'b1: write 1 clear the bit In order to be able to clear this bit, the synchronization mechanism must first exit the timeout condition, or the flag will stay at 1'b1. To exit this state, the user can: • Wait for the MSEBIS_CLK clock to start, so that the synchronization mechanism could end Properly • Stop the last chip select(s) started Then, the user will be able to clear the flag. See bit field: bMSEBIS_TIMEOUT_REG_ACCESS_DELAY in this register.	R/W

Table 10.38 rMSEBIS_CONFIG Register Contents (2/3)

Bit Position	Bit Name	Function	R/W
b27 to b24	bMSEBIS_TIMEOUT_REG_ACCESS_DELAY	bMSEBIS_TIMEOUT_REG_ACCESS_DELAY defines the delay (number of MSEBIS_HCLK clock cycles) before the timeout is triggered when unlocking configuration registers for chip select [n] (n = 0..3). Write accesses to configuration register are locked when the chip select [n] is set as enable: bMSEBIS_CS_ENABLE Before writing a configuration register, user must unlock registers by clearing bMSEBIS_CS_ENABLE. When the MSEBI slave controller receives the unlock command, AHB access on configuration registers are suspended until the internal synchronization mechanism completes. • Synchronization lasts 2 MSEBIS_CLK clock cycle + 2 MSEBIS_HCLK clock Cycle. When MSEBIS_CLK clock is cut by the master of the bus, internal synchronization mechanism cannot complete. To avoid dead lock on AHB, MSEBI slave controller uses a timeout based on MSEBIS_HCLK. Delay before timeout can be configured: 4'h0: timeout after 4 MSEBIS_HCLK clock cycle 4'h1: timeout after 8 MSEBIS_HCLK clock cycle ... ... 4'hE: timeout after 60 MSEBIS_HCLK clock cycle 4'hF: timeout after 64 MSEBIS_HCLK clock cycle After timeout: • AHB bus is released • Synchronization is still ongoing • Write access to configuration registers are locked until the end of the synchronization mechanism • Flag is set on bMSEBIS_TIMEOUT_REG_ACCESS See **Section 10.4.7.8, MSEBI Slave: Configuration Registers & Synchronization**.	R/W
b23	bMSEBIS_AHB_MASTER_CACHE	Request from AHB master port of the slave can be configured to allow use of the cache. This parameter drives the AHB signal: HPROT[3]. Configuration is as follow: 1'd0: Data is not cacheable 1'd1: Data is cacheable	R/W
b22	bMSEBIS_AHB_MASTER_BUF	Request from AHB master port of the slave can be configured to allow use of bufferable access. This parameter drives the AHB signal: HPROT[2]. Configuration is as follow: 1'd0: Data is not bufferable 1'd1: Data is bufferable	R/W
b21 to b16	bMSEBIS_CPUTX_FIFO_LVL	CPU transmit FIFO Level This FIFO contains the data read on AHB bus after a read request from MSEBI bus with MSEBI_DMA_N = CPU mode. See **Table 10.47, Slave Detection of Request Initiator**. bMSEBIS_CPUTX_FIFO_LVL contains the number of valid data entries in the CPU transmit FIFO. 6'd0: 0 Data entry, CPU transmit FIFO empty 6'd1: 1 Data entry or an activity is present on AHB bus 6'd2: 2 Data entries ... ... 6'd31: 31 Data entries 6'd32: 32 Data entries, CPU transmit FIFO full Size of one data entry can be 8/16/32bits depending on size MSEBI bus and type access (burst, no burst). See **Figure 10.58, MSEBI Slave CPU FIFOs Example 1**, **Figure 10.59, MSEBI Slave CPU FIFOs Example 1 Mode8**, **Figure 10.60, MSEBI Slave CPU FIFOs Example 2**.	R
b15, b14	Reserved	Read as 0.	R

Table 10.38 rMSEBIS_CONFIG Register Contents (3/3)

Bit Position	Bit Name	Function	R/W
b13 to b8	bMSEBIS_CPURX_FIFO_LVL	CPU receive FIFO Level This FIFO contains all incoming requests from MSEBI bus with MSEBI_DMA_N = CPU mode. See **Table 10.47, Slave Detection of Request Initiator**. bMSEBIS_CPURX_FIFO_LVL contains the number of valid data entries in the CPU receive FIFO. 6'd0: 0 Data entry, CPU receive FIFO empty 6'd1: 1 Data entry or an activity is present on MSEBI bus 6'd2: 2 Data entries ... ... 6'd31: 31 Data entries 6'd32: 32 Data entries, CPU receive FIFO full Size of one data entry can be 8/16/32bits depending on size MSEBI bus. See **Figure 10.58, MSEBI Slave CPU FIFOs Example 1**, **Figure 10.59, MSEBI Slave CPU FIFOs Example 1 Mode8**, **Figure 10.60, MSEBI Slave CPU FIFOs Example 2**.	R
b7 to b4	Reserved	Read as 0.	R
b3, b2	bMSEBIS_BURST_SIZEMAX_CPUWRITE	MSEBI Slave controller can optimize access from slave's CPU receive FIFO to AHB master port by grouping single write access into burst. The size maximum of a burst can be configured. Size of burst is control by following values: 2'b00: 1 word max 2'b01: 4 words max 2'b10: 8 words max 2'b11: 16 words max See **Section 10.4.7.2, MSEBI Slave: Burst Mode**, **Section 10.4.7.2(1), Slave CPU FIFOs**.	R/W
b1, b0	bMSEBIS_BURST_SIZEMAX_CPUREAD	MSEBI Slave controller can optimize latency of read accesses coming from the CPU transmit FIFO of the master of the MSEBI bus with a prefetch mechanism on AHB master port. Size of a prefetch burst can be configured: 2'b00: 1 word max 2'b01: 4 words max 2'b10: 8 words max 2'b11: 16 words max See **Section 10.4.7.2, MSEBI Slave: Burst Mode**, **Section 10.4.7.2(1), Slave CPU FIFOs**.	R/W

10.3.3.11 rMSEBIS_STATUS_INT0 — Interrupt Status Register

These bits refer to the “end of block event” interrupt on the slave part of the controller.

They contain the MSEBI Slave interrupt status before mask (rMSEBIS_MASK_INT register) is applied. Each bit is independently set when the master of the MSEBI bus write 1 on the corresponding bit of the register:

- rMSEBIS_INT

See **Section 10.4.5, MSEBI Interrupt**.

Address: 400C 2804h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_INT0_DMATX		—	—	bMSEBIS_INT0_DMARX		bMSEBIS_INT0_CPUTX				bMSEBIS_INT0_CPURX			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.39 rMSEBIS_STATUS_INT0 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved	Read as 0.	R
b13, b12	bMSEBIS_INT0_DMATX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1 1’b0: end of block not detected 1’b1: end of block detected	R
b11, b10	Reserved	Read as 0.	R
b9, b8	bMSEBIS_INT0_DMARX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1 1’b0: end of block not detected 1’b1: end of block detected	R
b7 to b4	bMSEBIS_INT0_CPUTX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3 1’b0: end of block not detected 1’b1: end of block detected	R
b3 to b0	bMSEBIS_INT0_CPURX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3 1’b0: end of block not detected 1’b1: end of block detected	R

10.3.3.12 rMSEBIS_STATUS_INT1 — Masked Interrupt Status Register

These bits refer to the “end of block event” interrupt on the slave part of the controller.

They contain the MSEBI Slave interrupt status after mask (rMSEBIS_MASK_INT register) is applied on the rMSEBIS_STATUS_INT0 register. Each bit is the result of a logical and between the corresponding bit of register:

- rMSEBIS_MASK_INT
- rMSEBIS_STATUS_INT0

See **Section 10.4.5, MSEBI Interrupt**.

Address: 400C 2808h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_INT1_DMATX		—	—	bMSEBIS_INT1_DMARX		bMSEBIS_INT1_CPUTX				bMSEBIS_INT1_CPURX			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.40 rMSEBIS_STATUS_INT1 Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved	Read as 0.	R
b13, b12	bMSEBIS_INT1_DMATX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1 1’b0: end of block interrupt not detected or masked 1’b1: end of block detected and not masked	R
b11, b10	Reserved	Read as 0.	R
b9, b8	bMSEBIS_INT1_DMARX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1 1’b0: end of block interrupt not detected or masked 1’b1: end of block detected and not masked	R
b7 to b4	bMSEBIS_INT1_CPUTX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3 1’b0: end of block interrupt not detected or masked 1’b1: end of block detected and not masked	R
b3 to b0	bMSEBIS_INT1_CPURX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3 1’b0: end of block interrupt not detected or masked 1’b1: end of block detected and not masked	R

10.3.3.13 rMSEBIS_MASK_INT — Interrupt Mask Register

This register contains the interruption mask dedicated for MSEBI Slave interrupt.

Each bit refers to a bit on the rMSEBIS_STATUS_INT0 register.

Address: 400C 280Ch

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_MASK_INT_DMATX		—	—	bMSEBIS_MASK_INT_DMARX		bMSEBIS_MASK_INT_CPUTX				bMSEBIS_MASK_INT_CPURX			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.41 rMSEBIS_MASK_INT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved	Read as 0.	R
b13, b12	bMSEBIS_MASK_INT_DMATX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1 1'b0: end of block interrupt mask 1'b1: end of block interrupt not mask	R/W
b11, b10	Reserved	Read as 0.	R
b9, b8	bMSEBIS_MASK_INT_DMARX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1 1'b0: end of block interrupt mask 1'b1: end of block interrupt not mask	R/W
b7 to b4	bMSEBIS_MASK_INT_CPUTX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3 1'b0: end of block interrupt mask 1'b1: end of block interrupt not mask	R/W
b3 to b0	bMSEBIS_MASK_INT_CPURX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3 1'b0: end of block interrupt mask 1'b1: end of block interrupt not mask	R/W

10.3.3.14 rMSEBIS_CLR_INT — Interrupt Clear Register

When CPU writes 1 on any bit of this field, it clears the corresponding interrupt source on rMSEBIS_STATUS_INT0.

NOTE

Clearing an interrupt with this register also clears the corresponding bit on rMSEBIS_INT.

Address: 400C 2810h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_CLR_INT_DMATX		—	—	bMSEBIS_CLR_INT_DMARX		bMSEBIS_CLR_INT_CPUTX				bMSEBIS_CLR_INT_CPURX			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.42 rMSEBIS_CLR_INT Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved	Read as 0.	R
b13, b12	bMSEBIS_CLR_INT_DMATX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1 Write 1 to clear the interrupt status bit.	R/W
b11, b10	Reserved	Read as 0.	R
b9, b8	bMSEBIS_CLR_INT_DMARX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1 Write 1 to clear the interrupt status bit.	R/W
b7 to b4	bMSEBIS_CLR_INT_CPUTX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3 Write 1 to clear the interrupt status bit.	R/W
b3 to b0	bMSEBIS_CLR_INT_CPURX	Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3 Write 1 to clear the interrupt status bit.	R/W

10.3.3.15 rMSEBIS_EOB_ADDR — End Of Block Address Register

NOTE

Addresses for the descriptors for CS[n]_N (n = 0..3) are automatically computed.
See **Table 10.43, rMSEBIS_EOB_ADDR Register Contents**.

CAUTION

Before switching configuration, user must ensure that no accesses are ongoing by reading bMSEBIS_BUSY = 0 (of the corresponding chip select). Any update of this register must be done after clearing the bMSEBIS_CS_ENABLE of all chip selects MSEBI_CS[n]_N.

Address: 400C 2814h

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIS_EOB_ADDR															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIS_EOB_ADDR														—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.43 rMSEBIS_EOB_ADDR Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b2	bMSEBIS_EOB_ADDR	Contains the address of the descriptor used to complete the write transfer in memory for MSEBI_CS[n]_N • CPU mode: n = 0..3 • DMA mode: n = 0..1 This address is also used to compute the addresses of descriptors as follows: rMSEBIS_EOB_ADDR for CPU: rMSEBIS_EOB_ADDR refers to MSEBI_CS0_N rMSEBIS_EOB_ADDR + 32'h04 refers to MSEBI_CS1_N rMSEBIS_EOB_ADDR + 32'h08 refers to MSEBI_CS2_N rMSEBIS_EOB_ADDR + 32'h0C refers to MSEBI_CS3_N rMSEBIS_EOB_ADDR for DMA: rMSEBIS_EOB_ADDR + 32'h10 refers to MSEBI_CS0_N rMSEBIS_EOB_ADDR + 32'h14 refers to MSEBI_CS1_N The MSEBI slave controller writes the block transfer descriptor at address rMSEBIS_EOB_ADDR+offset (writes 0x1234_5678 through AHB bus) when "end of block event" arrives from MSEBI bus master. MSEBI master writes the bMSEBIS_SET_INT_CPUTX or bMSEBIS_SET_INT_DMATX in the rMSEBIS_INT register depending on: • The initiator of the transfer (CPU or DMA) • The side of the transfer (TX) • The MSEBI_CS[n]_N (n = 0..3) used for the transfer ***Note*)** After a read of the descriptor on slave memory, the user needs to clear the descriptor to 32'h0 and reset bMSEBIS_SET_INT_CPUTX or bMSEBIS_SET_INT_DMATX to allow next transfer. See **Section 10.4.5.3, MSEBI Interrupt: End of Block Detection by the Slave**.	R/W
b1, b0	Reserved	These bits [1:0] are reset to 0 (Alignment 32 bits).	R

10.3.4 Register Description MSEBI Slave from MSEBI

10.3.4.1 rMSEBIS_INT — Slave Interrupt Register

CAUTION

This register cannot be access by the CPU. It is reserved for MSEBI bus accesses. MSEBI Master can access this register of each MSEBI Slave using MSEBI_CS[n]_N (n = 0..3) according to **Section 10.2.4, Register Map MSEBI Slave from MSEBI**.

Address: 400C 1000h (MSEBI_CS0_N access)
400C 1400h (MSEBI_CS1_N access)
400C 1800h (MSEBI_CS2_N access)
400C 1C00h (MSEBI_CS3_N access)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	bMSEBIS_SET_INT_DMATX		—	—	bMSEBIS_SET_INT_DMARX		bMSEBIS_SET_INT_CPUTX				bMSEBIS_SET_INT_CPURX			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.44 rMSEBIS_INT Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b14	Reserved	Read as 0.	R
b13, b12	bMSEBIS_SET_INT_DMATX	Set MSEBI Slave interrupt “end of block event” detection on DMA TX channel. When this field is written: • First the block transfer descriptor is written on memory at address at corresponding MSEBI_CS[n]_N (n = 0..1) offset in DMA mode: – MSEBIS_EOB_ADDR – See **Figure 10.42, MSEBI Slave: End of Write Block Transfer from MSEBI CPU Master**. 1’b0: write 0 has no effect 1’b1: write 1 set the corresponding flag “end of block event” on the rMSEBIS_STATUS_INT0 register A read returns the value of the corresponding bit on rMSEBIS_STATUS_INT0 register. Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1. A clear on the corresponding bit of the rMSEBIS_CLR_INT register clears this field.	R/W
b11, b10	Reserved	Read as 0.	R
b9, b8	bMSEBIS_SET_INT_DMARX	Set MSEBI Slave interrupt “end of block event” detection on DMA RX channel. When this register is written: 1’b0: write 0 has no effect 1’b1: write 1 set the corresponding flag “end of block event” on the rMSEBIS_STATUS_INT0 register. A read returns the value of the corresponding bit on rMSEBIS_STATUS_INT0 register. Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..1. A clear on the corresponding bit of the rMSEBIS_CLR_INT register clears this field.	R/W

Table 10.44 rMSEBIS_INT Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b7 to b4	bMSEBIS_SET_INT_CPUTX	Set MSEBI Slave interrupt “end of block event” detection on MSEBI master CPU TX channel. When this field is written: • First the block transfer descriptor is written on memory at address at corresponding MSEBI_CS[n]_N (n = 0..3) offset in CPU mode: – rMSEBIS_EOB_ADDR – See **Figure 10.42, MSEBI Slave: End of Write Block Transfer from MSEBI CPU Master**. 1’b0: write 0 has no effect 1’b1: write 1 set the corresponding flag “end of block event” on the rMSEBIS_STATUS_INT0 register. A read returns the value of the corresponding bit on rMSEBIS_STATUS_INT0 register. Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3. A clear on the corresponding bit of the rMSEBIS_CLR_INT register clears this field.	R/W
b3 to b0	bMSEBIS_SET_INT_CPURX	Set MSEBI Slave interrupt “end of block event” detection on MSEBI master CPU RX channel. When this register is written: 1’b0: write 0 has no effect 1’b1: write 1 set the corresponding “end of block event” on the rMSEBIS_STATUS_INT0 register A read returns the value of the corresponding bit on rMSEBIS_STATUS_INT0 register. Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3. A clear on the corresponding bit of the rMSEBIS_CLR_INT register clears this field.	R/W

10.3.4.2 rMSEBIS_STATUS — Slave Status Register

CAUTION

This register cannot be access by the CPU. It is reserved for MSEBI bus accesses. MSEBI Master can access this register of each MSEBI Slave using MSEBI_CS[n]_N (n = 0..3) according to **Section 10.2.4, Register Map MSEBI Slave from MSEBI**.

Address: 400C 1004h (MSEBI_CS0_N access)
400C 1404h (MSEBI_CS1_N access)
400C 1804h (MSEBI_CS2_N access)
400C 1C04h (MSEBI_CS3_N access)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	bMSEBIS_ERROR_CS_CONFIGURATION				bMSEBIS_ERROR_WEN				bMSEBIS_ERROR_ADDR			
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.45 rMSEBIS_STATUS Register Contents (1/2)

Bit Position	Bit Name	Function	R/W
b31 to b12	Reserved	Read as 0.	R
b11 to b8	bMSEBIS_ERROR_CS_CONFIGURATION	Give MSEBI slave controller error status. A change configuration has been detected inside configuration registers group dedicated to MSEBI_CS[n]_N (n = 0..3) with following potential issues: • Data exchanged between MSEBI master and slave can be lost. • MSEBIS master and slave synchronization can be lost. See bit field: bMSEBIS_CS_ENABLE in rMSEBIS_CONFIG_CS[n]_N register. Read by the master of the MSEBI bus: 1'b0: no error detected 1'b1: MSEBI slave controller has detected a configuration change, data and synchronization can be lost. Clear by the master of the MSEBI bus: Write 1'b0 has no effect Write 1'b1 clear the flag This bit can be used by the master MSEBI to check that a new configuration has been correctly loaded in the slave. Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3.	R/W
b7 to b4	bMSEBIS_ERROR_WEN	Give MSEBI slave controller error status. A write access from the master of the bus has been detected while write protect is enabled on the slave. See bit filed: bMSEBIS_WEN in rMSEBIS_CONFIG_CS[n]_N register. Read by the master of the MSEBI bus: 1'b0: no error detected 1'b1: MSEBI slave controller has detected an error on write access write Clear by the master of the MSEBI bus: Write 1'b0 has no effect Write 1'b1 clear the flag Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3.	R/W

Table 10.45 rMSEBIS_STATUS Register Contents (2/2)

Bit Position	Bit Name	Function	R/W
b3 to b0	bMSEBIS_ERROR_ADDR	Give MSEBI slave controller error status. During an access on AHB master port, NoC has reported an error due to a bad address range. It may be due to an error of the master of the bus or an error on the configuration of addressing mode. See bit field: bMSEBIS_ADDR_MODE in rMSEBIS_CONFIG_CS[n]_N register. Read by the master of the MSEBI bus: 1'b0: no error detected 1'b1: MSEBI slave controller has detect an access to bad address Clear by the master of the MSEBI bus: Write 1'b0 has no effect Write 1'b1 clear the flag Bit [n] of this field refers to the MSEBI_CS[n]_N with n = 0..3.	R/W

10.3.4.3 rMSEBIS_ID_CS[n]_N — Slave ID Register (n = 0..3)

Used by the master to determine if the chip select is available.

CAUTION

This register cannot be access by the CPU. It is reserved for MSEBI bus accesses. MSEBI Master can access this register of each MSEBI Slave using MSEBI_CS[n]_N (n = 0..3) according to **Section 10.2.4, Register Map MSEBI Slave from MSEBI**.

Address: 400C 1008h + 4h × n (MSEBI_CS0_N access)
400C 1408h + 4h × n (MSEBI_CS1_N access)
400C 1808h + 4h × n (MSEBI_CS2_N access)
400C 1C08h + 4h × n (MSEBI_CS3_N access)

Bit	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	bMSEBIS_ID															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	bMSEBIS_ID															
Value after reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10.46 rMSEBIS_ID_CS[n]_N Register Contents

Bit Position	Bit Name	Function	R/W
b31 to b0	bMSEBIS_ID	This register is read by the master of the bus to determine when a chip select is ready for communication. If bMSEBIS_CS_ENABLE =1: • Master read on rMSEBIS_ID_CS[n]_N (n = 0..3) returns 0x1234_FEDn (n = 0..3). If bMSEBIS_CS_ENABLE = 0 • Master read on rMSEBIS_ID_CS[n]_N (n = 0..3) returns 0x0. See **Section 10.4.7.5, MSEBI Slave: Chip select Configuration Status**.	R

10.4 Operation

10.4.1 AHB Interface

10.4.1.1 AHB Slave Interface

MSEBI Master and Slave have independent AHB slaves, respectively.

For MSEBI Master

MSEBI supports 8/16/32 bits AHB access. So, one AHB 32 bits access generates one MSEBI access (Mode32) or two MSEBI accesses (Mode16) or four accesses (Mode8). When AHB burst transfer, the AHB access size is required to be equal to or more than the MSEBI bus size.

For MSEBI Slave

MSEBI supports 8/16/32 bits AHB access.

10.4.1.2 AHB Master Interface (MSEBI Slave only)

Burst size, 4/8/16 words, can be managed on following registers:

- rMSEBIS_CONFIG (CPU only)
- rMSEBIS_DMARDLR_CS[n]_N (DMA receive, n = 0..1)
- rMSEBIS_DMATDLR_CS[n]_N (DMA transmit, n = 0..1)

Attributes of AHB accesses can be managed by the following bits:

- bMSEBIS_AHB_MASTER_BUF
- bMSEBIS_AHB_MASTER_CACHE

AHB master is able to manage an “error” response from the NoC in case of bad address.

See bMSEBIS_ERROR_ADDR in rMSEBIS_STATUS register.

10.4.2 Use Case Device Connection

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface

Signal Name	Description
MSEBI(x)_ACD[31:0]	Address, Control and Data multiplexed on 32, 16 and 8 bits, controlled by MSEBIM_ALE, MSEBIM_CLE, MSEBIM_DLE and latched on rising edge clock of MSEBIM_CLK
MSEBI(x)_CLK	• For Master Mode, the clock can be configured to: MSEBIM_HCLK divided by 2, 3, 4, 5, 6, 7 or 8. • For Slave Mode, the clock is given by the master of the bus. ***Caution***) Clock duty cycle can be different of 50 %
MSEBI(x)_WAIT[n]_N	**For Master Mode** • MSEBIM_WAIT[n]_N can be configured in synchronous or asynchronous mode by bMSEBIM_CONFIG bits. • MSEBIM_WAIT[n]_N are taken into account during the VALID sub phase with MSEBI_CS[n]_N is low (with n = 0..3) and MSEBIM_DLE high when: – RDDLEDATA_NB/WRDLEDATA_NB time expired for the first access of burst access in synchronous mode, or for no burst access in asynchronous or synchronous mode. Or – RDDLEDATA_B/WRDLEDATA_B time expired for all burst access after the first access in synchronous mode. **For Slave Mode** <Only in synchronous mode> • MSEBIS_WAIT[n]_N is generated during the VALID sub phase with MSEBI_CS[n]_N Low (with n: 0..3) and MSEBIS_DLE High when: – RDDLEDATA_NB/WRDLEDATA_NB time expired for the first access of a burst or for no burst access. Or – RDDLEDATA_B/WRDLEDATA_B time expired for all burst access after the first access in synchronous mode.
MSEBI(x)_CLE	Address and Control Latch Enable (active high)
MSEBI(x)_DLE	Data Latch Enable (active high) The first data transfer is done during the length of RDDLEDATA_NB (read) or WRDLEDATA_NB (Write), then (only in burst mode) if MSEBI_DLE is kept active, successive transfer occurs at the burst interval programmed by RDDLEDATA_B (read) or WRDLEDATA_B (write). The burst has a linear address increment and length is undefined.
MSEBI(x)_ALE	Address Latch Enable (active high)
MSEBIM_ALE1	Address Latch Enable (active high) *Used only for Master in Parallel Mode and Mode8 or Mode16*.
MSEBIM_ALE2	Address Latch Enable (active high) *Used only for Master in Parallel Mode and Mode8*.
MSEBIM_ALE3	Address Latch Enable (active high) *Used only for Master in Parallel Mode and Mode8*.
MSEBIM_RD_N	Read enable (active low) Same as inverted MSEBI_DLE during read only *Optional function, used only for Master in asynchronous mode*.
MSEBIM_WR_N	Write enable (active low) Same as inverted MSEBI_DLE during write only *Optional function, used only for Master in asynchronous mode*.

10.4.2.1 One Device, Mode32, Synchronous

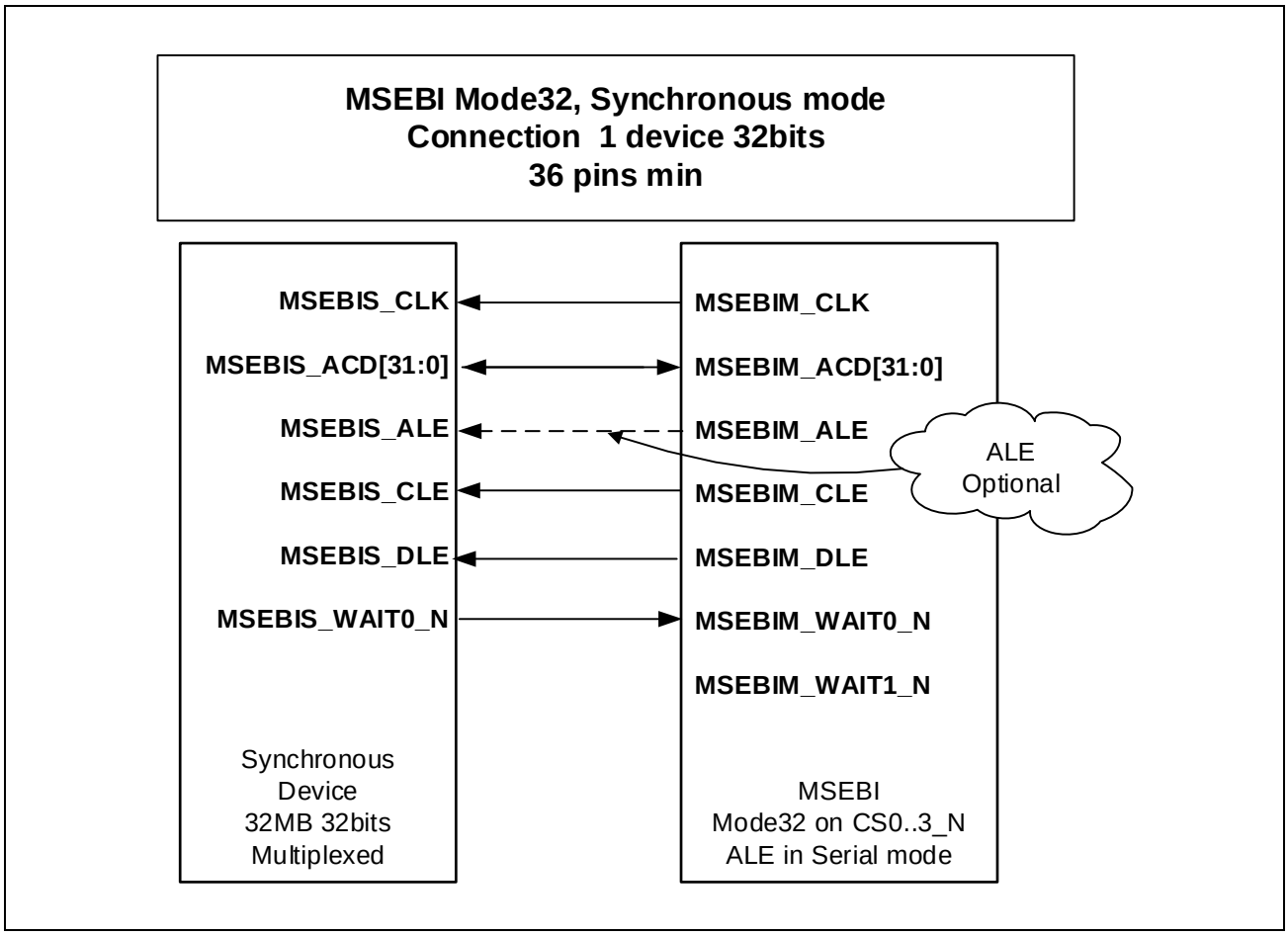

Figure 10.2 One Device, Mode32, Synchronous

10.4.2.2 One Device, Mode16, Synchronous

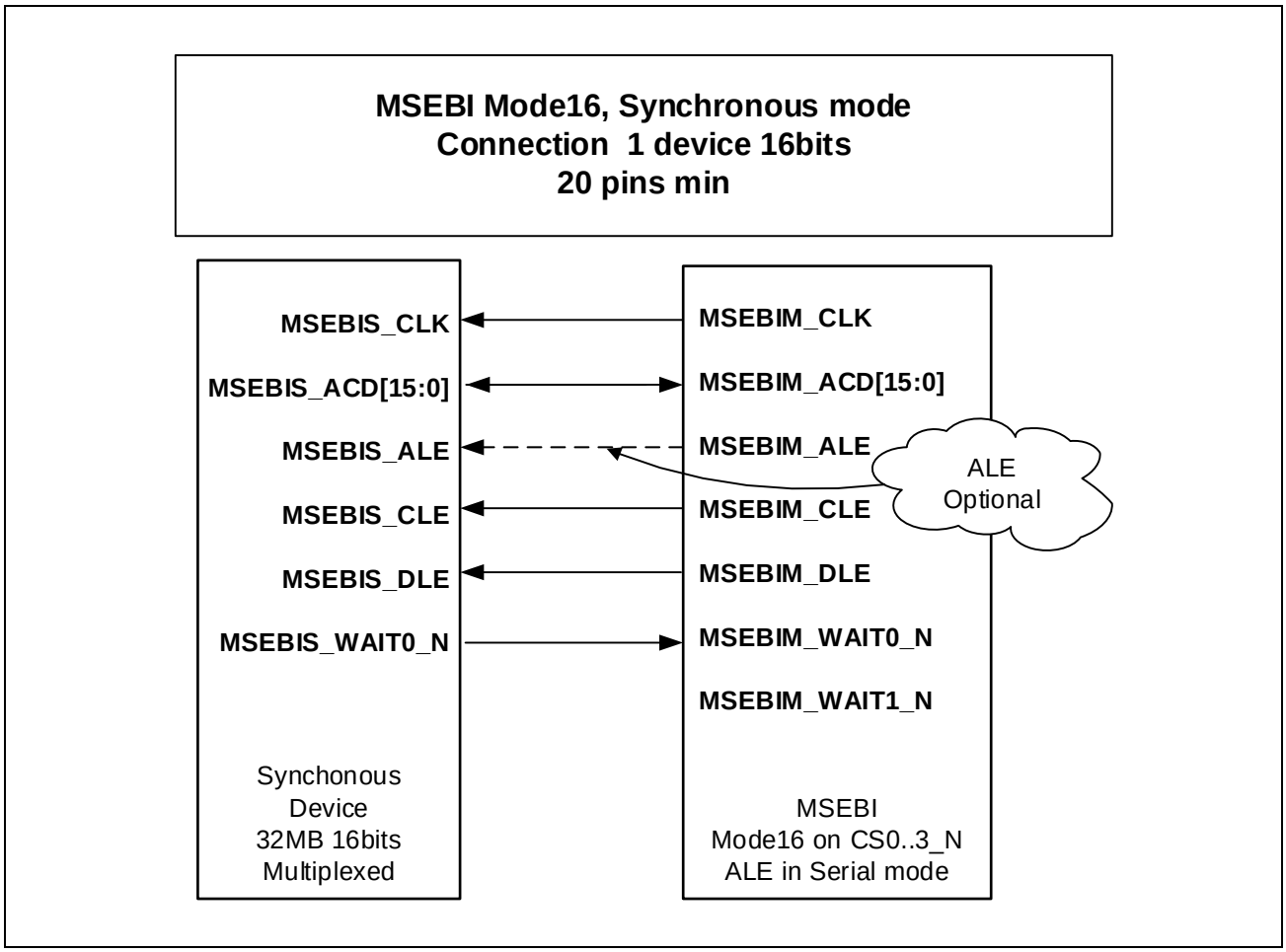

Figure 10.3 One Device, Mode16, Synchronous

10.4.2.3 One Device, Mode8, Synchronous

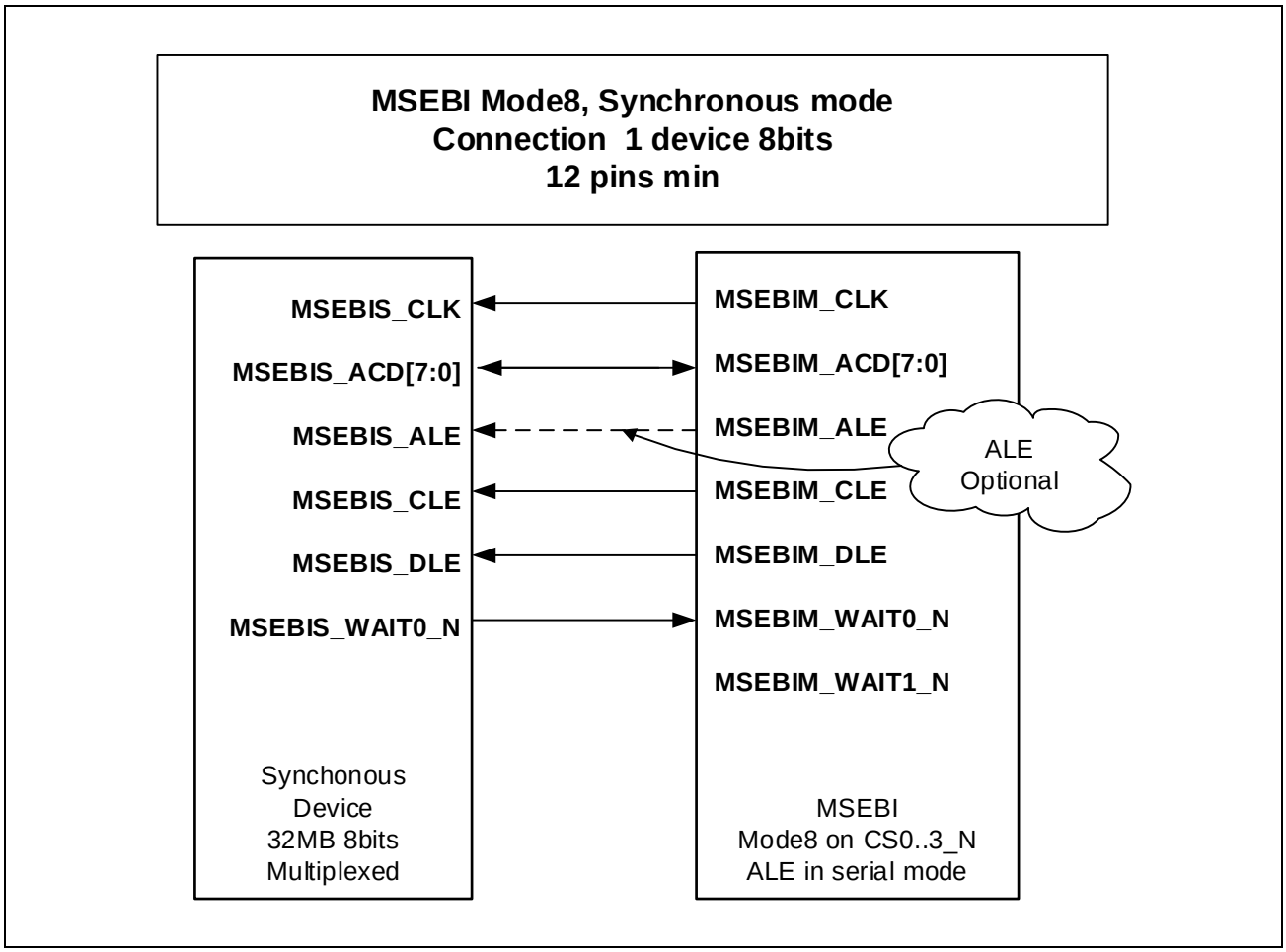

Figure 10.4 One Device, Mode8, Synchronous

10.4.2.4 Three Devices, Mode8/16/32, Synchronous

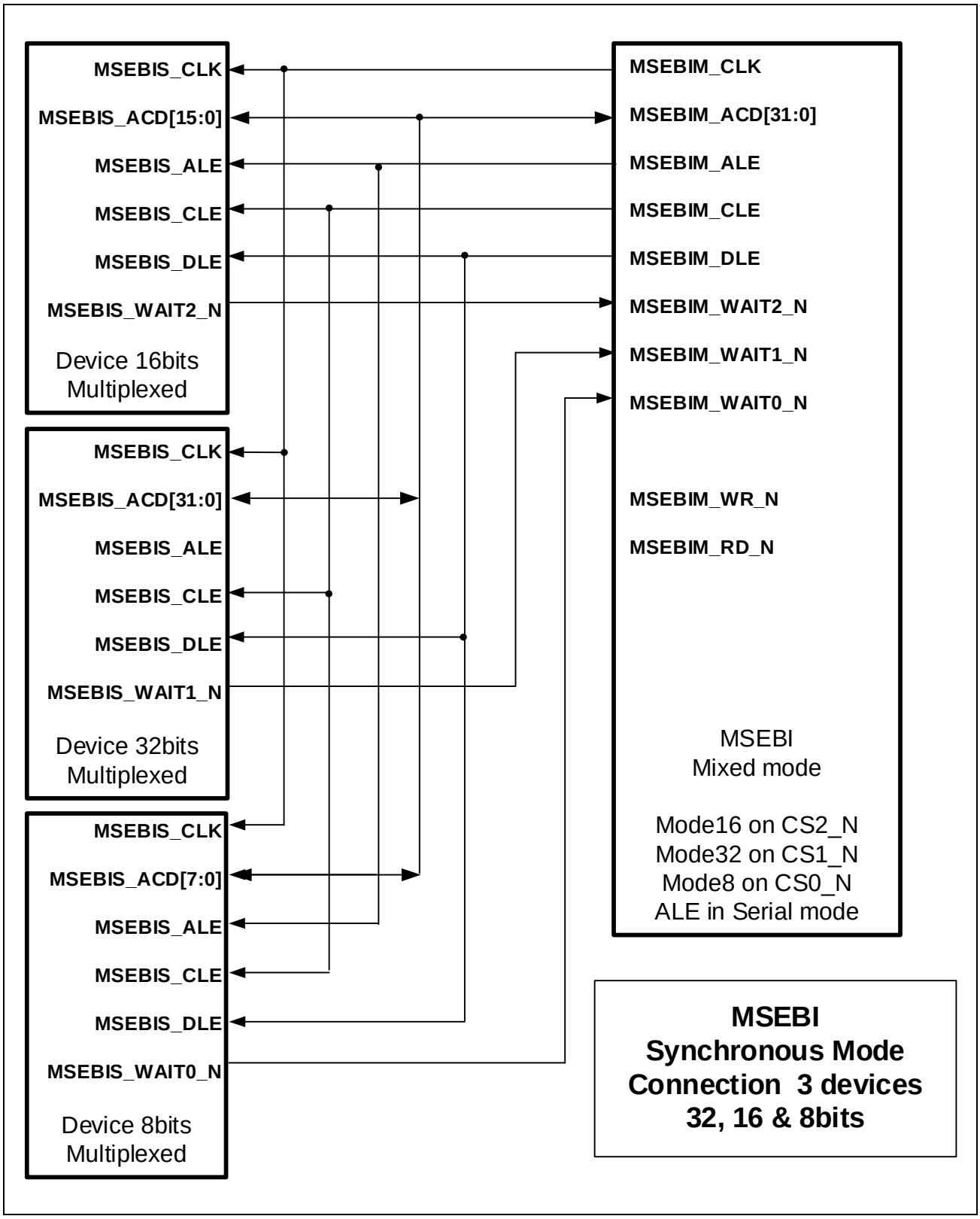

Figure 10.5 Three Devices, Mode8/16/32, Synchronous

10.4.2.5 Three Devices, Mode8/16/32, Asynchronous

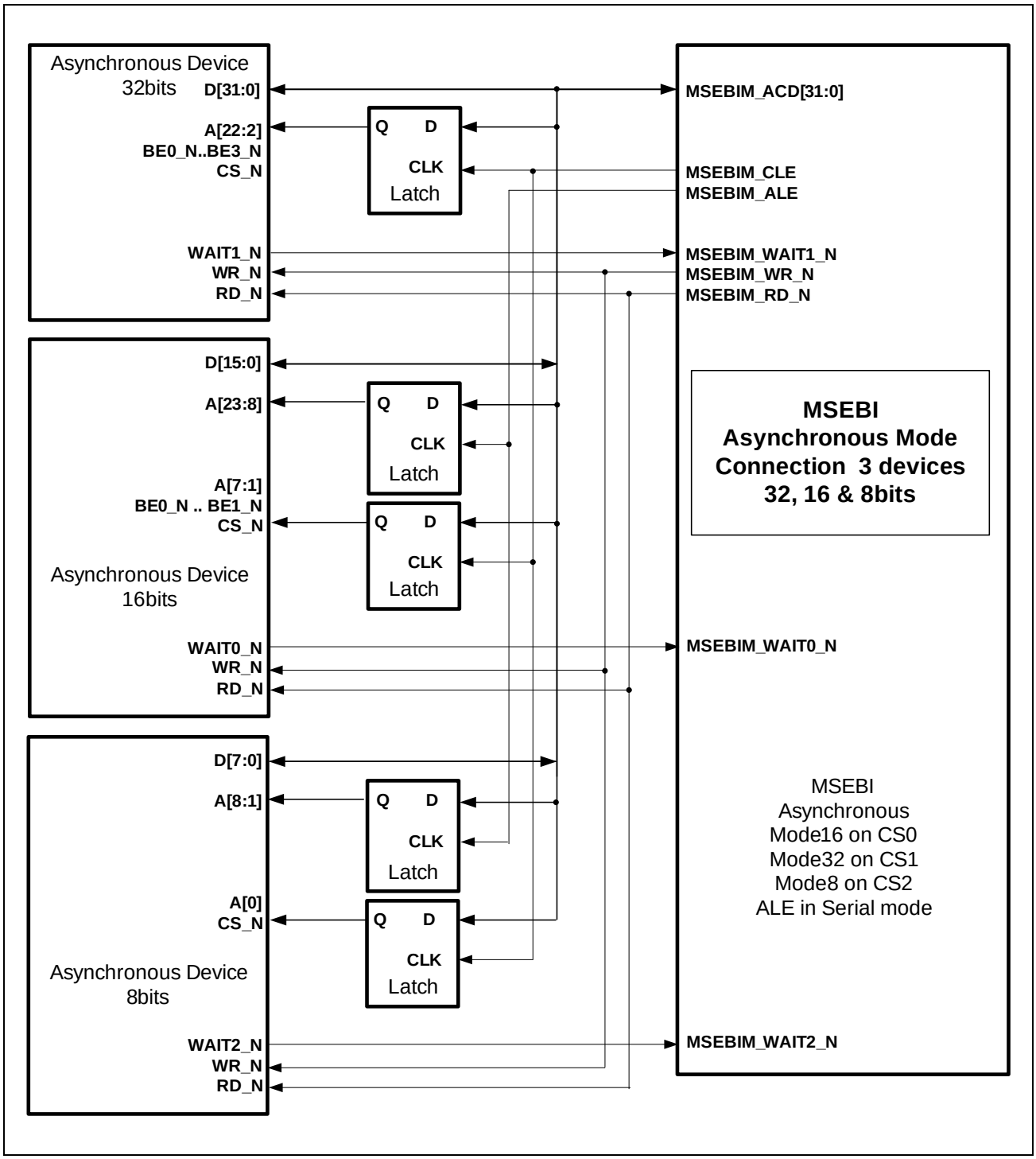

Figure 10.6 Three Devices, Mode8/16/32, Asynchronous

10.4.2.6 Three Devices, Mode8/16/32, Mixed Synchronous and Asynchronous

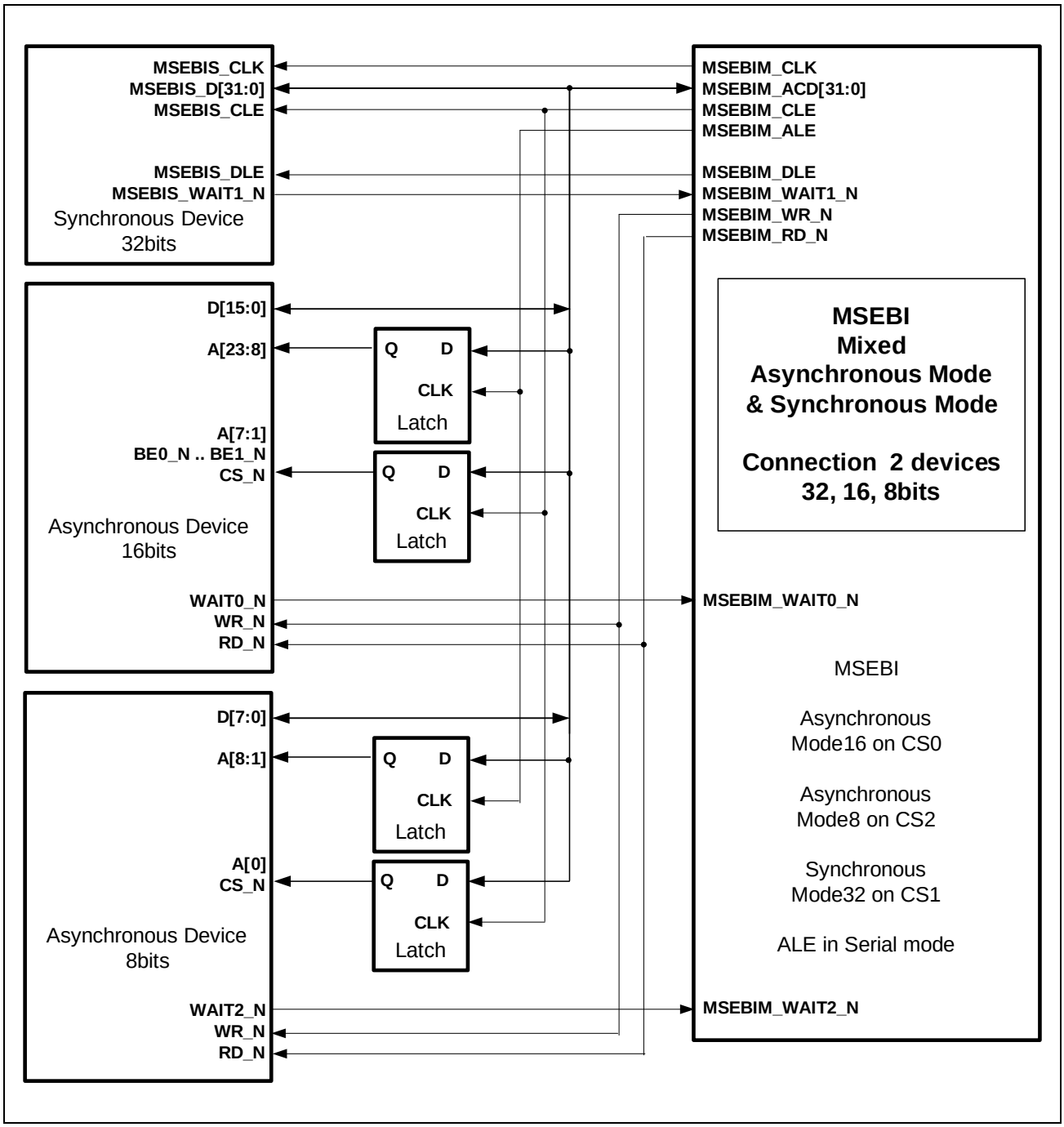

Figure 10.7 Three Devices, Mode8/16/32, Mixed Synchronous and Asynchronous

10.4.2.7 One Device, Mode8, Asynchronous, ALE in Parallel Mode

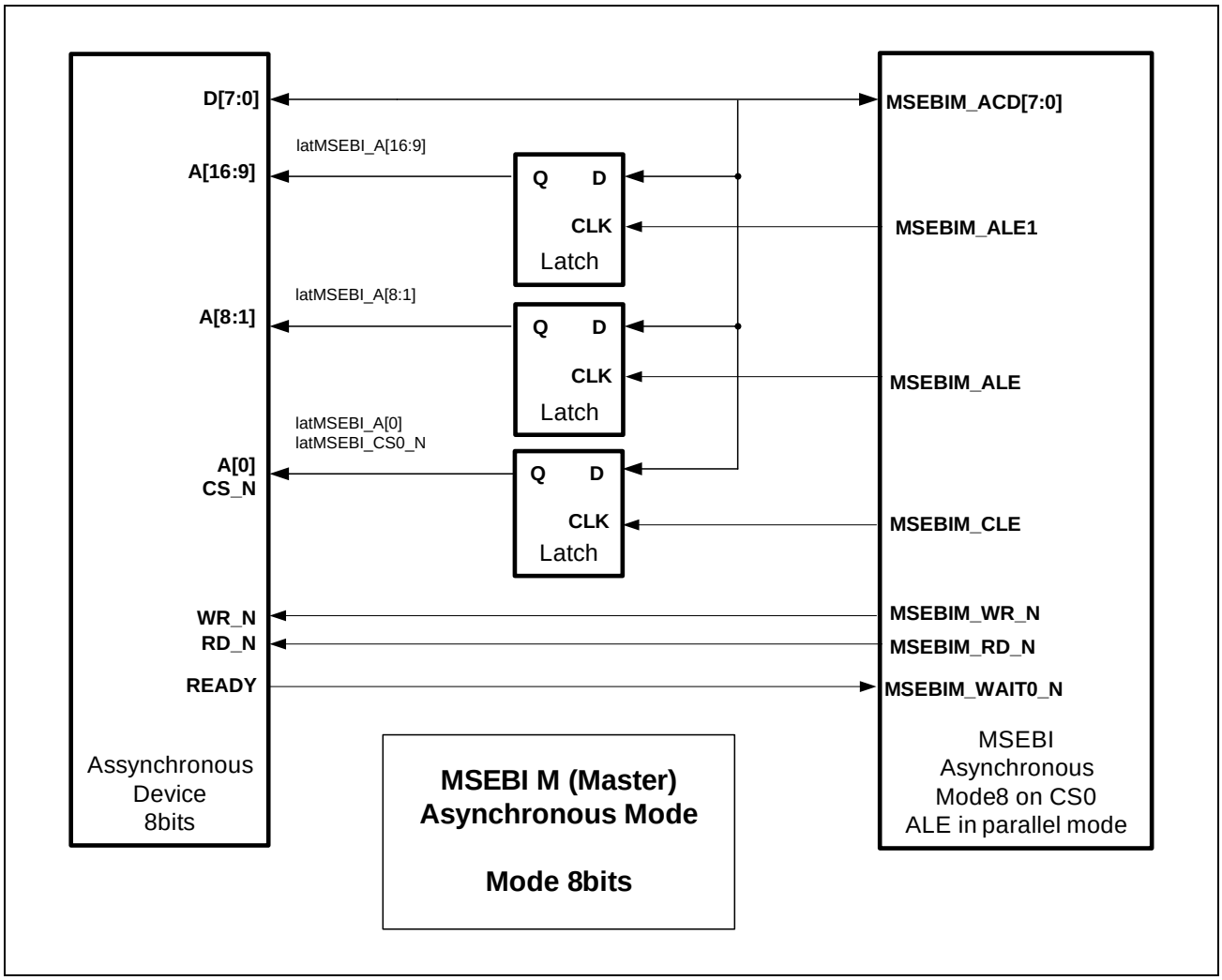

Figure 10.8 One Device, Mode8, Asynchronous, ALE in Parallel Mode

10.4.3 Main Principle of Phase ADDRESS CONTROL and DATA

Accesses are made of 3 major phases:

- Address latch (ADDRESS phase) controlled by MSEBIM_ALE/MSEBIS_ALE (depending if master or slave), this phase is optional
- Control latch (CONTROL phase) controlled by MSEBIM_CLE/MSEBIS_CLE (depending if master or slave)
- Data transfer (DATA phase) controlled by MSEBIM_DLE/MSEBIS_DLE (depending if master or slave)

The following signal are controlled and configured by the master:

- MSEBIM_CLK
- MSEBIM_ALE
- MSEBIM_CLE
- MSEBIM_DLE

The following signals are driven by the master of the bus. The configuration parameters on slave side must be compliant with master parameters:

- MSEBIS_CLK
- MSEBIS_ALE
- MSEBIS_CLE
- MSEBIS_DLE

CAUTION

Due to synchronization limitations, the frequency of the MSEBIS_CLK (generated by the MSEBI master) must be lower than the frequency of the MSEBIS_HCLK (AHB Bus Clock).

10.4.3.1 Address Latch Phase ALE (ADDRESS)

MSEBIM_ALE, MSEBIM_ALE1, MSEBIM_ALE2, MSEBIM_ALE3 are used to latch from bus a part of address depending on the mode in used.

These phases are optional. The assignment of address bits depends on the number of MSEBI_ALE phases used (From 0 until 4 depending on mode).

- See **Table 10.4, MSEBI Mode32, Multiplexer Function on ACD31..0**.
- See **Table 10.6, MSEBI Mode16, Multiplexer Function on ACD15..0**.
- See **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

For Master Mode Only

The length of these phases is configurable per CS[n]_N (n = 0..3) from 1 to 2 MSEBIM_CLK periods (Configured only by master).

In asynchronous mode, an external latch (74x16373 type) can be used to latch address part by connecting MSEBIM_ALE, MSEBIM_ALE1, MSEBIM_ALE2, MSEBIM_ALE3 to inputs of the latch. Use an MSEBI_ALE phase length of 2 to guarantee hold time of external latch, or use a synchronous latch. MSEBIM_ALE, MSEBIM_ALE1, MSEBIM_ALE2, MSEBIM_ALE3 are driven only during the first clock period of this phase.

This phase is optional. The assignment of address bits depends on the number of MSEBI_ALE phases used and allows a maximum flexibility on address capability.

MSEBI_ALE can be configured in serial mode or parallel mode by bMSEBIM_ALE_MODE bit.

- Serial mode is recommended for synchronous interface.
  - See **Figure 10.15, MSEBI Timing, Asynchronous Mode, Read, NoWait, NoBurst, Two ALE Serial Mode** and **Figure 10.17, MSEBI Timing, Asynchronous Mode, Write, NoWait, NoBurst, Two ALE Serial Mode**.
- Parallel mode is recommended for asynchronous interface and connect external latch (74x16373 type), allowing a minimal cost on board.
  - See **Figure 10.16, MSEBI Timing, Asynchronous Mode, Read, NoWait, NoBurst, Two ALE Parallel Mode** and **Figure 10.18, MSEBI Timing, Asynchronous Mode, Write, NoWait, NoBurst, Two ALE Parallel Mode**.

For example, in Mode8, access by CPU

[*Case 1*]

4 MSEBI_ALE phases allow to latch:

- First access: A8..A1
- Second access: A16..A9
- Third access: A24..A17
- Fourth access: A31..A25
- A0 is generated on MSEBIM_CLE phase allowing an address capability of 4 GB.
- 4 Chip selects available: CS[n]_N (n = 0..3)
- MSEBI_A26 .. MSEBI_A0 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A27 are driven by register

See **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

[*Case 2*]

3 MSEBI_ALE phases allow to latch:

- First access: A8..A1
- Second access: A16..A9
- Third access: A24.. A17
- A0 is generated on MSEBIM_CLE phase allowing an address capability of 32 MB.
- 4 Chip selects available: CS[n]_N (n = 0..3)
- MSEBI_A24 .. MSEBI_A0 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A25 are not used in this configuration

See **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

[*Case 3*]

2 MSEBI_ALE phases allow to latch:

- First access: A8..A1
- Second access: A16..A9
- A0 is generated on MSEBIM_CLE phase allowing an address capability of 128 KB.
- 4 Chip selects available: CS[n]_N (n = 0..3)
- MSEBI_A16 .. MSEBI_A0 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A17 are not used in this configuration

See **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

[*Case 4*]

1 MSEBI_ALE phase allows to latch:

- First access: A8..A1
- A0 is generated on MSEBIM_CLE phase allowing an address capability of 512 B.
- 4 Chip selects available: CS[n]_N (n = 0..3)
- MSEBI_A8 .. MSEBI_A0 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A9 are not used in this configuration

See **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

[*Case 5*]

0 MSEBI_ALE phase:

- A0 is generated on MSEBIM_CLE phase allowing an address capability of 2 B.
- 4 Chip selects available: CS[n]_N (n = 0..3)
- MSEBI_A0 is directly driven by CPU
- MSEBI_A31 .. MSEBI_A1 are not used in this configuration

See **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

Each CS[n]_N (n = 1..3) can be used as address bit to extend address capability depending on user use case.

The tables below describe in detail all address capabilities:

- See **Table 10.5, MSEBI Mode32, Chip Selects Management**.
- See **Table 10.7, MSEBI Mode16, Chip Selects Management**.
- See **Table 10.9, MSEBI Mode8, Chip Selects Management**.

For example, in Mode8, access by CPU

[*Case 1*]

1 MSEBI_ALE phase allows to latch:

- First access: A8..A1
- A0 is generated on MSEBIM_CLE phase allowing an address capability of 512 B.
- 4 Chip selects available: CS[n]_N (n = 0..3)
- MSEBI_A8 .. MSEBI_A0 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A9 are not used in this configuration

See **Table 10.9, MSEBI Mode8, Chip Selects Management**.

[*Case 2*]

1 MSEBI_ALE phase allows to latch:

- First access: A8..A1
- CS3_N configured to automatically map: A9
- A0 is generated on MSEBIM_CLE phase allowing an address capability of 1 KB.
- 3 Chip selects available: CS[n]_N (n = 0..2)
- MSEBI_A9 .. MSEBI_A0 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A10 are not used in this configuration

See **Table 10.9, MSEBI Mode8, Chip Selects Management**.

[*Case 3*]

1 MSEBI_ALE phase allows to latch:

- First access: A8..A1
- CS3_N configured to automatically map: A9
- CS2_N configured to automatically map: A10
- A0 is generated on MSEBIM_CLE phase allowing an address capability of 2 KB.
- 2 Chip selects available: CS[n]_N (n = 0..1)
- MSEBI_A10 .. MSEBI_A0 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A11 are not used in this configuration

See **Table 10.9, MSEBI Mode8, Chip Selects Management**.

[*Case 4*]

1 MSEBI_ALE phase allows to latch:

- First access: A8..A1
- CS3_N configured to automatically map: A9
- CS2_N configured to automatically map: A10
- CS1_N configured to automatically map: A11

- A0 is generated on MSEBIM_CLE phase allowing an address capability of 4 KB.
- 1 Chip select available: CS0_N
- MSEBI_A11 .. MSEBI_A0 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A12 are not used in this configuration

See **Table 10.9, MSEBI Mode8, Chip Selects Management**.

For example, in Mode16 access by CPU

[*Case 1*]

1 MSEBI_ALE phase allows to latch:

- First access: A23..A8
- A7..A1 is generated on MSEBIM_CLE phase allowing an address capability of 16 MB.
- 4 Chip selects available: CS[n]_N (n = 0..3)
- MSEBI_A23 .. MSEBI_A1 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A24 are not used in this configuration

See **Table 10.7, MSEBI Mode16, Chip Selects Management**.

[*Case 2*]

1 MSEBI_ALE phase allows to latch:

- First access: A23..A8
- CS3_N configured to automatically map: A24
- A7..A1 is generated on MSEBIM_CLE phase allowing an address capability of 32 MB.
- 3 Chip selects available: CS[n]_N (n = 0..2)
- MSEBI_A24 .. MSEBI_A1 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A25 are not used in this configuration

See **Table 10.7, MSEBI Mode16, Chip Selects Management**.

[*Case 3*]

1 MSEBI_ALE phase allows to latch:

- First access: A23..A8
- CS3_N configured to automatically map: A24
- CS2_N configured to automatically map: A25
- A7..A1 is generated on MSEBIM_CLE phase allowing an address capability of 64 MB.
- 2 Chip selects available: CS[n]_N (n = 0..1)
- MSEBI_A25 .. MSEBI_A1 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A26 are not used in this configuration

See **Table 10.7, MSEBI Mode16, Chip Selects Management**.

[*Case 4*]

1 MSEBI_ALE phase allows to latch:

- First access: A23..A8
- CS3_N configured to automatically map: A24
- CS2_N configured to automatically map: A25
- CS1_N configured to automatically map: A26
- A7..A1 is generated on MSEBIM_CLE phase allowing an address capability of 128 MB.
- 1 Chip select available: CS0_N
- MSEBI_A26 .. MSEBI_A1 are directly driven by CPU
- MSEBI_A31 .. MSEBI_A27 are not used in this configuration

See **Table 10.7, MSEBI Mode16, Chip Selects Management**.

(1) Remarks Regarding ALE Phase Behavior

For Master

The number of ALE phases directly impacts the addressing range of the master.

It is obvious that a wrap effect will appear when the address given by the CPU is not reachable by the MSEBI bus due to insufficient number of ALE phases.

For example, a master is configured in 8bits with 1 ALE phase (no rerouting), thus it can access 512 B of data (range [A8:A0] or 0x000001FF to 0x00000000) through the MSEBI bus.

If the CPU wants to put the address 0x000002F4 (756th byte) on the MSEBI bus, it will be impossible to do so without more ALE phases (address will be: 0x000000F4): a wrap just occurred.

For Slave

If the master is configured to generate a burst on the MSEBI bus, the address is given only once to the slave, at the beginning of the burst.

The slave then generates by itself, for each beat of the burst, the corresponding address.

Using that, it is possible for a master to access data outside of the normal address range given by the number of ALE phases.

However, this possibility must not be used as it is not reliable, because a burst is generated by the master as soon as the number of data items in FIFO is sufficient and burst is enabled (only maximum burst length is configurable, not minimum).

As we cannot predict the size of the burst (or even if a burst will occur or not), it is not advisable to try to access data out of address range.

[Example 1]

- The master is configured in 8bits with 1 ALE phase (no rerouting), thus it can access 512 B of data (range [A8:A0] or 0x000001FF to 0x00000000) through the MSEBI bus.
- If the CPU starts a burst4 at address 0x000001FF, the slave will correctly managed addresses for each beat, i.e. it starts with 0x000001FF, then it generates addresses 0x00000200, then 0x00000201, and last 0x00000202.

- If burst was not used, we would have ended with wrapping single access at addresses: 0x000001FF, 0x00000000, 0x00000001, 0x00000002.

[Example 2]

- The master is configured in 16bits with 1 ALE phase (no rerouting), thus it can access 16 MB of data (range [A23:A0] or 0x00FFFFFF to 0x00000000) through the MSEBI bus.
- If the CPU starts a burst4 at address 0x00FFFFFF, the slave will correctly managed addresses for each beat, i.e. it starts with 0x00FFFFFF, then it generates addresses 0x01000000, then 0x01000001, and last 0x01000002.
- If burst was not used, we would have ended with wrapping single accesses at addresses: 0x00FFFFFF,0x00000000, 0x00000001, 0x00000002.

10.4.3.2 Control Latch Phase CLE (CONTROL)

MSEBIM_CLE/MSEBIS_CLE is used to latch from bus low part of address bus and control: R/W_N, BE[n]_N, DMA_N, CSREG_N, CS[n]_N depending on mode in use (n = 0..3).

- See **Table 10.4, MSEBI Mode32, Multiplexer Function on ACD31..0**.
- See **Table 10.6, MSEBI Mode16, Multiplexer Function on ACD15..0**.
- See **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

For Master Mode Only

The length of this phase is configurable per CS[n]_N from 1 or 2 MSEBIM_CLK/MSEBIS_CLK periods.

In asynchronous mode, an external latch (74x16373 type) can be used to latch all signals by connecting MSEBIM_CLE to inputs of the latch. Use an MSEBI_CLE phase length of 2 to guarantee hold time of external latch, or use a synchronous latch. MSEBI_CLE is driven only during the first clock period of this phase.

10.4.3.3 Data Phase SETUP + VALID + HOLD (DATA)

Data phase consist of 3 sub phases: SETUP, VALID and HOLD.

(1) SETUP

[On Write command]

SETUP sub phase is used to extend address, control, data setup time before the start of Write command.
See bMSEBIM_WRDLESETUP in rMSEBIM_SETUPHOLD_CS[n]_N / bMSEBIS_WRDLESETUP in rMSEBIS_SETUPHOLD_CS[n]_N bits.

[On Read command]

SETUP sub phase is also used to extend address, control signal setup time before the start of Read command. During this sub phase, data bus is floating to avoid bus conflict with external buffer. It is like turn around state (See bMSEBIM_RDDLESETUP in rMSEBIM_SETUPHOLD_CS[n]_N / bMSEBIS_RDDLESETUP in rMSEBIS_SETUPHOLD_CS[n]_N).

The length of this phase is configurable per CS[n]_N (n = 0..3) from 0 to 63 MSEBIM_CLK periods.

CAUTION

Length “0” should not be used for read access to external peripheral (to avoid bus conflict).

(2) VALID

For Master

During VALID sub phase, MSEBIM_DLE and MSEBIM_RD_N (asynchronous mode only) or MSEBIM_WR_N (asynchronous mode only) are actively driven.

For Slave

During VALID sub phase, MSEBIS_DLE is asserted.

For Both (n = 0..3)

In Single transfer mode, data are sampled at the end of last VALID clock period. The length of this phase is configurable per CS[n]_N from 1 to 256 MSEBIM_CLK/MSEBIS_CLK.
See bMSEBIM_RDDLEDATA_NB and bMSEBIM_WRDLEDATA_NB bits in rMSEBIM_CYCLESIZE_CS[n]_N for master.
See bMSEBIS_RDDLEDATA_NB and bMSEBIS_WRDLEDATA_NB bits in rMSEBIS_CYCLESIZE_CS[n]_N for slave.

In Burst mode (Synchronous Mode only), first data is sampled like for single transfer mode, at end of programmed length. Then the phase is extended per CS[n]_N by RDDLEDATA_B and WRDLEDATA_B (or RDDLEDATA_B and WRDLEDATA_B for slave) between 1..4 MSEBIM_CLK for each access in the burst cycle. The next data are sampled at each burst period clock interval. Address is linearly incremented, never cross a 1 kB boundary.

For increment we can have the following values:

- Mode32: 4 Bytes.
- Mode16: 2 Bytes.
- Mode8: 1 Byte.

Slave is allowed to prefetch data in burst read. (so that burst transfer should not be done on FIFO for instance).

CAUTION

The burst length is undefined from 1/2/4 (depending on mode) to 1Kbytes and never cross a 1 kB boundary.

Burst transfer can be aborted by DMA/CPU.

If external MSEBIM_WAIT[n]_N / MSEBIS_WAIT[n]_N (n = 0..3) is implemented, the VALID sub phase is paused as long as wait is signaled.

Outside of VALID sub phase, wait signal is ignored.

(3) HOLD

[On Write command]

Hold sub phase is used to maintain address, control, data hold time after the end of Write command

See bMSEBIM_WRDLEHOLD bits in rMSEBIM_SETUPHOLD_CS[n]_N for master.

[On Read command]

Hold sub phase is used to put the data bus in floating to avoid bus conflict with external buffer.
It is like turn around state.

See bMSEBIM_RDDLEHOLD bits in rMSEBIM_SETUPHOLD_CS[n]_N for master.

This sub phase is recommended on Read command and can also be used in Write command to delay the next cycle. At end of this phase, during read cycle, the peripheral shall have returned its data bus to floating.

For Master Only

The length of this phase is configurable per CS[n]_N (n = 0..3) from 0 to 63 MSEBIM_CLK periods.

CAUTION

Length “0” should not be used for read access to external peripheral (to avoid bus conflict).

10.4.4 MSEBI Timing

10.4.4.1 Asynchronous Mode, One ALE

For Master Mode Only

To manage the asynchronous mode, MSEBI interface must be configured with following features (n = 0..3):

- Asynchronous is enabled (Set bMSEBIM_CONFIG bits)
- Burst mode is always disabled, bMSEBIM_BURST_ENABLE bit is ignored
- External MSEBIM_WAIT[n]_N is generated by an external slave on the bus and synchronized on internal MSEBIM_HCLK clock (take into account potential latency from 1 to 3 periods of MSEBIM_HCLK clock and take into account when MSEBI_CS[n]_N is Low and MSEBIM_DLE is High, and at the end of VALID sub phase when RDDLEDATA_NB (read) or WRDLEDATA_NB (write) time expired).
- MSEBI master bus controls all following signals in asynchronous mode:
  - MSEBIM_CLK
  - MSEBIM_ALE
  - MSEBIM_CLE
  - MSEBIM_DLE
  - MSEBIM_RD_N
  - MSEBIM_WR_N

NOTE

The timing parameter must be compliant between master and slave.

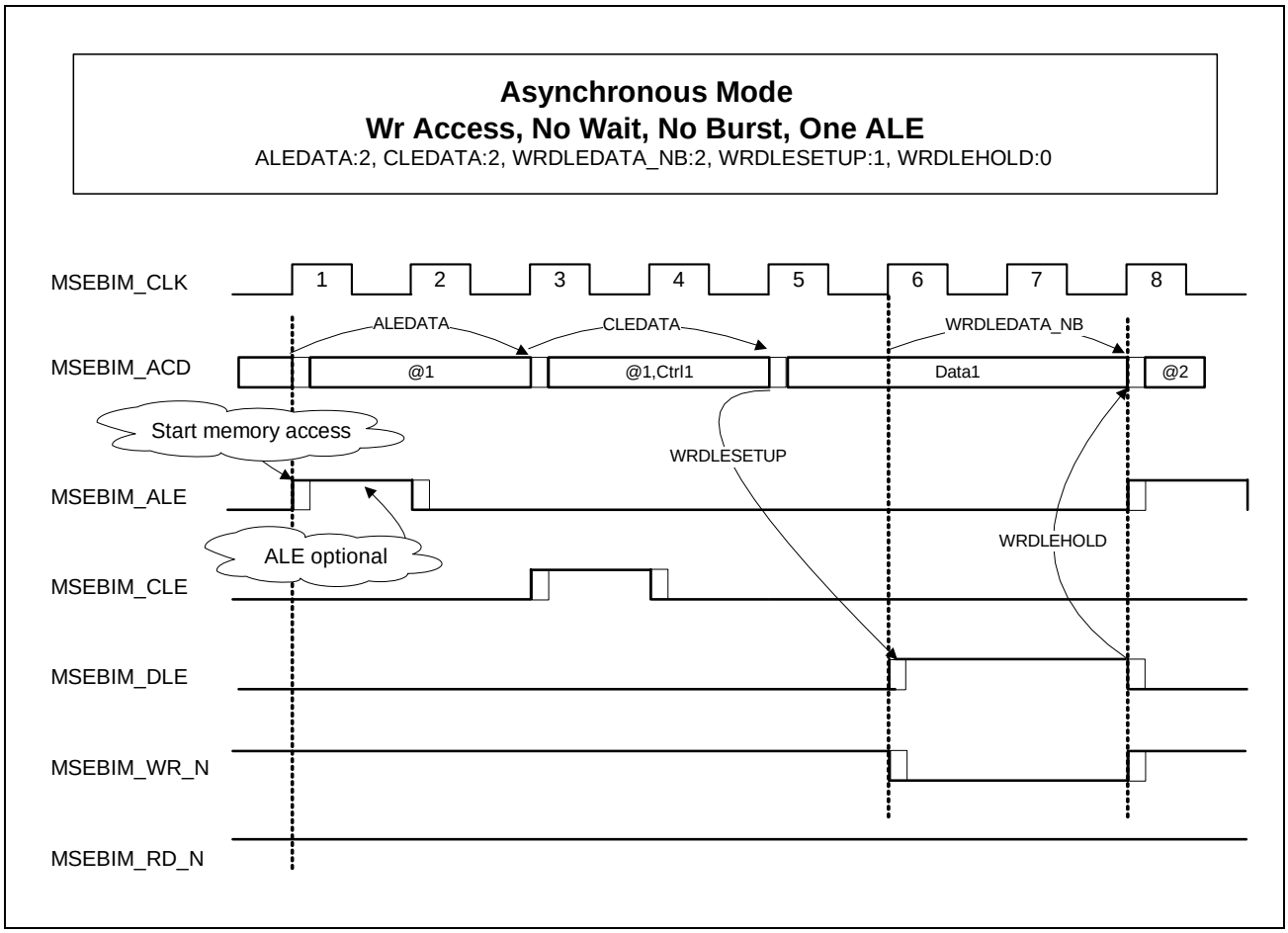

Figure 10.9 MSEBI Timing, Asynchronous Mode, Write1, NoWait, NoBurst, One ALE

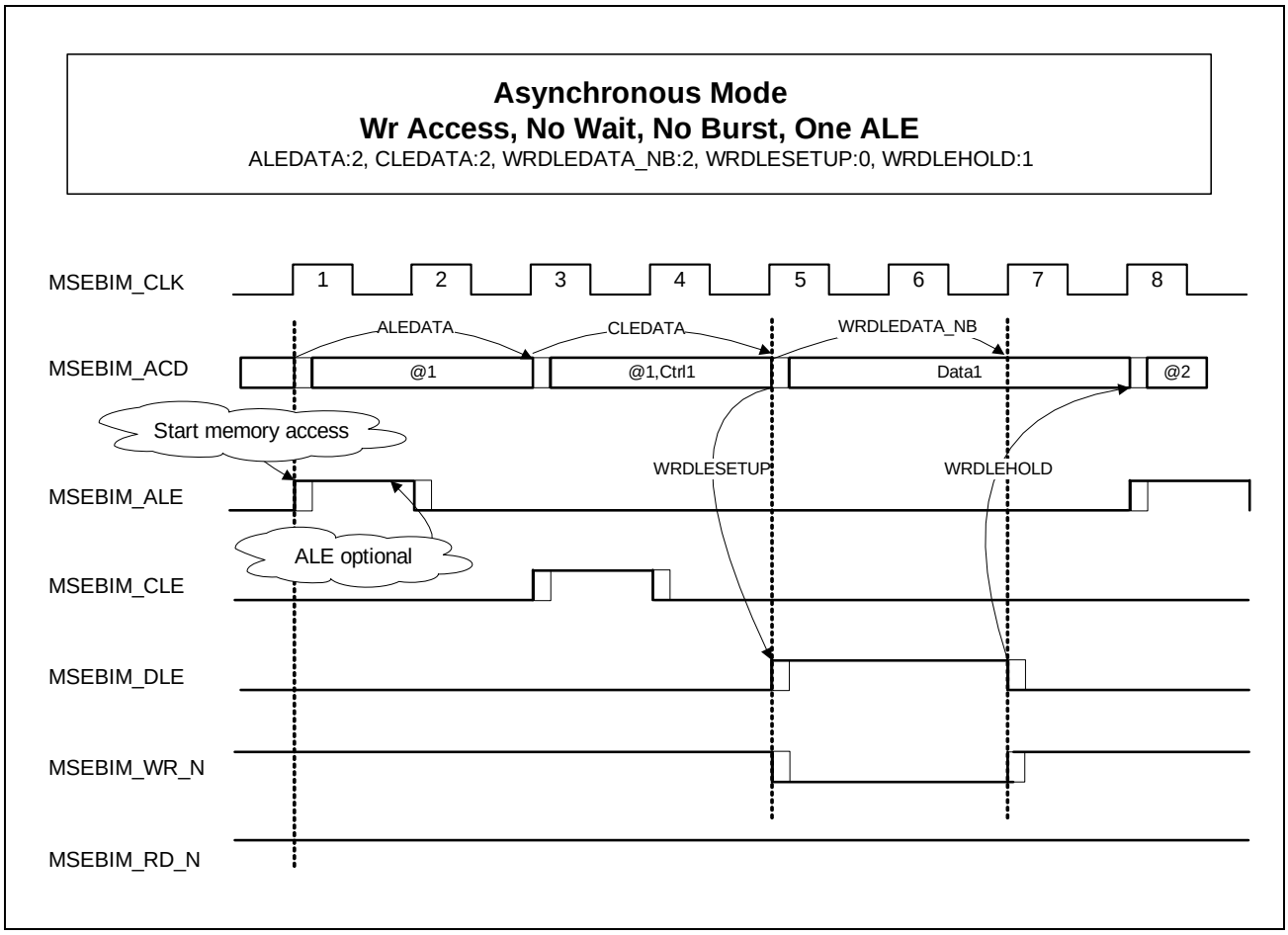

Figure 10.10 MSEBI Timing, Asynchronous Mode, Write2, NoWait, NoBurst, One ALE

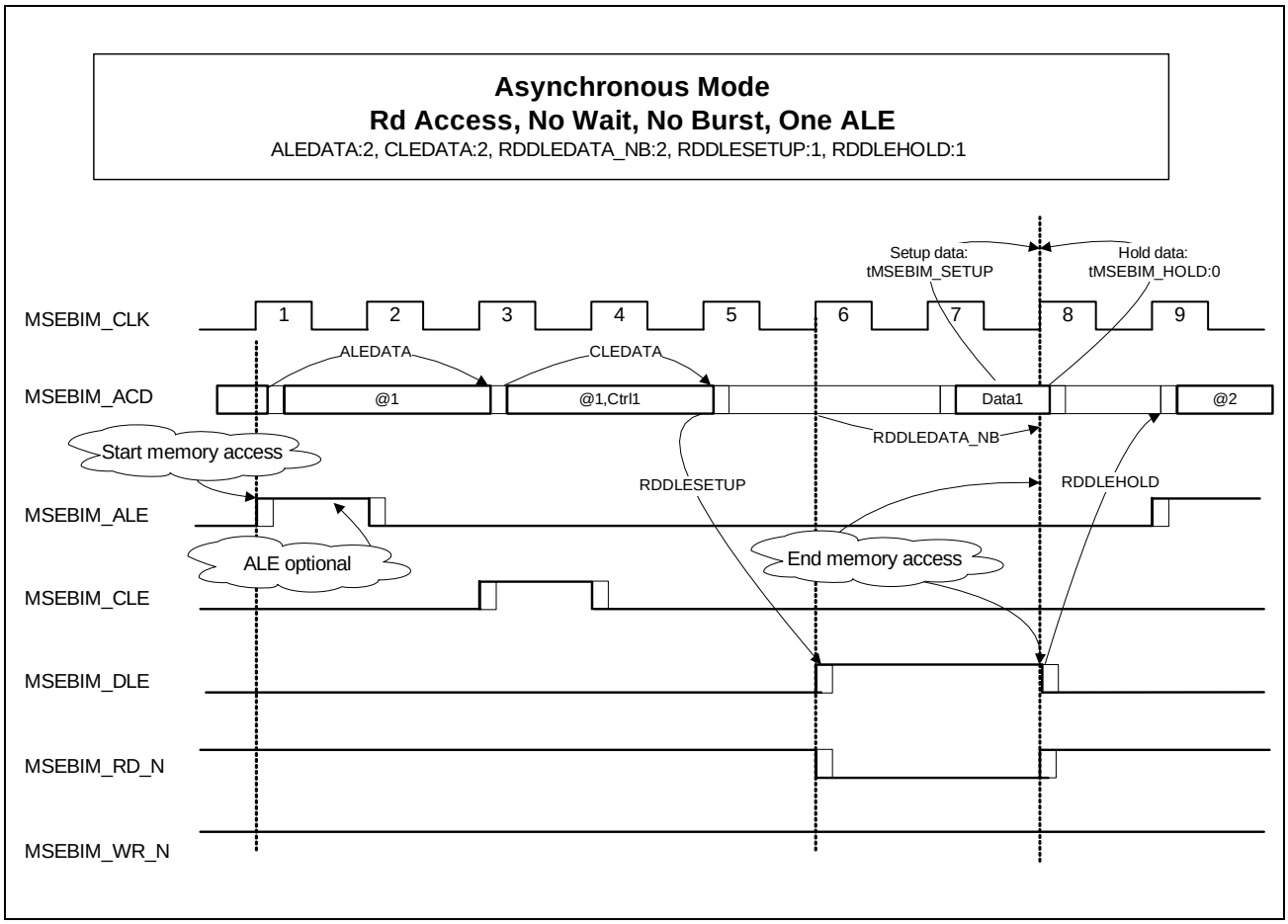

Figure 10.11 MSEBI Timing, Asynchronous Mode, Read1, NoWait, NoBurst, One ALE

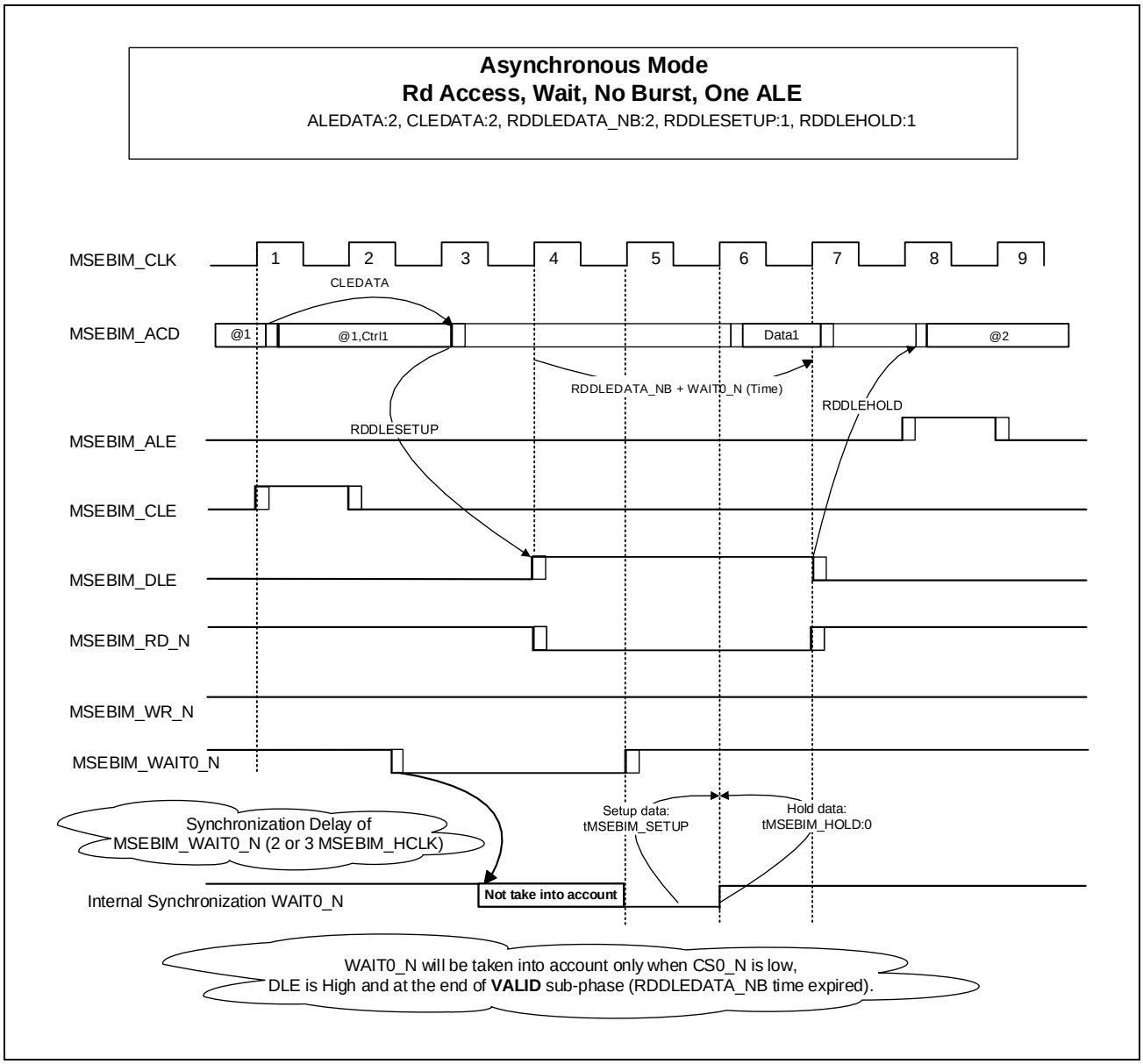

Figure 10.12 MSEBI Timing, Asynchronous Mode, Read2, Wait, NoBurst, One ALE

10.4.4.2 Asynchronous Mode, No ALE MSEBI Master Only

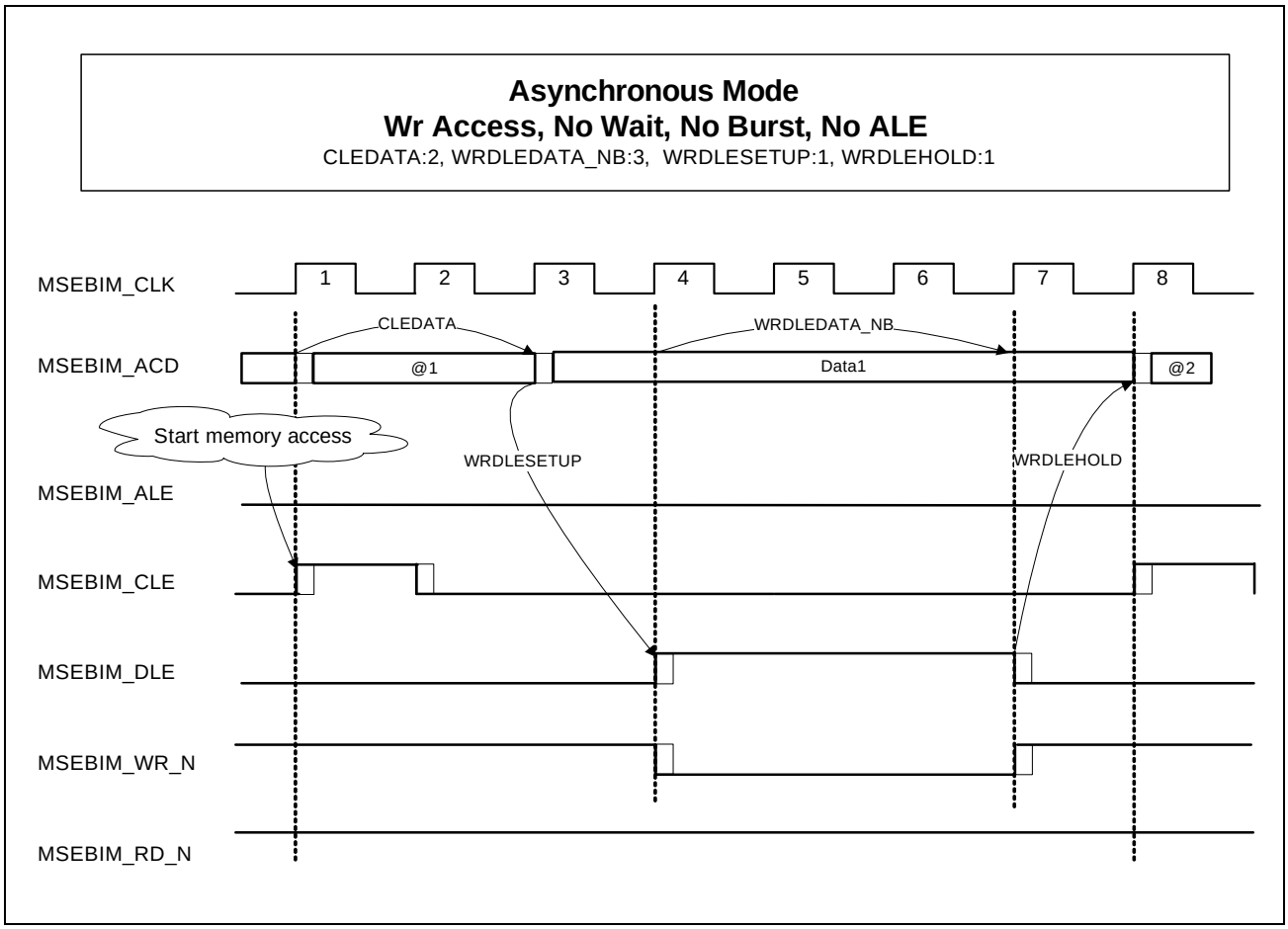

Figure 10.13 MSEBI Timing, Asynchronous Mode, Write1, No Wait, NoBurst, No ALE

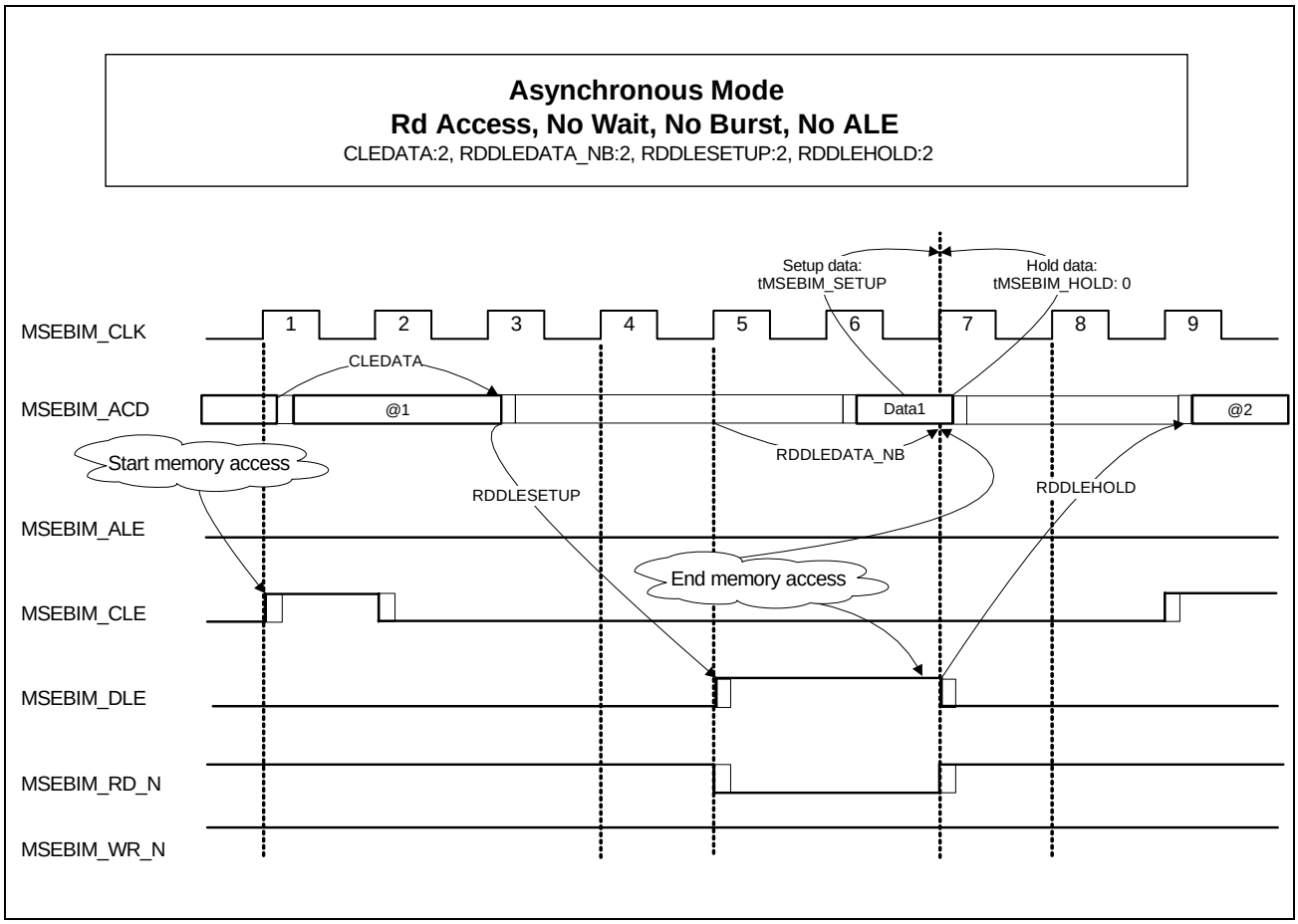

Figure 10.14 MSEBI Timing, Asynchronous Mode, Read1, No Wait, NoBurst, No ALE

10.4.4.3 Asynchronous Mode, Two ALE

For Master Mode Only

To manage the asynchronous mode, MSEBI interface must be configured with following features (n = 0..3):

- Asynchronous enabled (Set bMSEBIM_CONFIG bits).
- Burst mode is always disabled, bMSEBIM_BURST_ENABLE bit is ignored
- External MSEBIM_WAIT[n]_N is generated by an external slave on the bus and synchronized on internal MSEBIM_HCLK clock (take into account potential latency from 1 to 3 periods of MSEBIM_HCLK clock and take into account when MSEBI_CS[n]_N is Low and MSEBIM_DLE is High, and at the end of VALID sub phase when RDDLEDATA_NB (read) or WRDLEDATA_NB (write) time expired).
- MSEBI master bus controls all following signals in asynchronous mode:
  - MSEBIM_CLK
  - MSEBIM_ALE (optional MSEBIM_ALE1), 2 modes available
    Serial, ALE are generated on same line (MSEBIM_ALE)
    Parallel mode, each ALE is routed on different line (MSEBIM_ALE, MSEBIM_ALE1)
  - MSEBIM_CLE
  - MSEBIM_DLE
  - MSEBIM_RD_N
  - MSEBIM_WR_N

NOTE

The timing parameter must be compliant between master and slave.

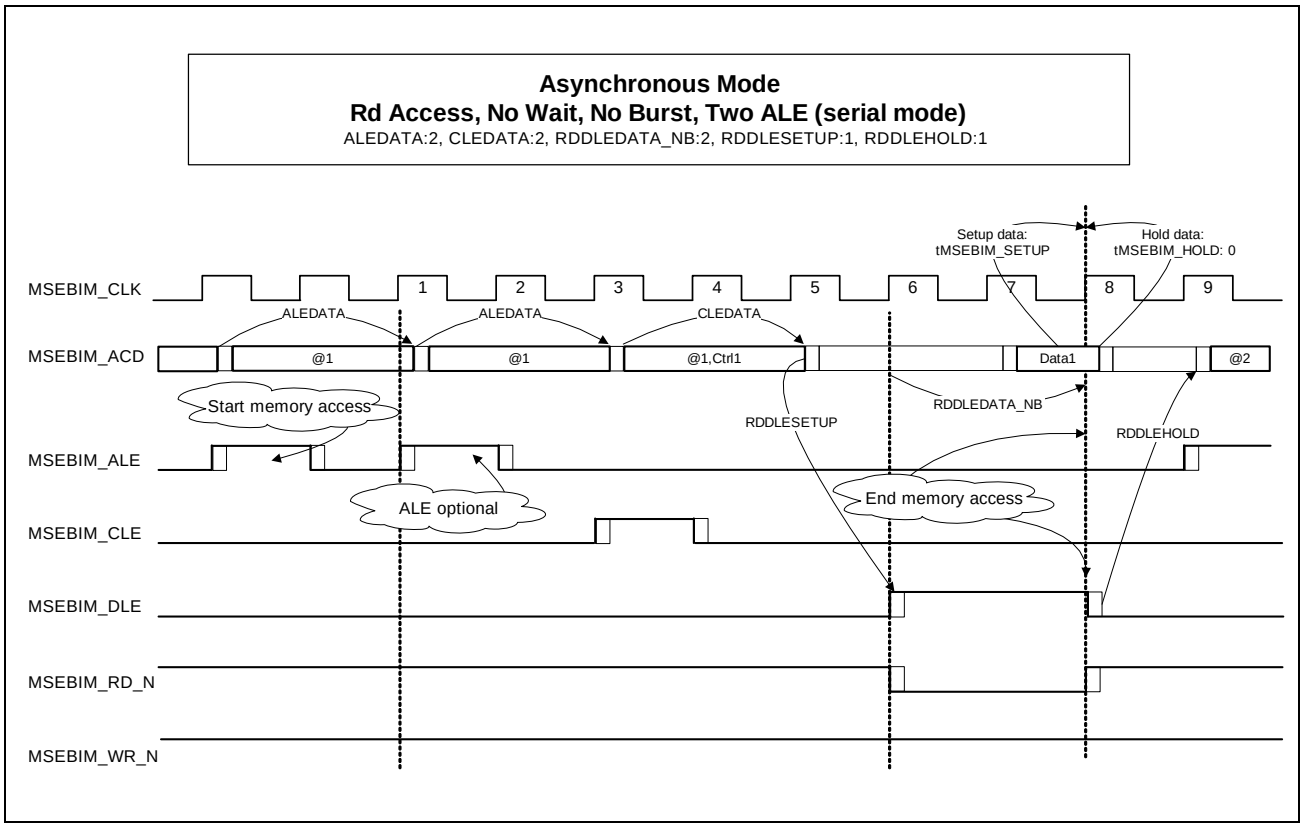

Figure 10.15 MSEBI Timing, Asynchronous Mode, Read, NoWait, NoBurst, Two ALE Serial Mode

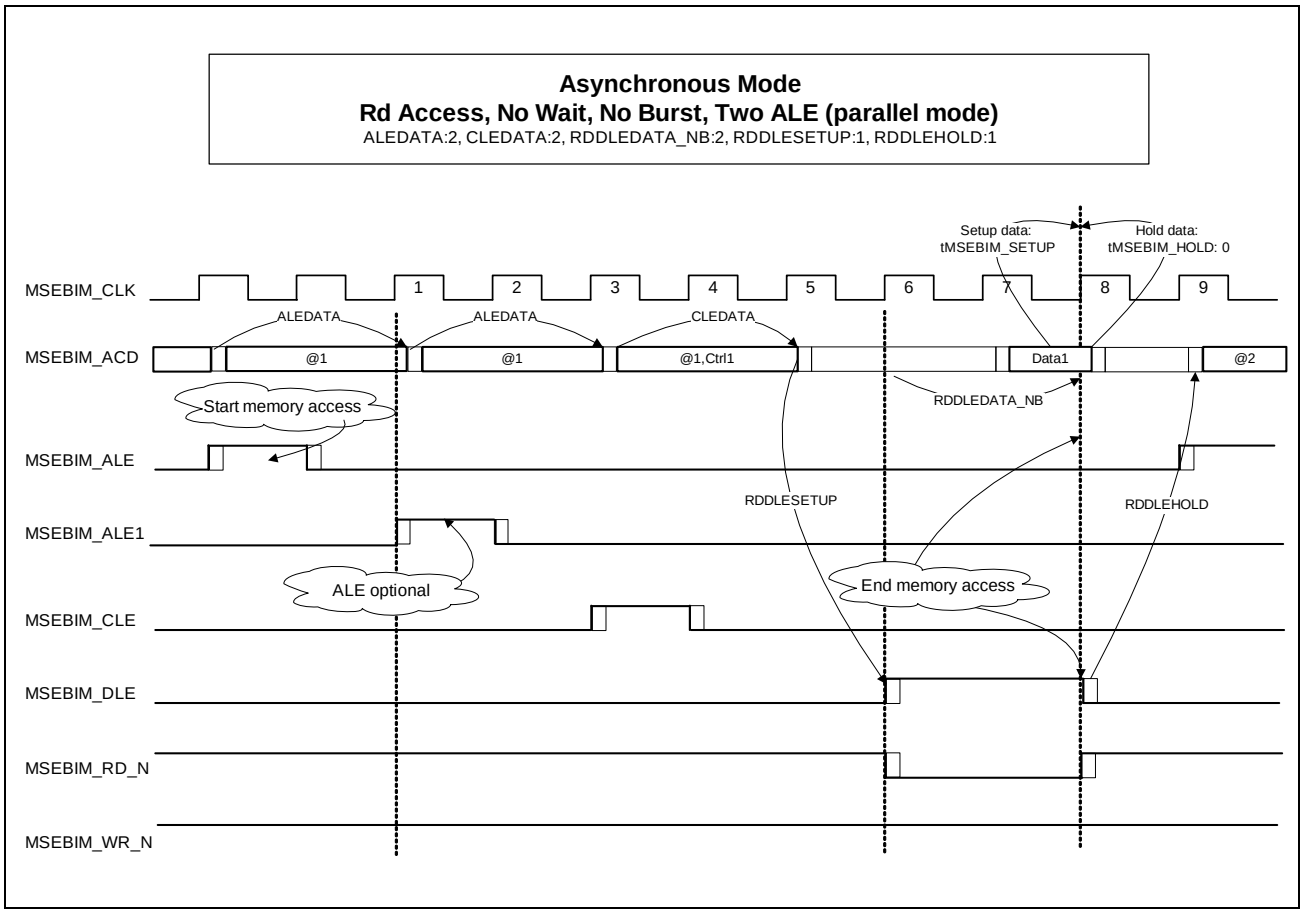

Figure 10.16 MSEBI Timing, Asynchronous Mode, Read, NoWait, NoBurst, Two ALE Parallel Mode

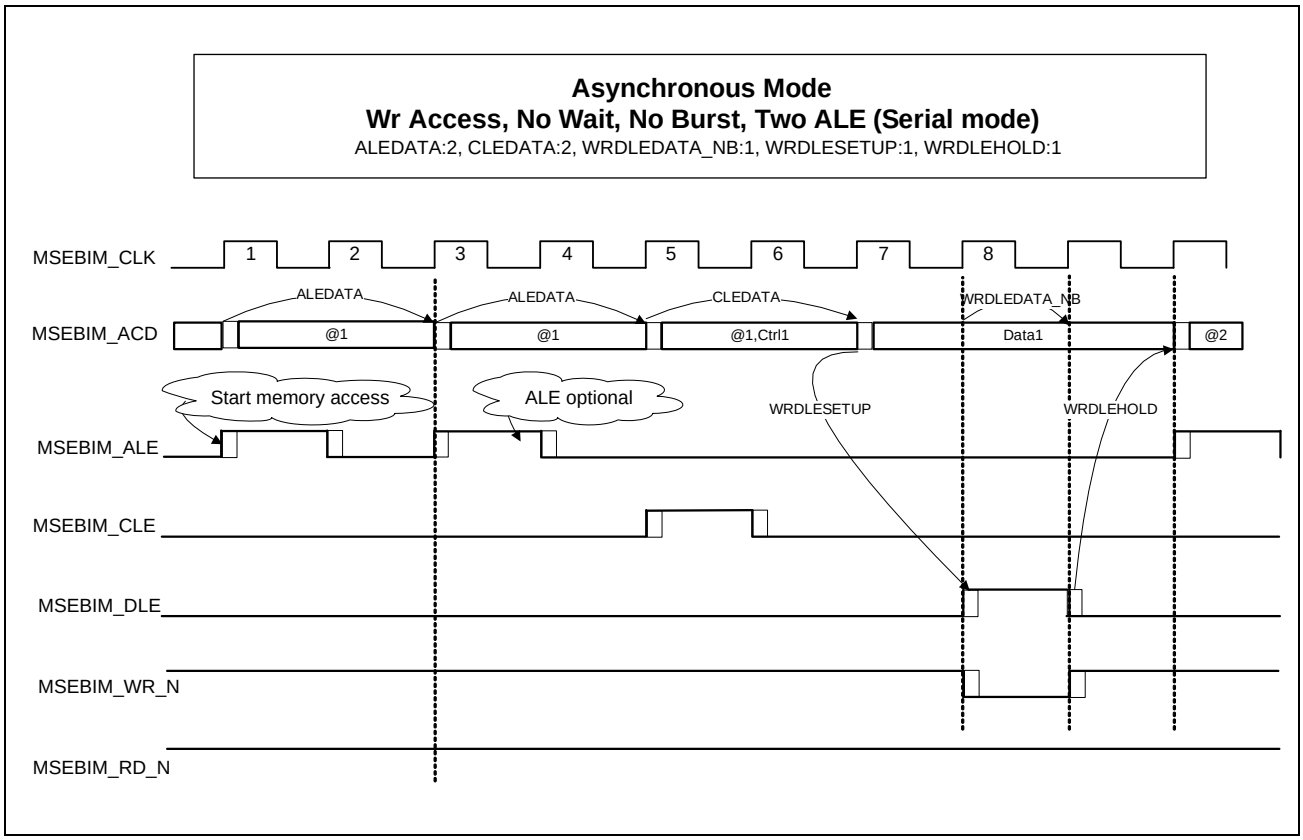

Figure 10.17 MSEBI Timing, Asynchronous Mode, Write, NoWait, NoBurst, Two ALE Serial Mode

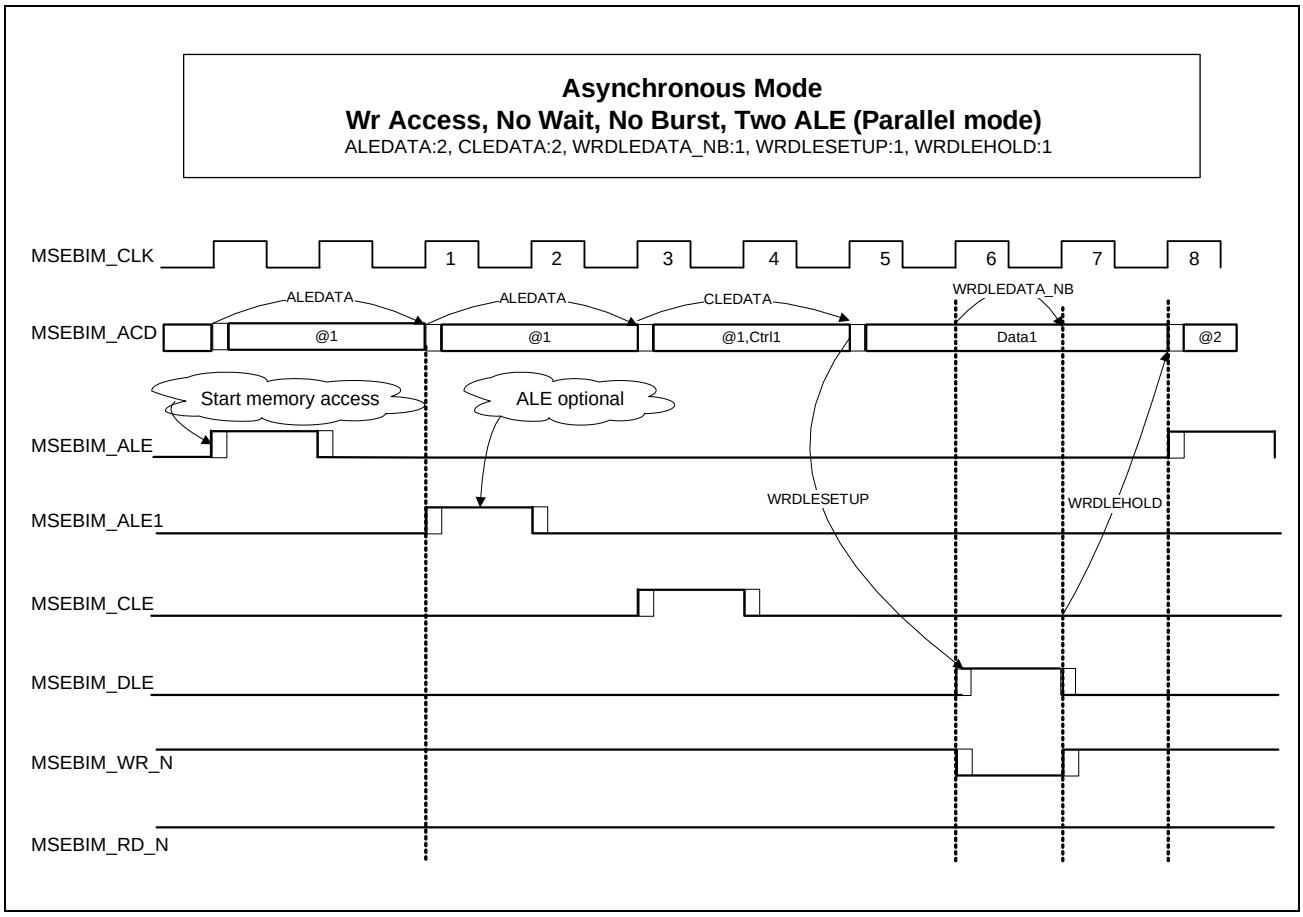

Figure 10.18 MSEBI Timing, Asynchronous Mode, Write, NoWait, NoBurst, Two ALE Parallel Mode

10.4.4.4 Synchronous Mode, No Burst, One ALE

To manage the synchronous mode, MSEBI interface must be configured with following features (n = 0..3):

- Synchronous enabled (Set bMSEBIM_CONFIG bits for master or bMSEBIS_CONFIG for slave).
- The burst mode can be disabled on master side with the bMSEBIM_BURST_ENABLE bit.
- External MSEBIM_WAIT[n]_N is generated by an external slave on the bus and is synchronous with MSEBIM_CLK clock. It is taken into account by the master during the VALID sub phase with MSEBI_CS[n]_N is low and MSEBIM_DLE is high and when RDDLEDATA_NB (read) or WRDLEDATA_NB (write) time expired.
- MSEBI master bus controls all following signals in synchronous mode:
  - MSEBIM_CLK
  - MSEBIM_ALE
  - MSEBIM_CLE
  - MSEBIM_DLE
- And set following signals to 1:
  - MSEBIM_RD_N
  - MSEBIM_WR_N
- MSEBI slave device on the bus, use the following signals:
  - MSEBIS_CLK
  - MSEBIS_ALE
  - MSEBIS_CLE
  - MSEBIS_DLE
- And generate following signals:
  - MSEBIM_WAIT[n]_N

NOTE

The timing parameter must be compliant between master and slave.

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

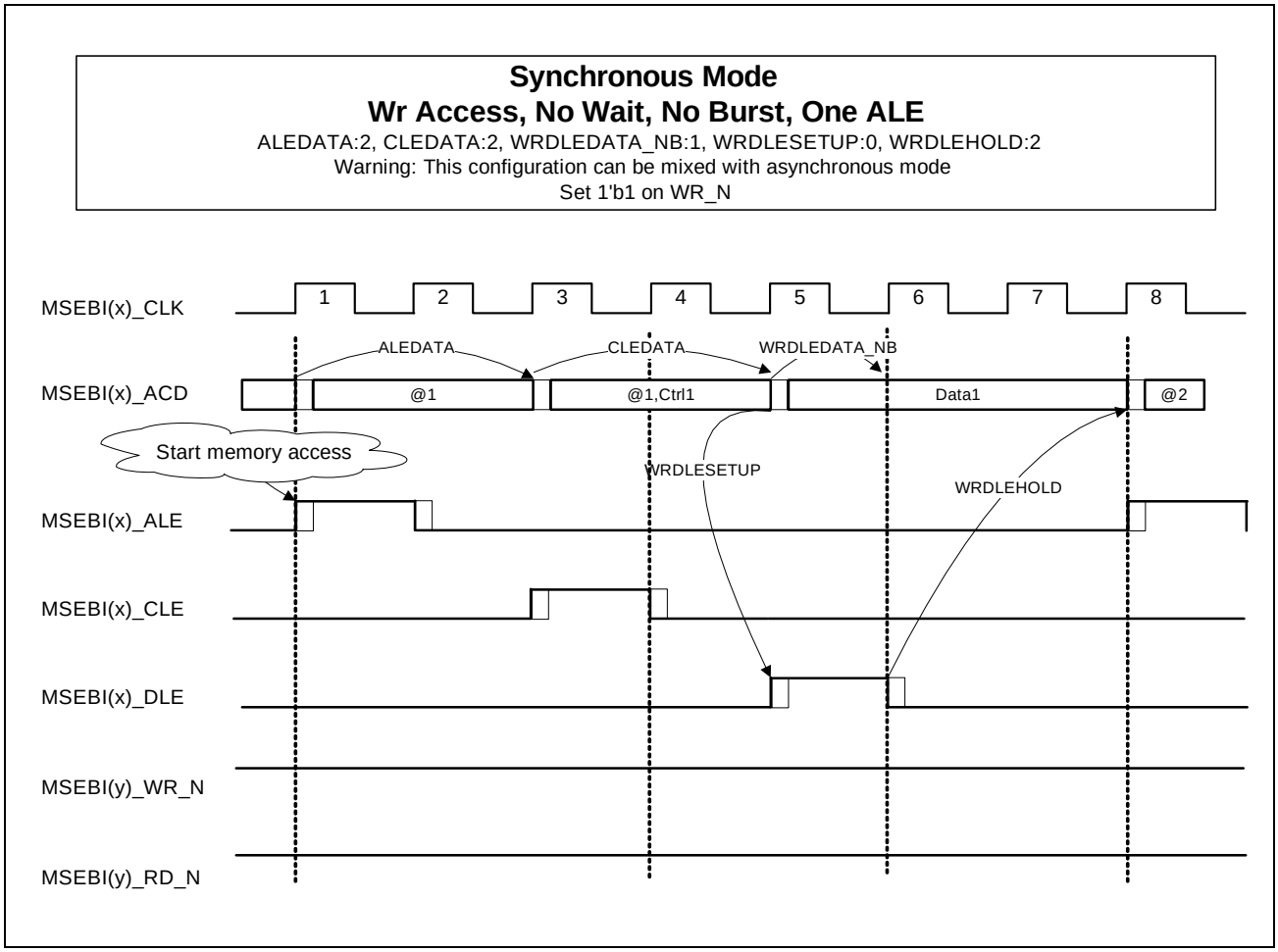

Figure 10.19 MSEBI Timing, Synchronous Mode, Write1, NoWait, NoBurst, One ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

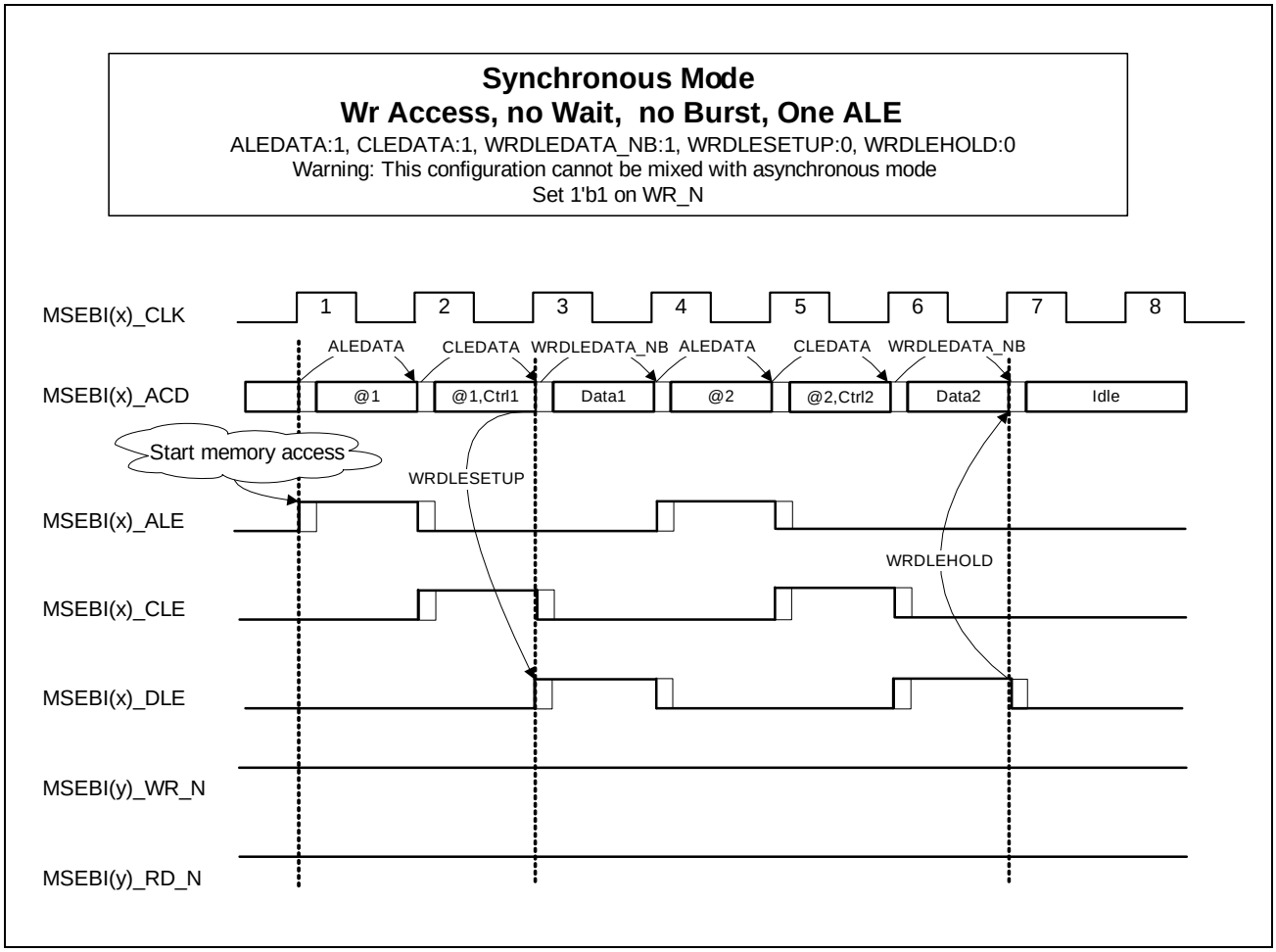

Figure 10.20 MSEBI Timing, Synchronous Mode, Write2, NoWait, NoBurst, One ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

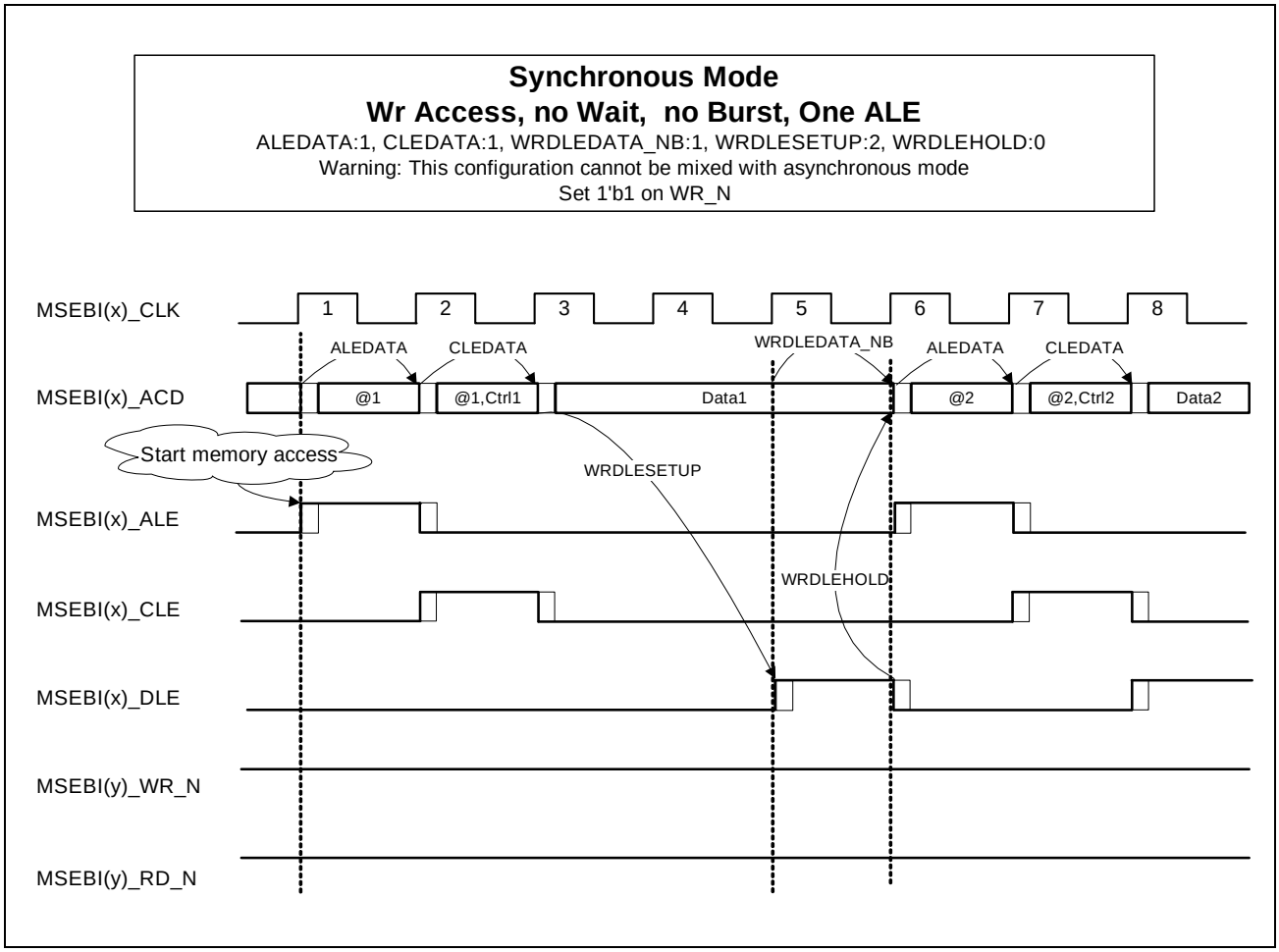

Figure 10.21 MSEBI Timing, Synchronous Mode, Write3, NoWait, NoBurst, One ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

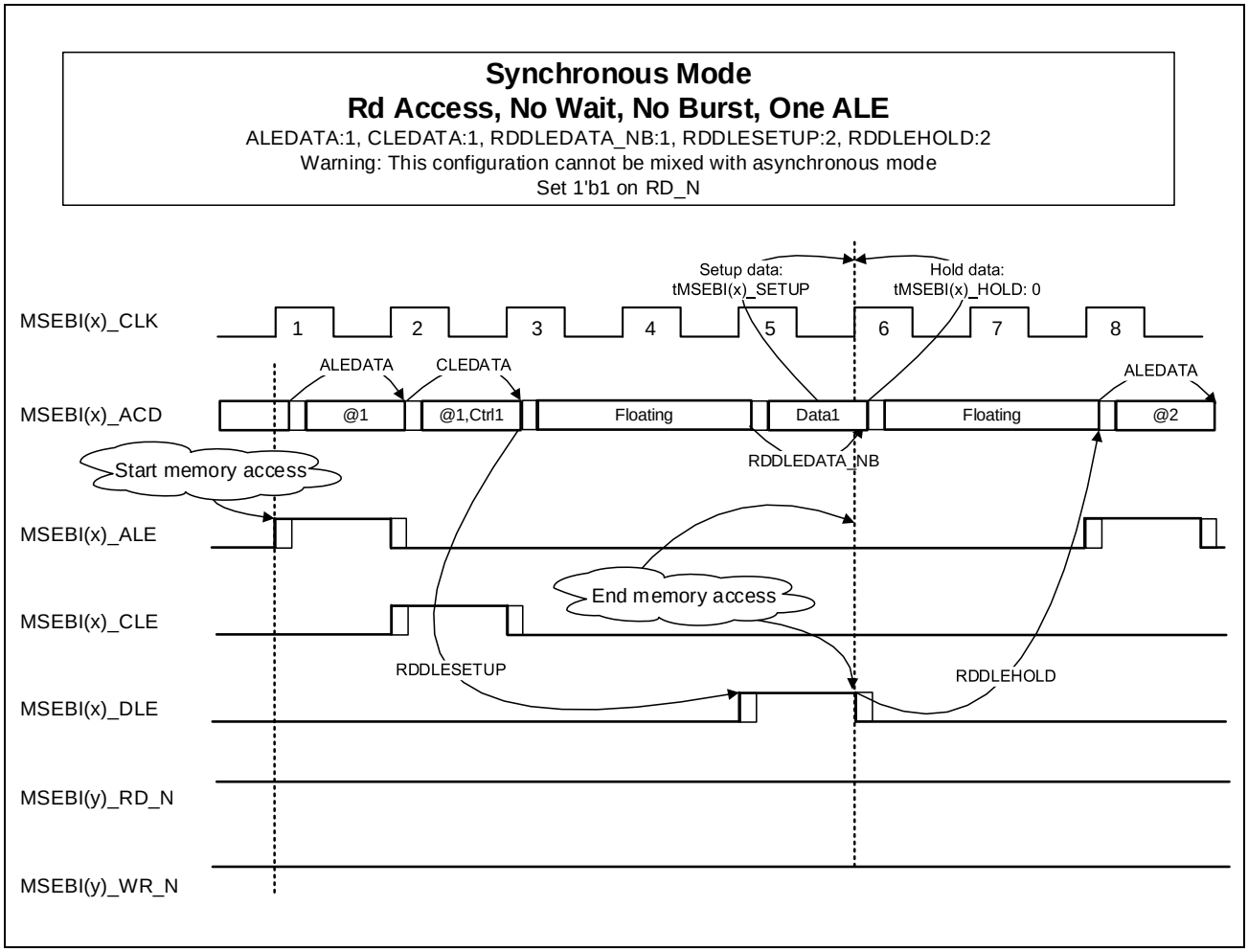

Figure 10.22 MSEBI Timing, Synchronous Mode, Read1, NoWait, NoBurst, One ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

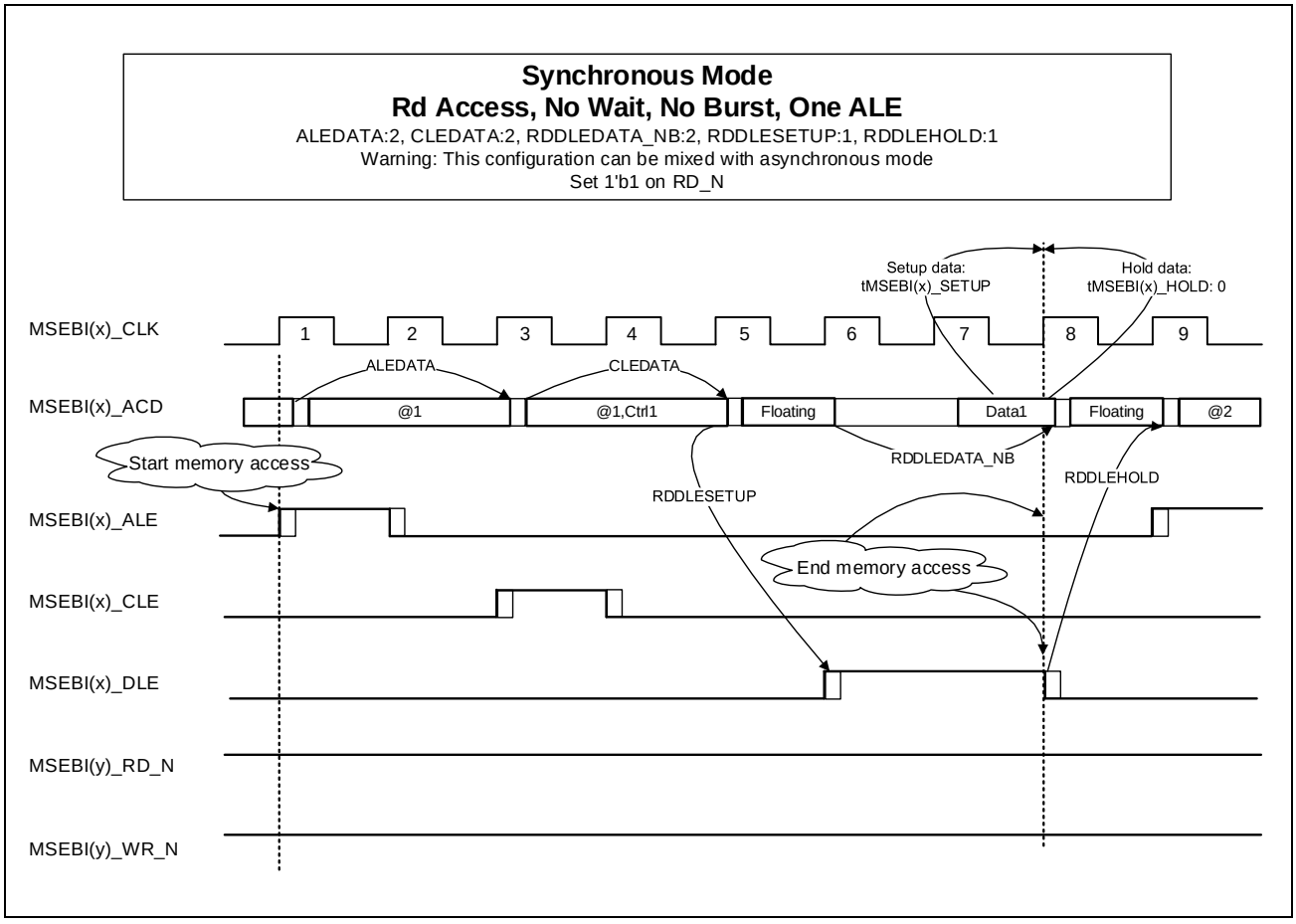

Figure 10.23 MSEBI Timing, Synchronous Mode, Read2, NoWait, NoBurst, One ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

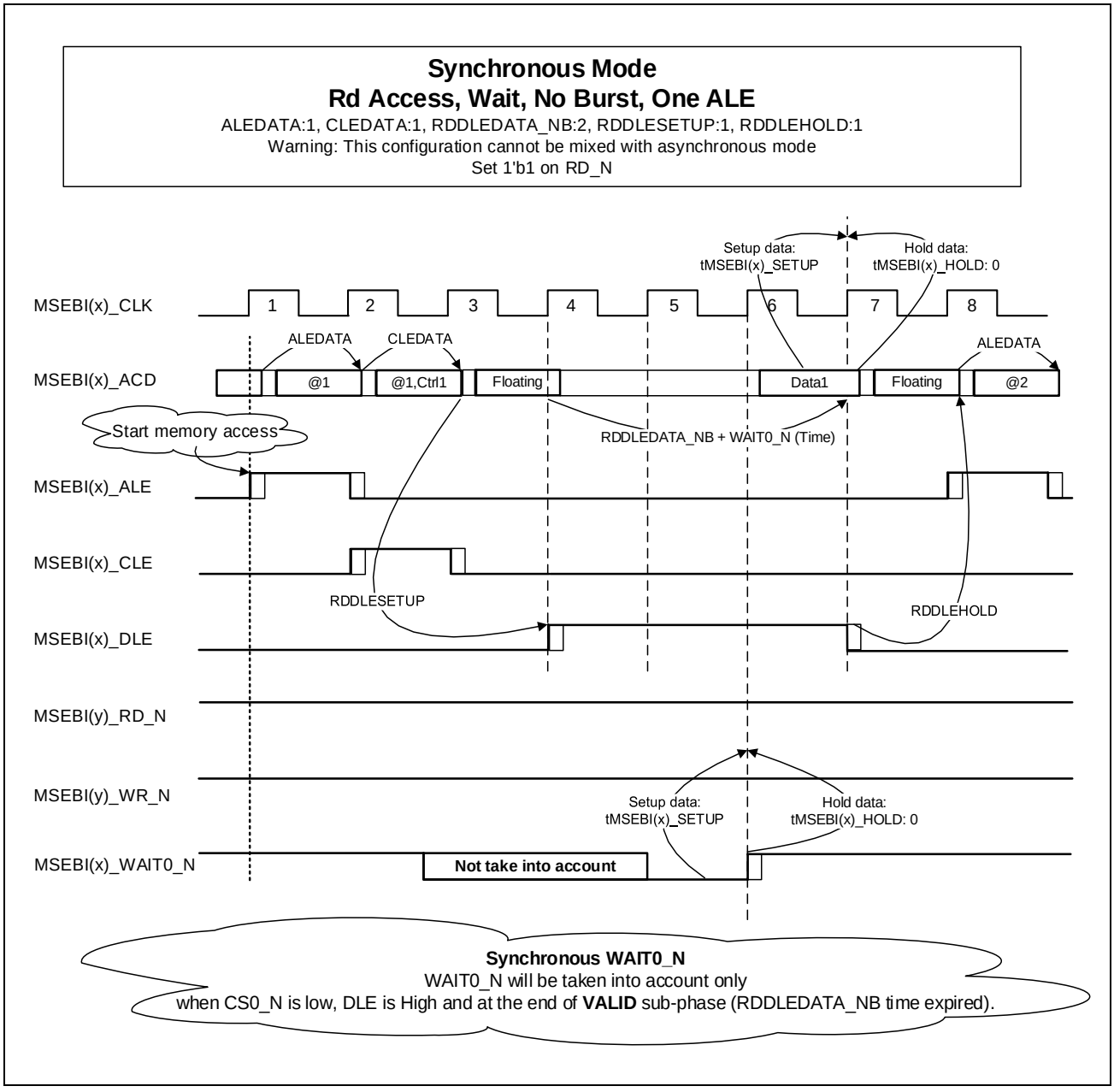

Figure 10.24 MSEBI Timing, Synchronous Mode, Read3, Wait, NoBurst, One ALE

10.4.4.5 Synchronous Mode, No Burst, No ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

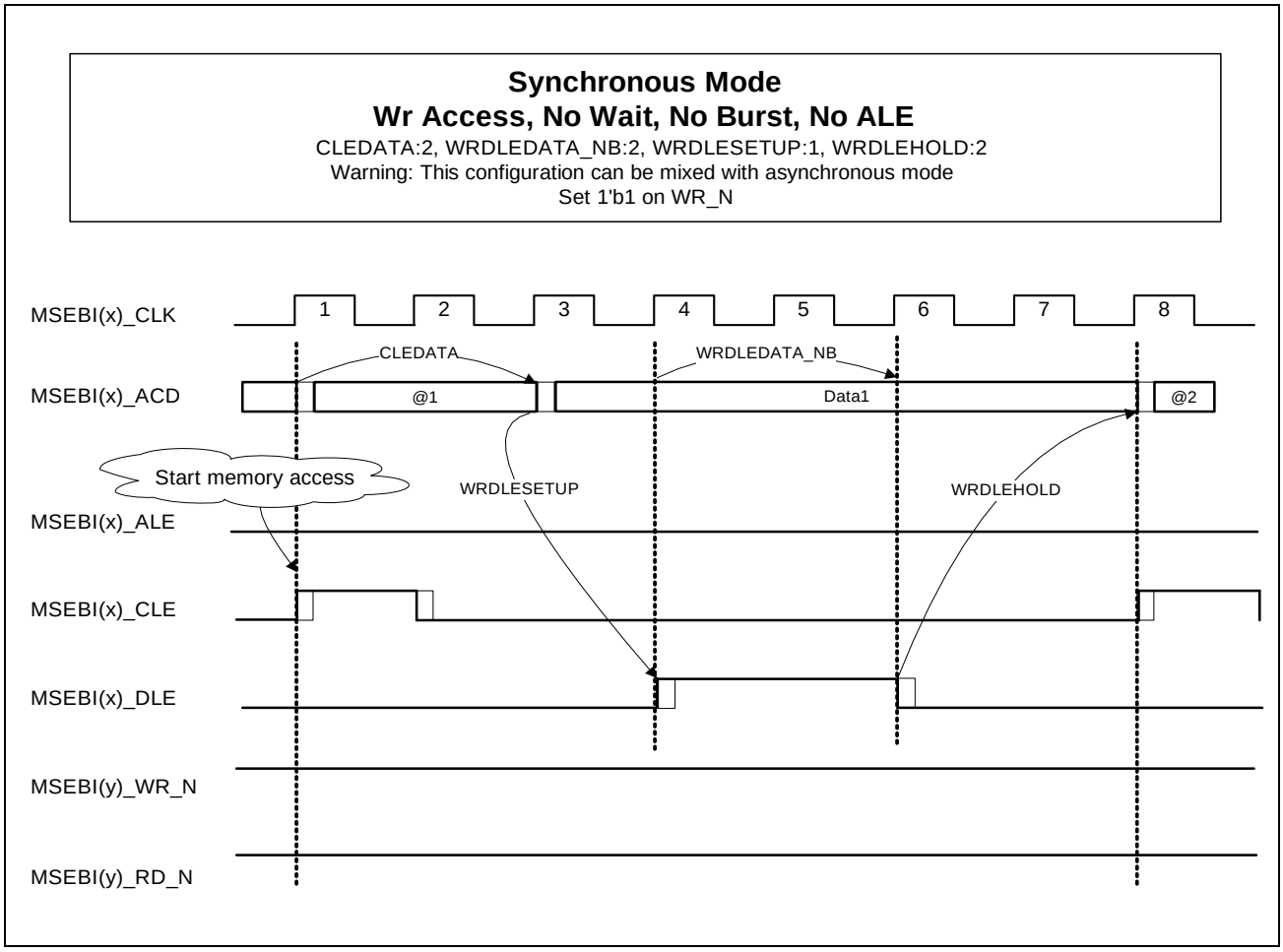

Figure 10.25 MSEBI Timing, Synchronous Mode, Write1, No Wait, NoBurst, No ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

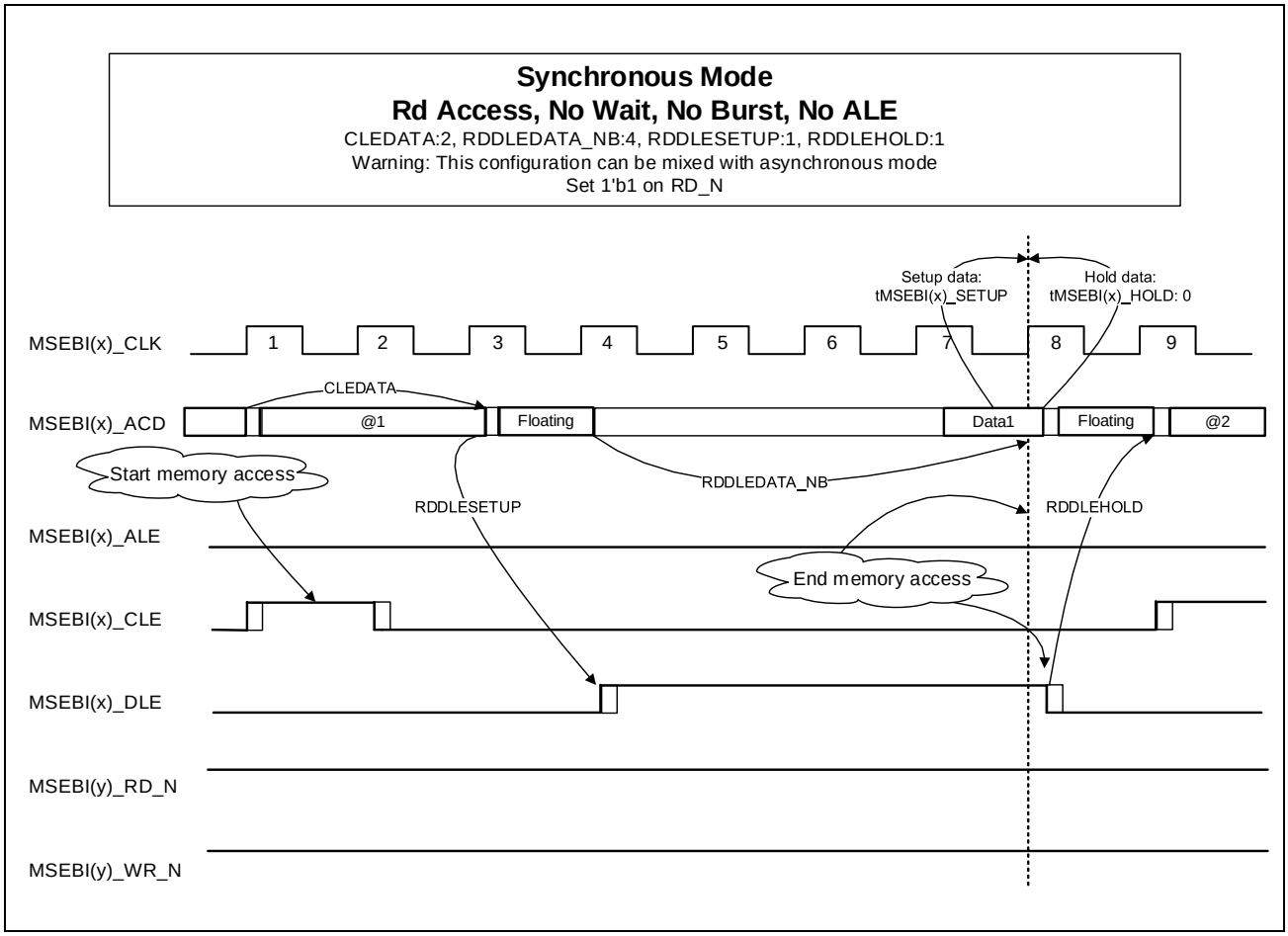

Figure 10.26 MSEBI Timing, Synchronous Mode, Read1, No Wait, NoBurst, No ALE

10.4.4.6 Synchronous Mode, No Burst, Multiple ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

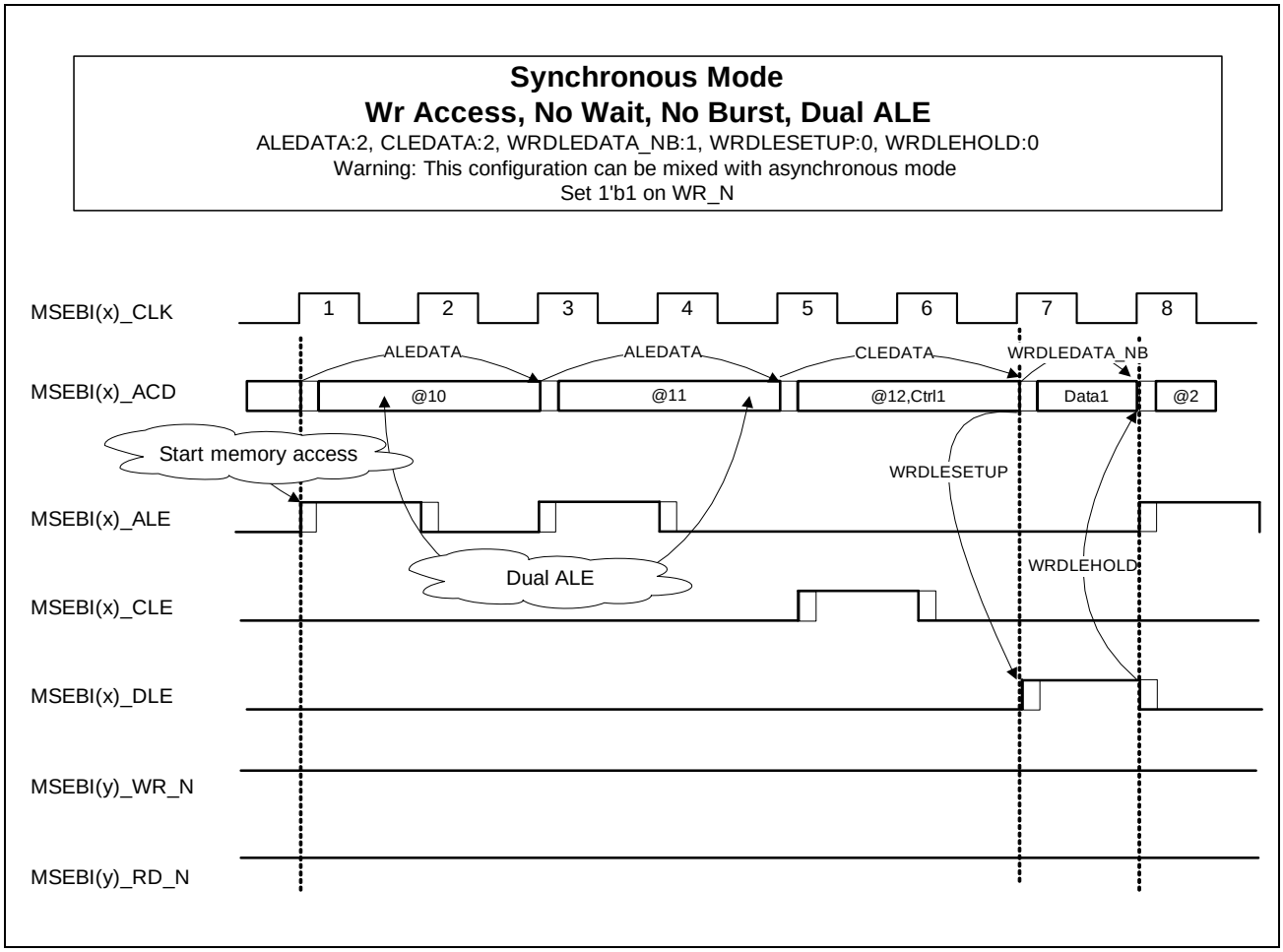

Figure 10.27 MSEBI Timing, Synchronous Mode, Write1, No Wait, NoBurst, Dual ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

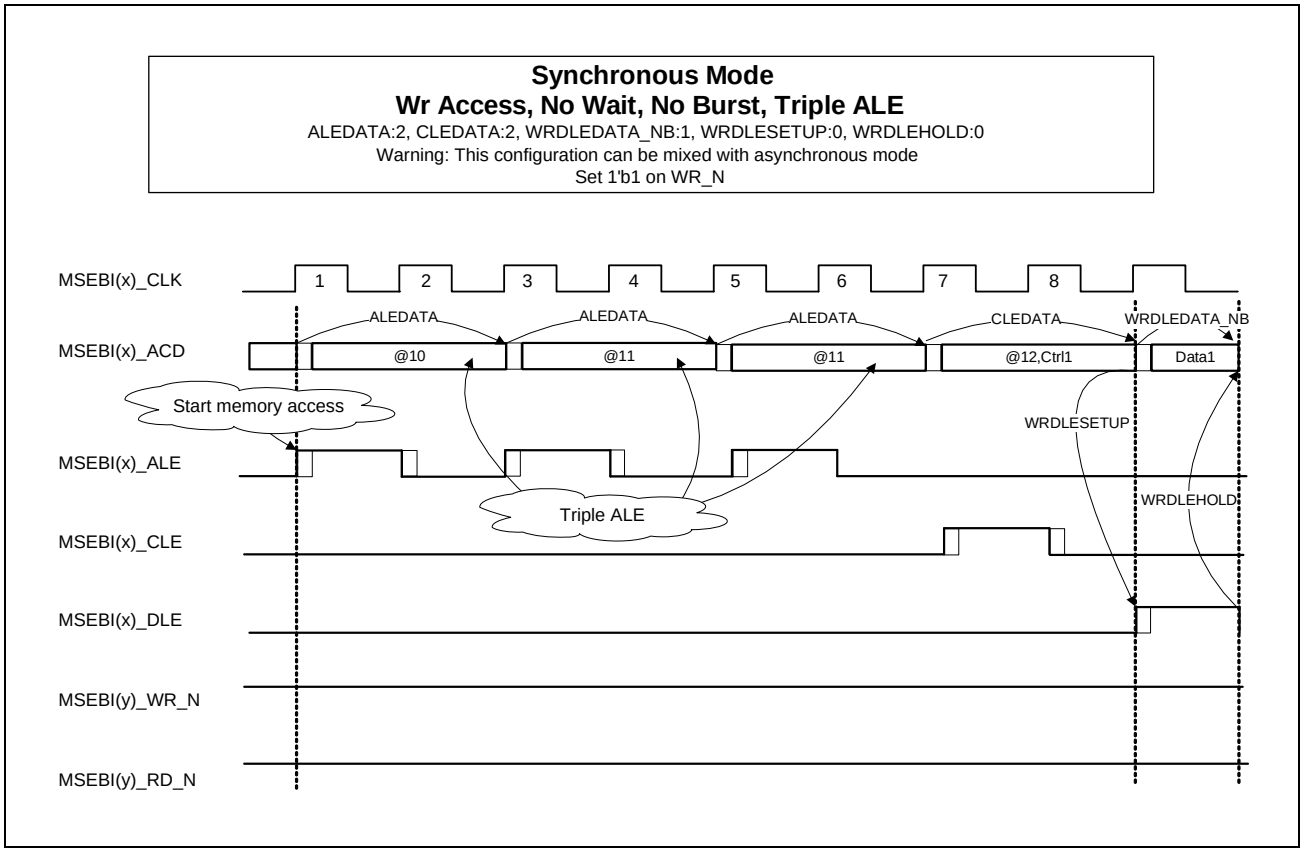

Figure 10.28 MSEBI Timing, Synchronous Mode, Write2, No Wait, NoBurst, Triple ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

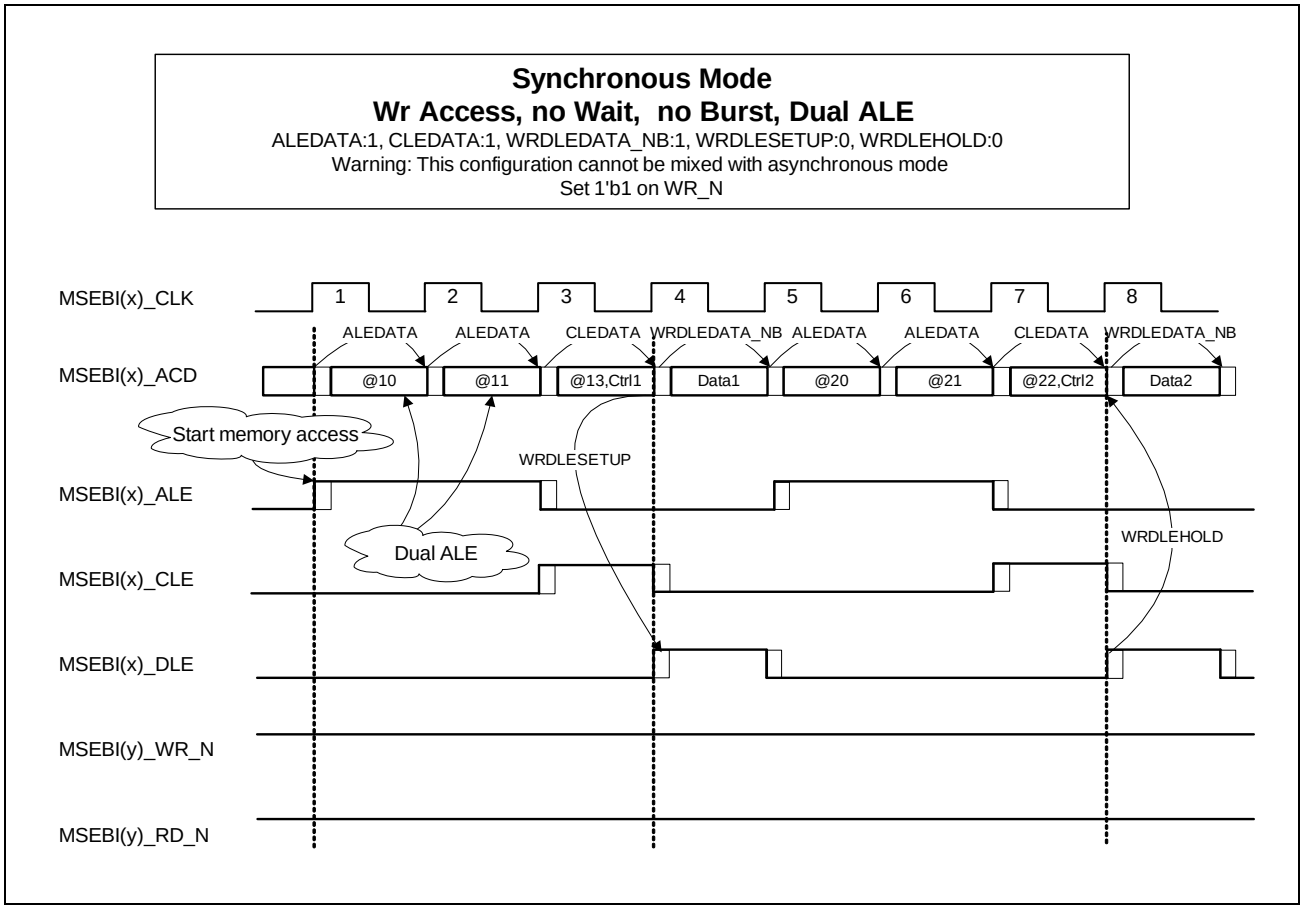

Figure 10.29 MSEBI Timing, Synchronous Mode, Write3, No Wait, NoBurst, Dual ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

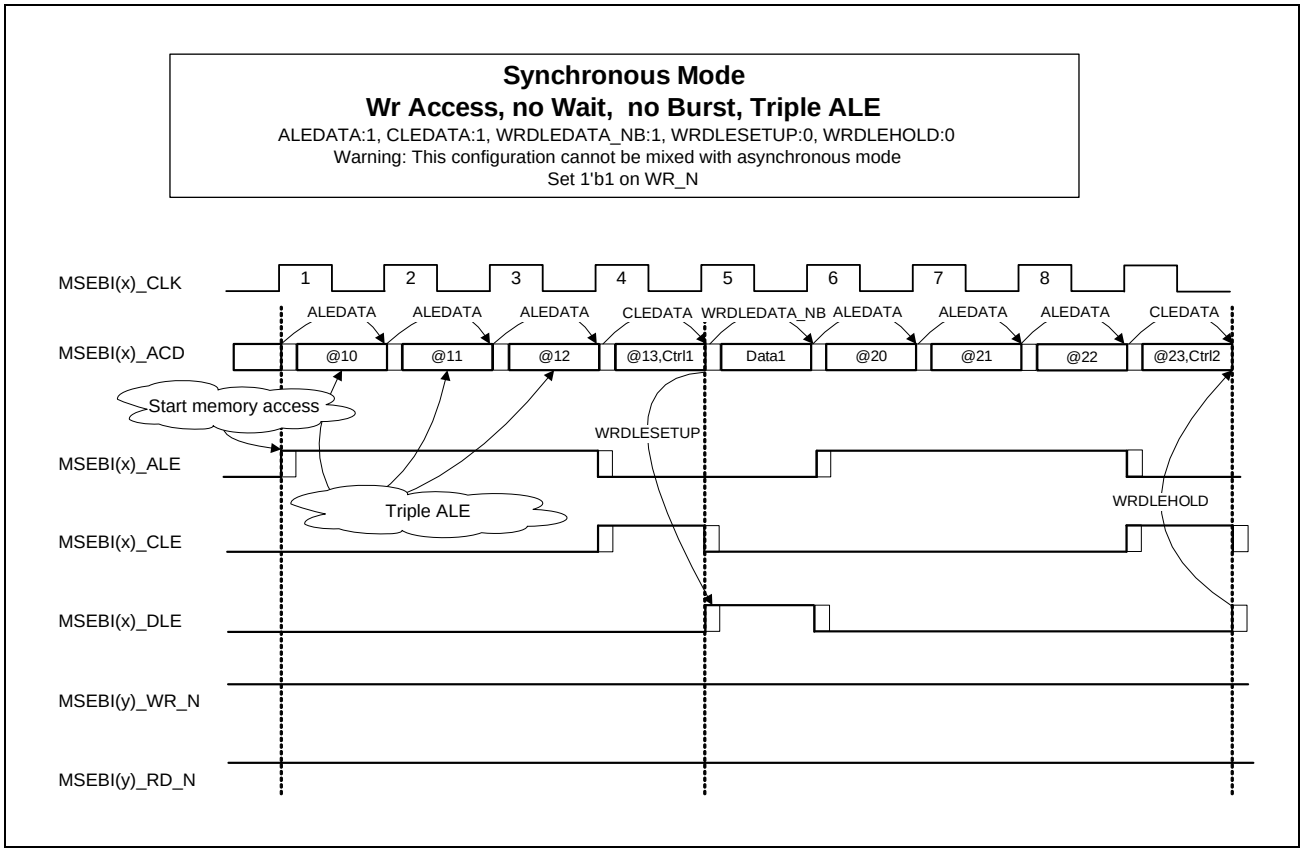

Figure 10.30 MSEBI Timing, Synchronous Mode, Write4, No Wait, NoBurst, Triple ALE

10.4.4.7 Synchronous Mode, Burst, One ALE

To manage the synchronous mode, MSEBI interface must be configured with following features (n = 0..3):

- Synchronous enable (Set bMSEBIM_CONFIG bits for master or bMSEBIS_CONFIG for slave).
- The burst mode is enabled on master side by the bMSEBIM_BURST_ENABLE bit.
- External MSEBIM_WAIT[n]_N is generated by an external slave on the bus and is synchronous with MSEBIM_CLK clock. It is taken into account by the master when MSEBI_CS[n]_N is low and MSEBIM_DLE is high, and at the end of VALID sub phase when RDDLEDATA_NB (read) or WRDLEDATA_NB (write) time expired on the first access of burst access, or RDDLEDATA_B (read) or WRDLEDATA_B (write) time expired on all burst access after the first access.
- MSEBI master bus controls all following signals in synchronous mode:
  - MSEBIM_CLK
  - MSEBIM_ALE
  - MSEBIM_CLE
  - MSEBIM_DLE
- And set following signals to 1:
  - MSEBIM_RD_N
  - MSEBIM_WR_N
- MSEBI slave device on the bus, use the following signals:
  - MSEBIS_CLK
  - MSEBIS_ALE
  - MSEBIS_CLE
  - MSEBIS_DLE
- And generate following signals:
  - MSEBIM_WAIT[n]_N

NOTE

The timing parameter must be compliant between master and slave.

- The max burst size on MSEBI bus is configured on master side by
  - CPU: bMSEBIM_BURST_SIZEMAX_CPUREAD and bMSEBIM_BURST_SIZEMAX_CPUWRITE bits
  - DMA: bMSEBIM_BURST_SIZEMAX_DMAREAD and bMSEBIM_BURST_SIZEMAX_DMAWRITE bits

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

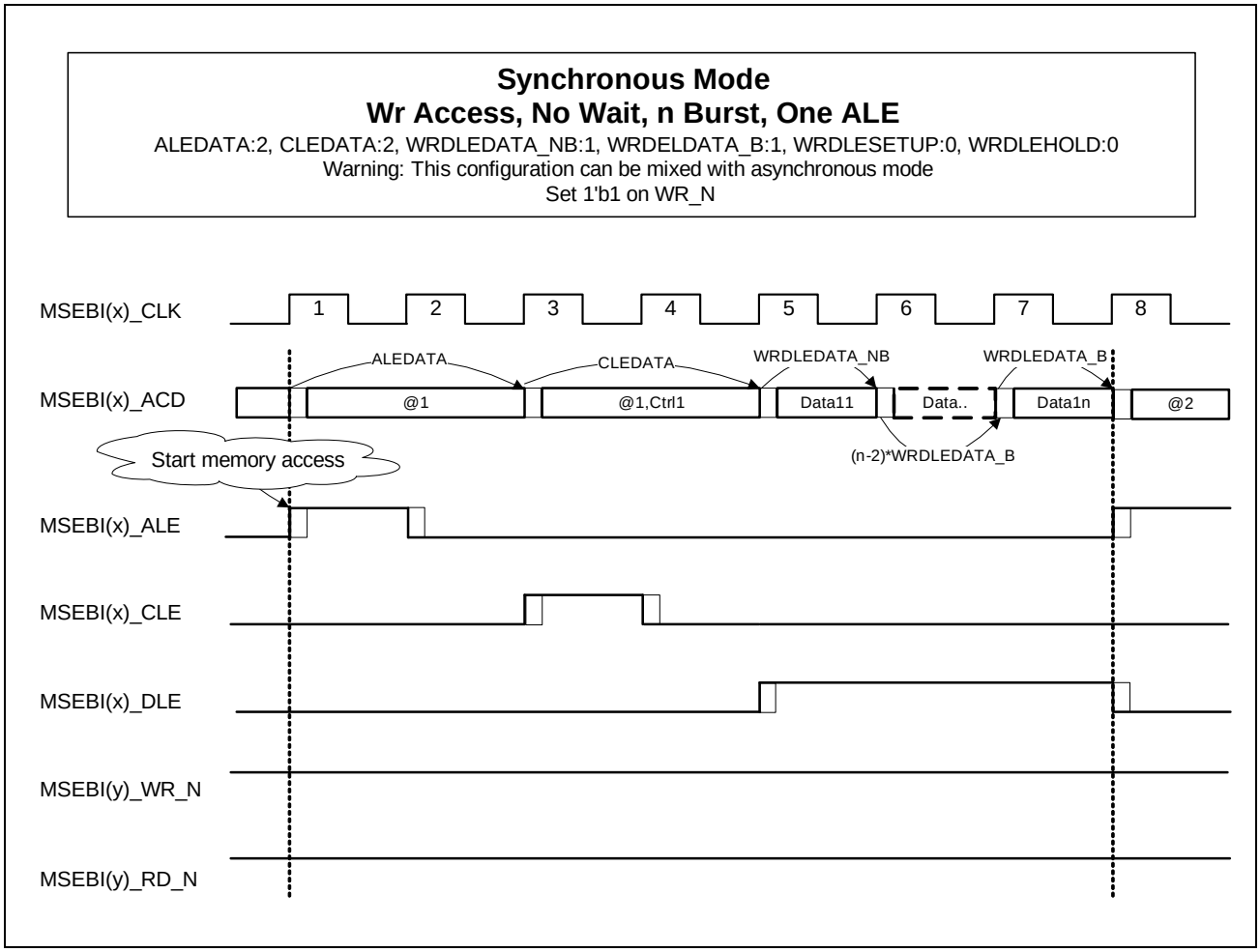

Figure 10.31 MSEBI Timing, Synchronous Mode, Write1, NoWait, n Burst, One ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

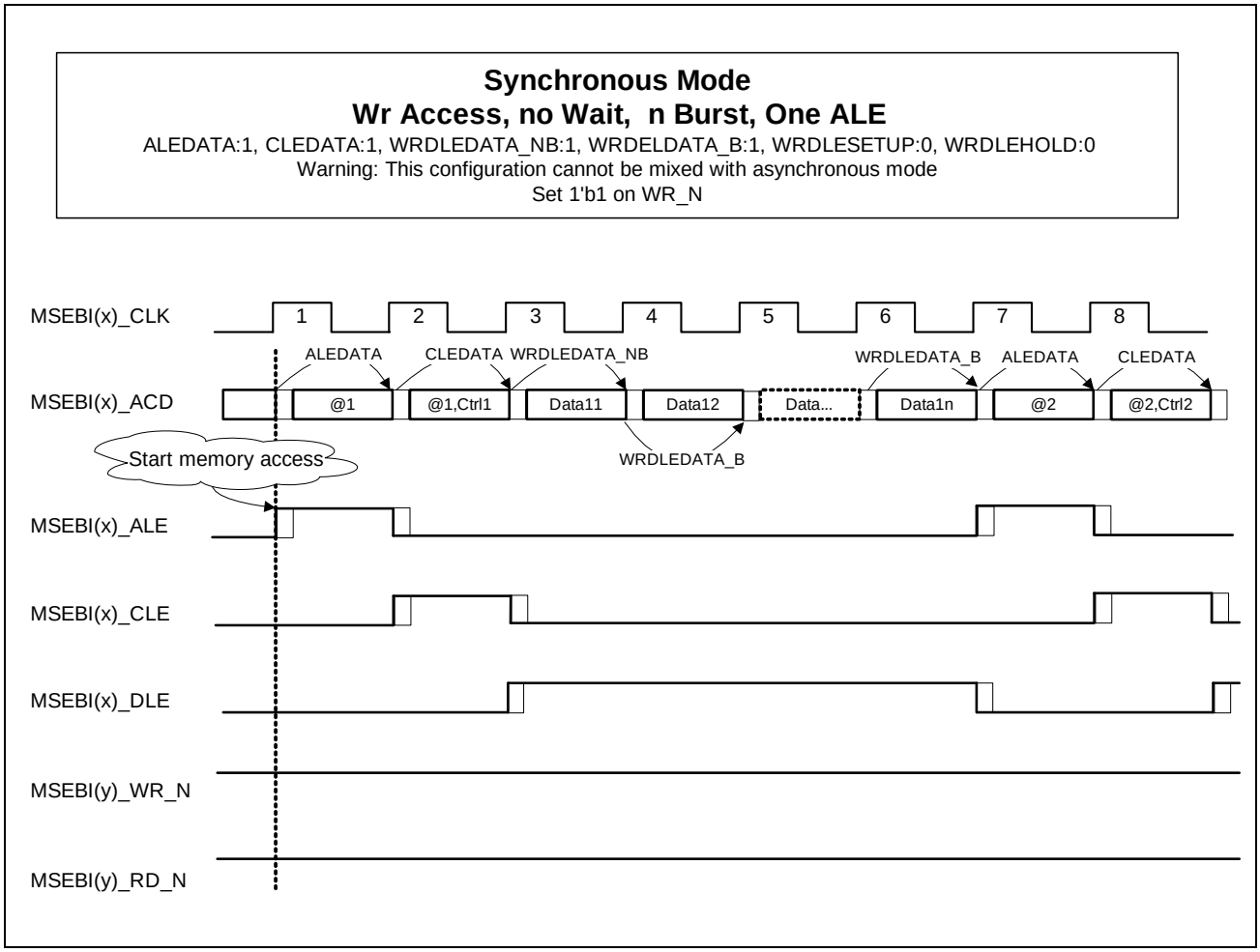

Figure 10.32 MSEBI Timing, Synchronous Mode, Write2, NoWait, n Burst, One ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

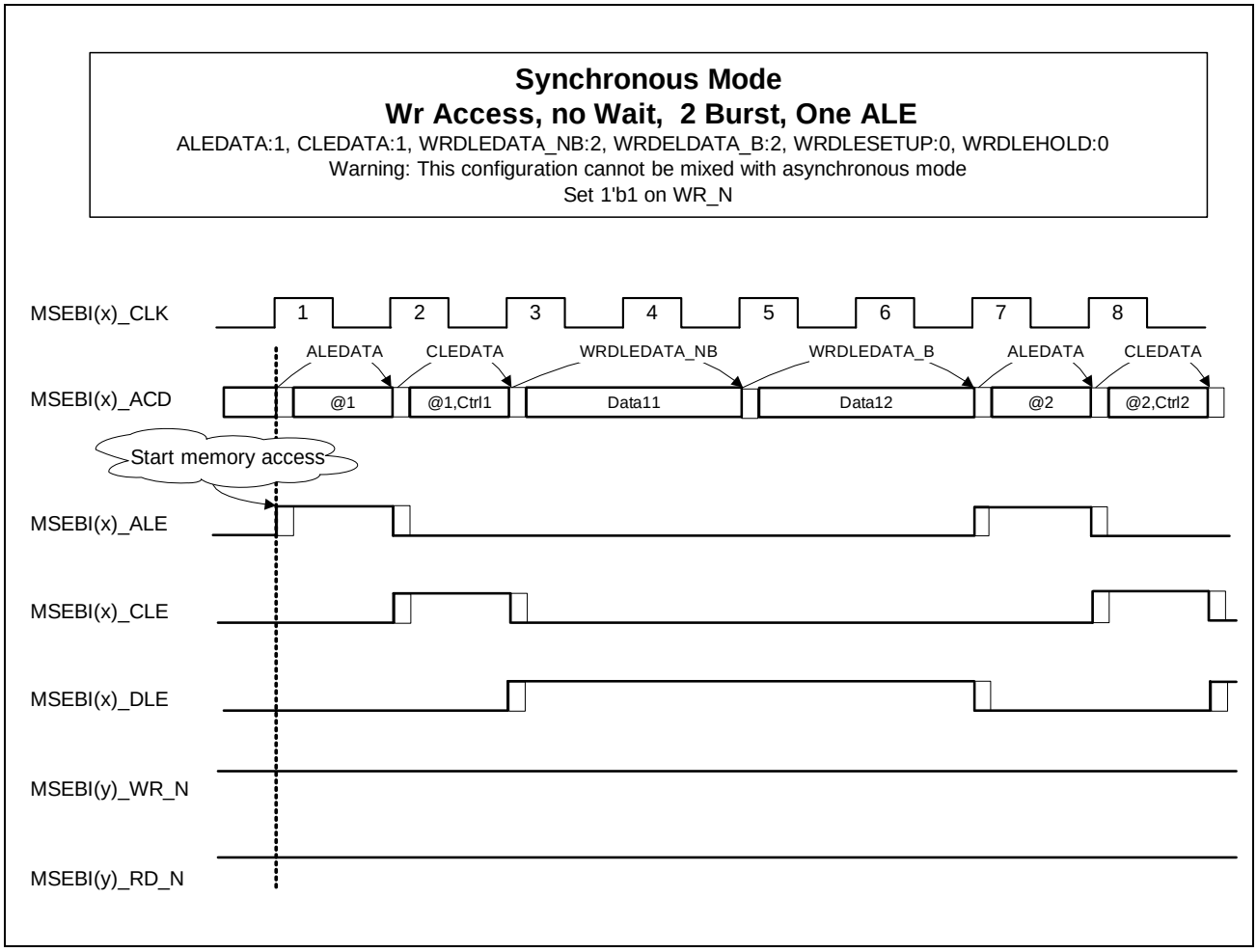

Figure 10.33 MSEBI Timing, Synchronous Mode, Write3, NoWait, 2 Burst, One ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

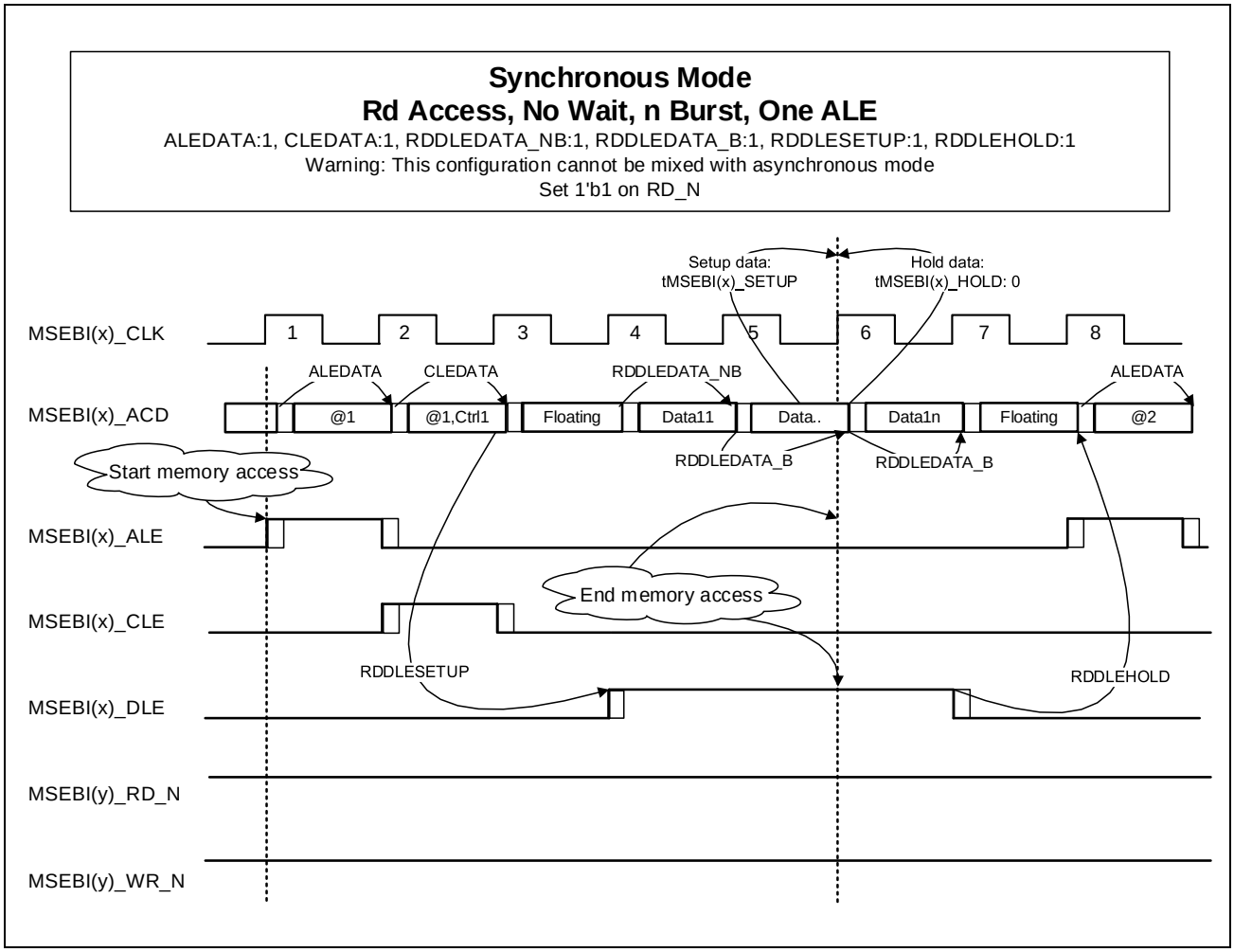

Figure 10.34 MSEBI Timing, Synchronous Mode, Read1, NoWait, n Burst

10.4.4.8 Synchronous Mode, Burst, No ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

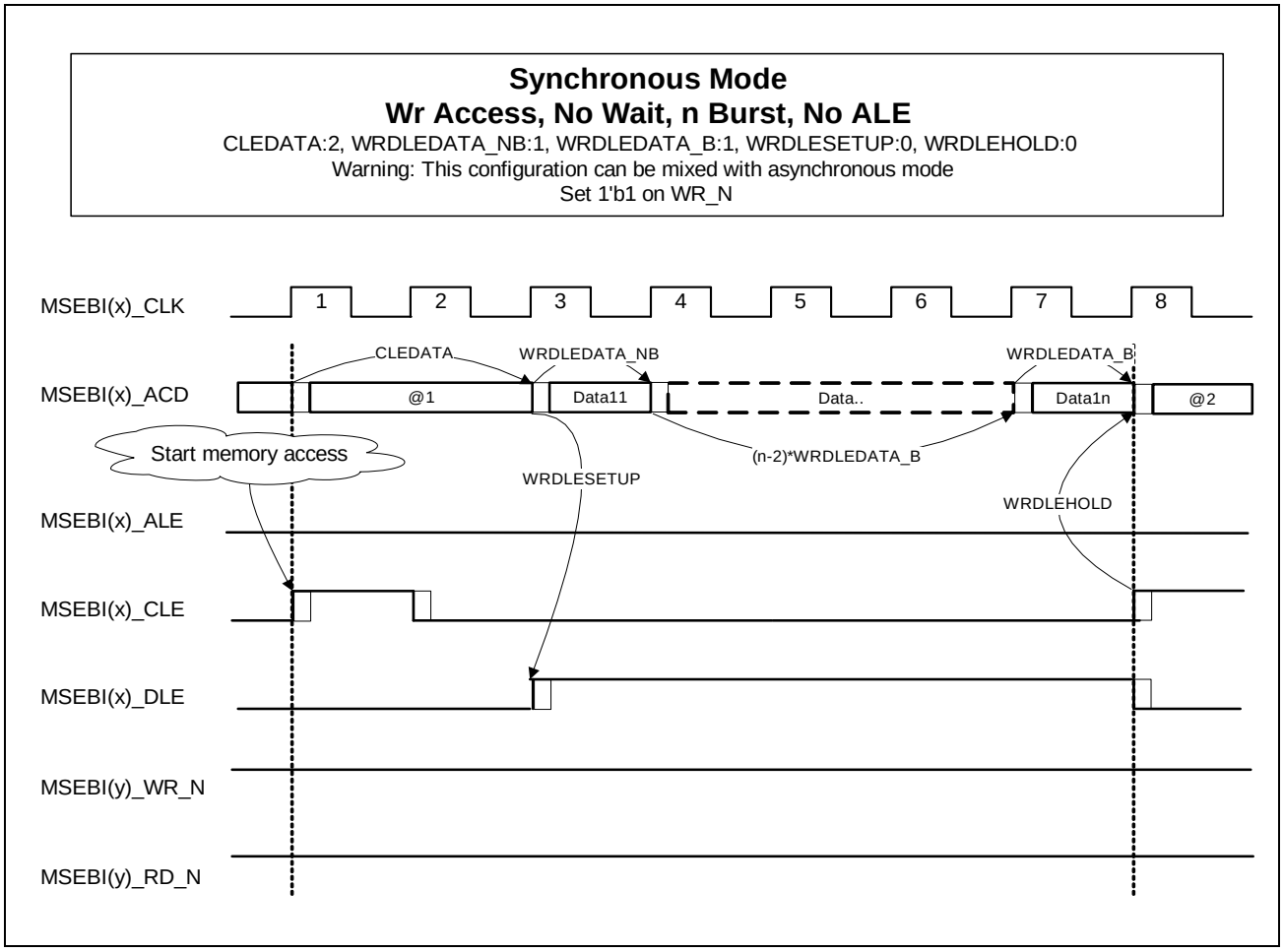

Figure 10.35 MSEBI Timing, Synchronous Mode, Write1, NoWait, Burst, No ALE

On signal name:

- MSEBI(x) represents MSEBIM for master interface
- MSEBI(x) represents MSEBIS for slave interface
- MSEBI(y) represents only MSEBIM for Master interface

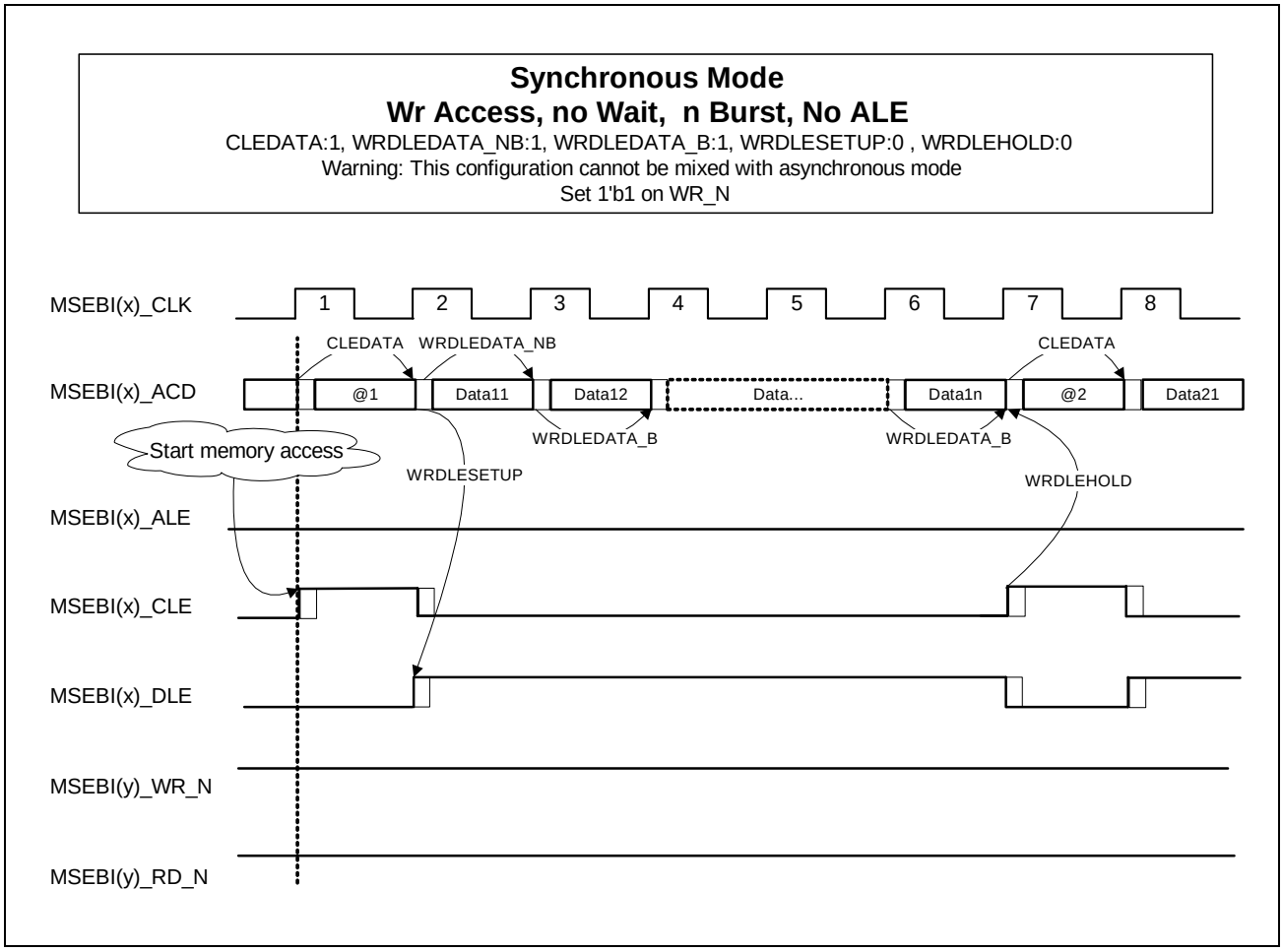

Figure 10.36 MSEBI Timing, Synchronous Mode, Write2, NoWait, Burst, No ALE

10.4.5 MSEBI Interrupt

10.4.5.1 MSEBI Interrupt: Overview

MSEBI provides one interrupt line: MSEBIS_Int.

This interrupt is used for “end of block event” detection by the slave.

See:

- Interrupt is set by register (access by CPU of the master through MSEBI).
  - **Section 10.3.4.1, rMSEBIS_INT — Slave Interrupt Register**.
- Status of interrupt is read by register (access by CPU).
  - **Section 10.3.3.11, rMSEBIS_STATUS_INT0 — Interrupt Status Register**.
  - **Section 10.3.3.12, rMSEBIS_STATUS_INT1 — Masked Interrupt Status Register**.
- Interrupt can be masked by register (access by CPU).
  - **Section 10.3.3.13, rMSEBIS_MASK_INT — Interrupt Mask Register**.
- Interrupt is cleared by register (access by CPU).
  - **Section 10.3.3.14, rMSEBIS_CLR_INT — Interrupt Clear Register**.
- Contains the address of the descriptor used to complete the write transfer in memory (access by CPU).
  - **Section 10.3.3.15, rMSEBIS_EOB_ADDR — End Of Block Address Register**.
  - CPU MSEBI_CS[n]_N (n = 0..3) and DMA MSEBI_CS[n]_N (n = 0..1) descriptors addresses are automatically computed (please refer to the register description).

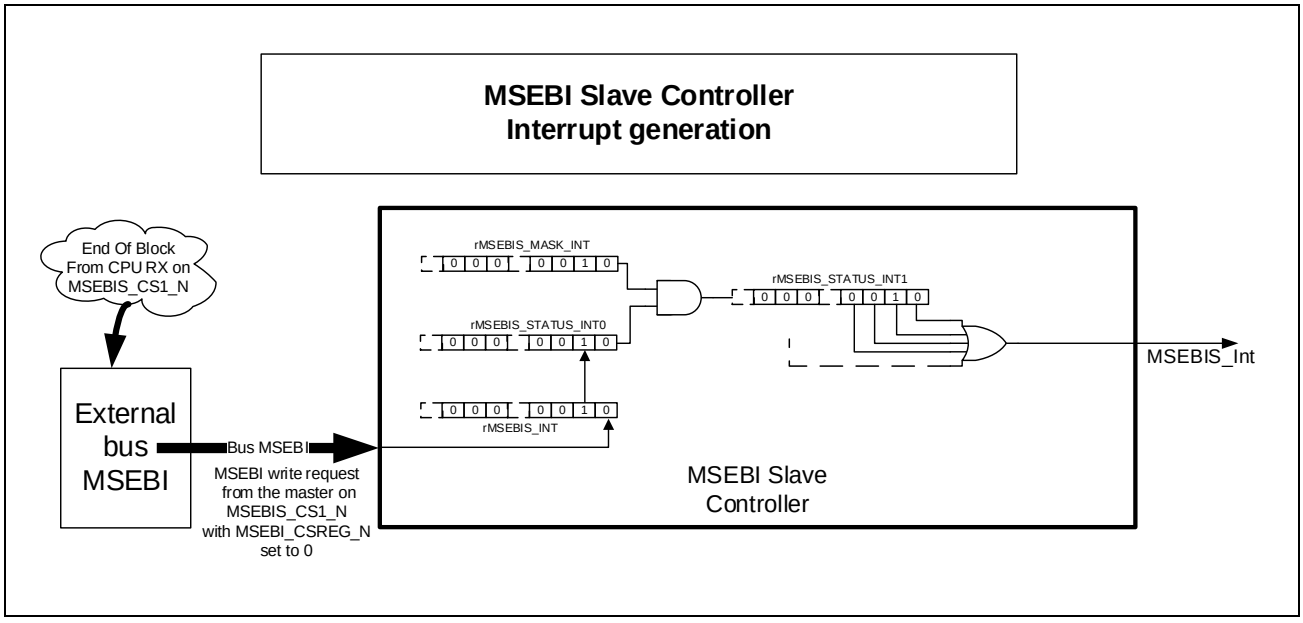

Figure 10.37 MSEBI Slave Interrupt Generation

10.4.5.2 MSEBI Interrupt: End of Block Detection by the Master

The section below provides a way to manage the end of block detection on master side.

For a transfer using DMA TX[n] FIFO (n = 0..1)

- At the end of block transfer with DMA, DMA can generate an interrupt to indicate that the last data of the transfer has been pushed on the DMA TX[n] FIFO.
  - See **Figure 10.48, MSEBI: Burst Mode, DMA Transmit FIFO and Bus Interface Coupling**.
- CPU manages the interrupt by polling the bMSEBIM_TDMAE1 which is automatically cleared after all data in the FIFO has been sent to the MSEBI bus and bMSEBIM_DEST_BLOCK_SIZE single elements have been sent.
  - See **Section 10.4.6.3, MSEBI Master: DMA Control**.

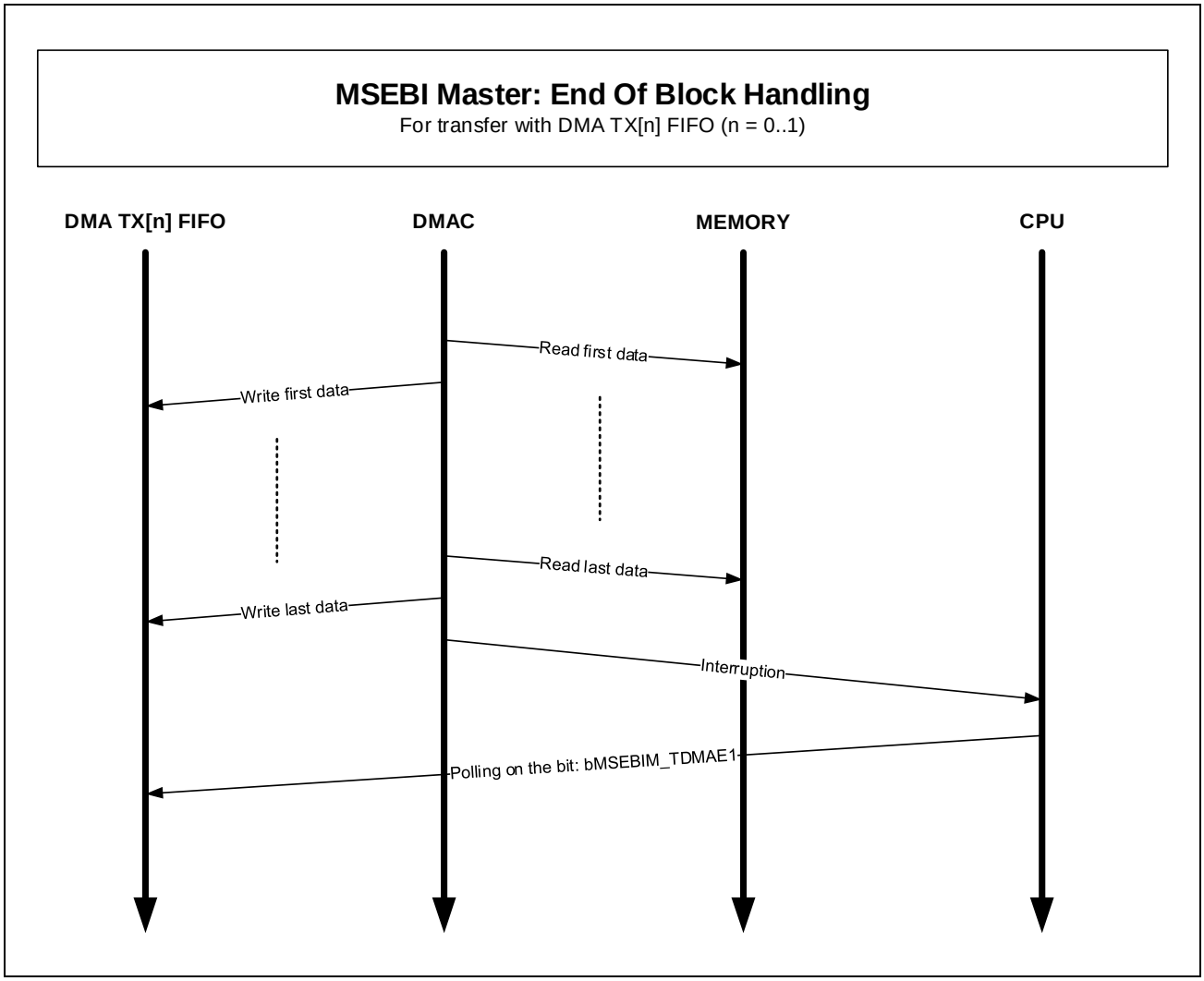

Figure 10.38 MSEBI Master: End of Block Transfer with DMA TX

For a transfer using DMA RX[n] FIFO (n = 0..1)

- At the end of block transfer with DMA, DMA can generate an interrupt to indicate that the last read data of the transfer has been sent to the memory.
  - See **Figure 10.49, MSEBI: Burst Mode, DMA Receive FIFO and Bus Interface Coupling**.
- CPU must ensure that all the data are really available on the memory (not stored in the write buffer) before using them. User may use the write back feature of the DMA (ch = 0..7).

  Write back feature is a mechanism where a status of the DMA transfer (DMAC.CTL[ch].DONE) is copied by the DMA to the memory at the end of a block transfer. The copy is done on a structure called LLI (linked list item) at a location pointed by the DMAC.LLP[ch].

  The CPU polls the location of the status on memory until the LLI DMAC.CTL[ch].DONE bit is set to 1. The transfer is guaranteed to be completed because:
  1) The order of write access (from a specific initiator) is guaranteed on the NoC
  2) The write back command is sent by the DMA after the last write of the transfer
  - See **Section 10.4.6.3, MSEBI Master: DMA Control**.

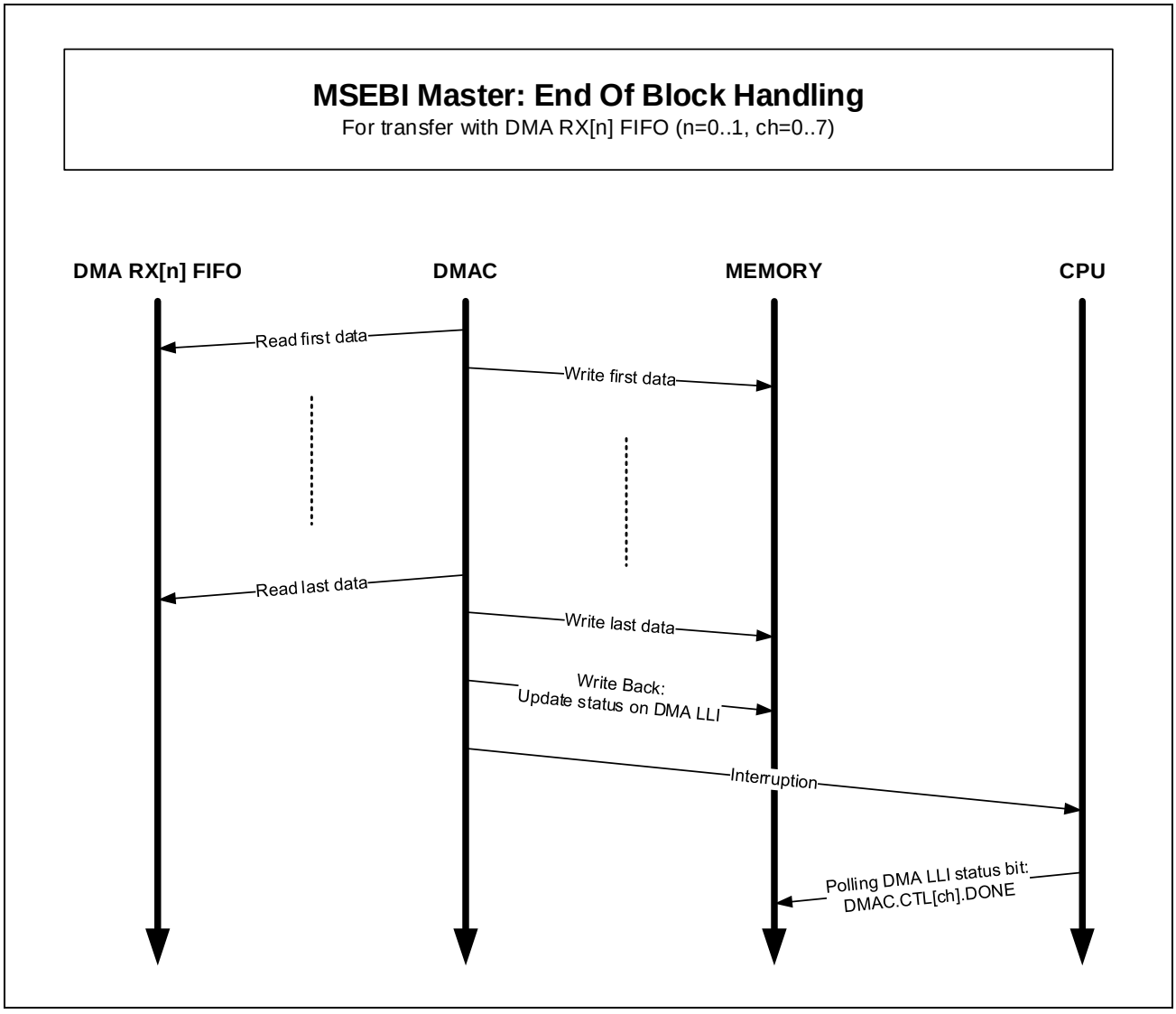

Figure 10.39 MSEBI Master: End of Block Transfer with DMA RX

For a write transfer using CPU

- At the end of block transfer with CPU, the end of the last write on MSEBI controller indicates that the command has been pushed on the CPU transmit FIFO.
  - See **Figure 10.46, MSEBI: Burst Mode, CPU Transmit and Receive FIFO and Bus Interface Coupling, Example1** and **Figure 10.47, MSEBI: Burst Mode, CPU Transmit and Receive FIFO and Bus Interface Coupling, Example2**.
- CPU manages the end of block to ensure that the command sent on the bus.
  By polling the FIFO level bit (bMSEBIM_CPU_TRANSMIT_FIFOLEVEL).

  Or by reading a status register in MSEBI Slave

  1) Read and write transfers are executed in the order in which they are received.
  2) In order to force a write in MSEBI slave, it is necessary to perform a read because a read will drain the write and read commands out of the CPU transmit FIFO.

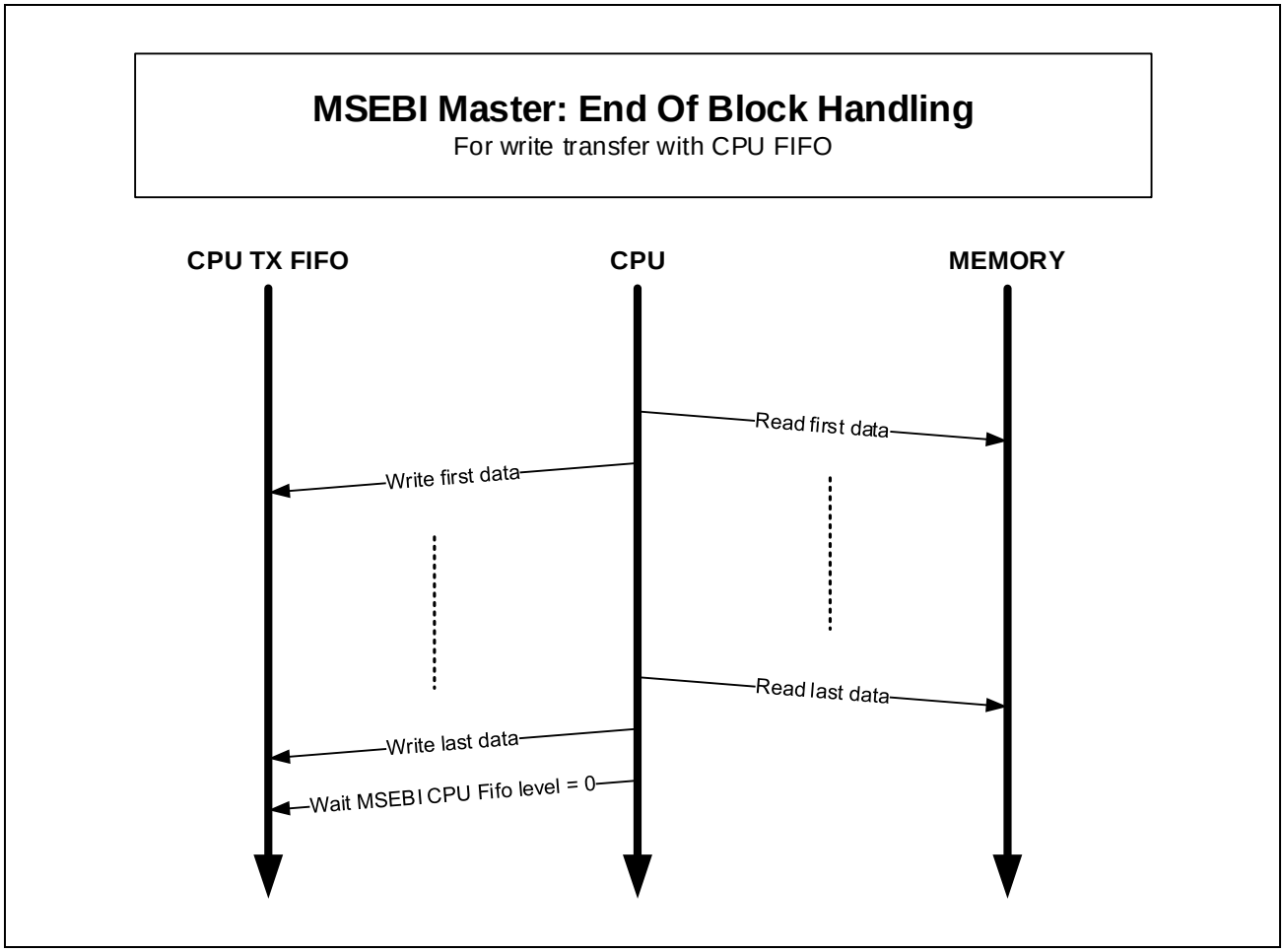

Figure 10.40 MSEBI Master: End of Write Block Transfer with CPU FIFO

For a Read transfer using CPU

- At the end of block transfer with CPU, the read data is copied by the CPU to the memory. No specific action is required.
  - See **Figure 10.46, MSEBI: Burst Mode, CPU Transmit and Receive FIFO and Bus Interface Coupling, Example1** and **Figure 10.47, MSEBI: Burst Mode, CPU Transmit and Receive FIFO and Bus Interface Coupling, Example2**.

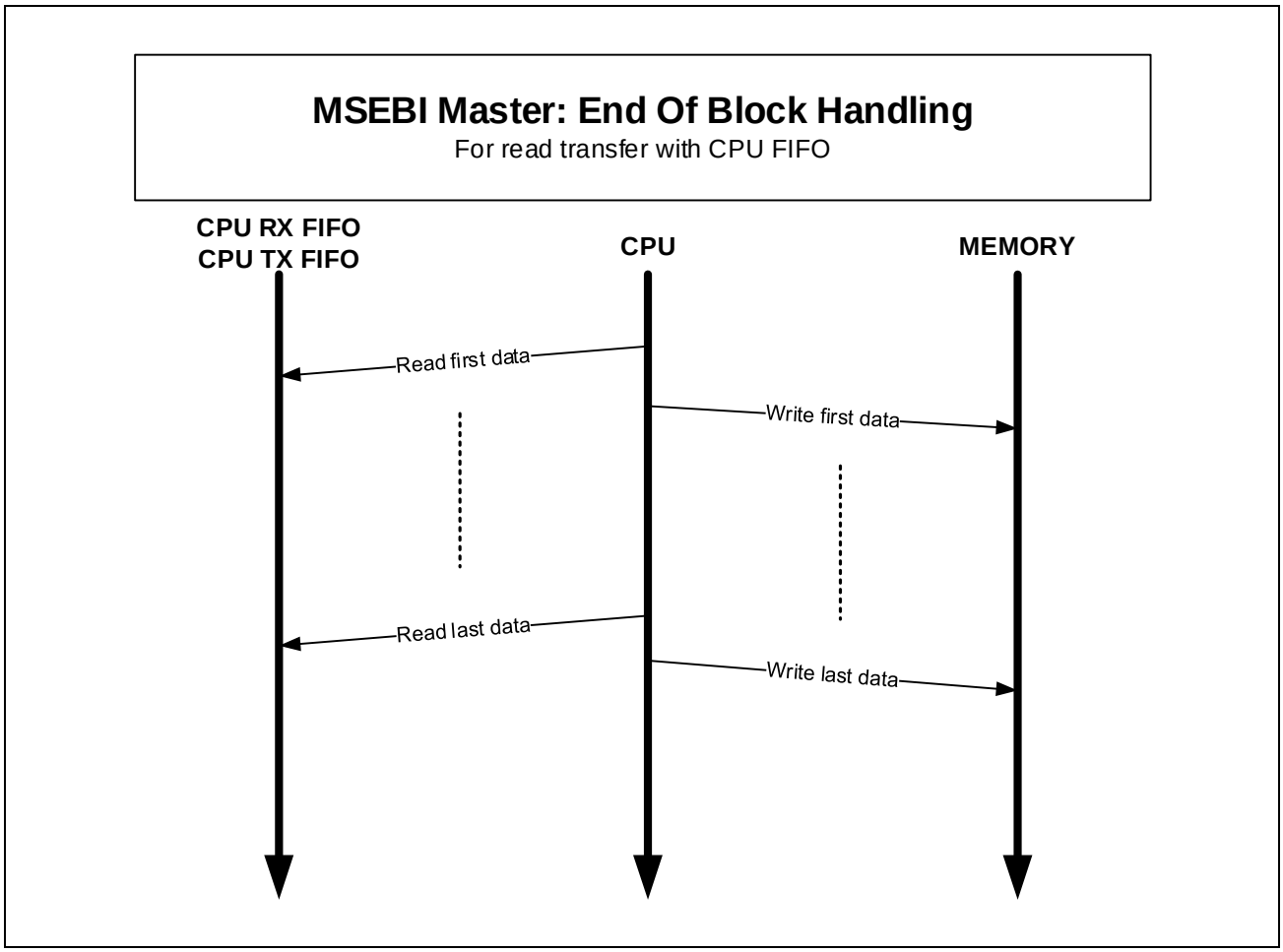

Figure 10.41 MSEBI Master: End of Read Block Transfer with CPU FIFO

10.4.5.3 MSEBI Interrupt: End of Block Detection by the Slave

After a block transfer between an MSEBI master and an MSEBI slave device, slave device may be informed of the block completion.

The section below presents the sequence that allows the slave to detect the end of a write block transfer (n = 0..1):

- MSEBI master transfers a block of data to the slave with CPU or DMA part of the MSEBI master controller.
  - See following figures:
    Figure 10.58, MSEBI Slave CPU FIFOs Example 1.
    Figure 10.61, Slave DMA FIFOs for Requests from Master MSEBI DMA TX FIFOs.
- At the end of the transfer, the firmware on master side must ensure that the block has been totally written on the slave memory:
  - Firmware on master side sends an “end of block event” to the slave by writing bMSEBIS_SET_INT_CPUTX or bMSEBIS_SET_INT_DMATX in the rMSEBIS_INT register (see **Section 10.4.7.4, MSEBI Slave: Register Access by Master**).
    The value to write on the rMSEBIS_INT register depends on:
    1) The initiator of the transfer (CPU/DMA)
    2) The side of the transfer (TX)
    3) The MSEBI_CS[n]_N used for the transfer
- At the reception of the “end of block event”, MSEBI slave pushes the block transfer descriptor on the corresponding FIFO (CPU receive FIFO or DMA transmit FIFO) at a location in memory pointed by rMSEBIS_EOB_ADDR register (an offset is applied depending on the MSEBI_CS[n]_N and access type: CPU/DMA).
- MSEBI slave sets the interrupt on rMSEBIS_STATUS_INT0 register
- CPU manages the interrupt by polling the descriptor on memory before resetting block transfer descriptor to 0 and clearing the interrupt with rMSEBIS_CLR_INT register

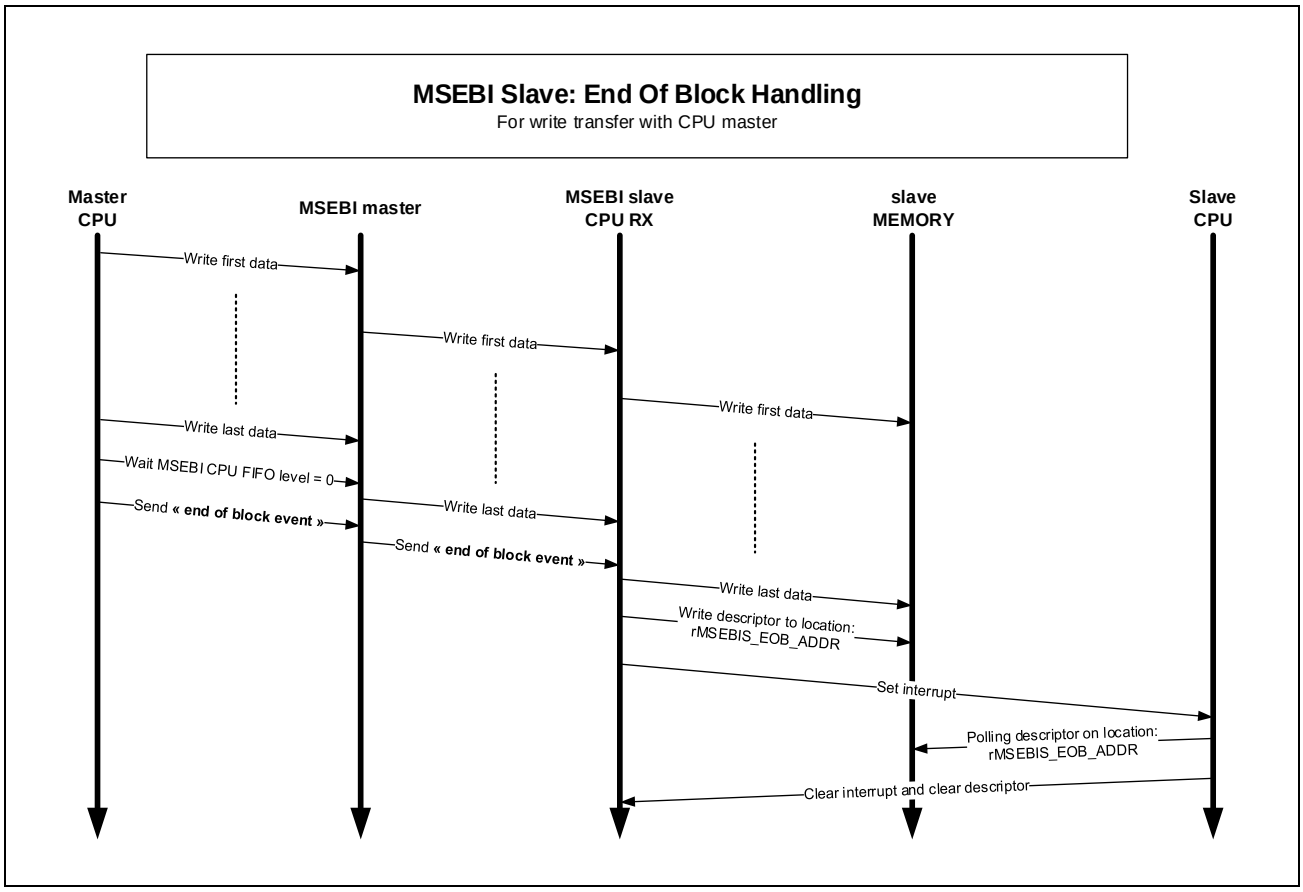

Figure 10.42 MSEBI Slave: End of Write Block Transfer from MSEBI CPU Master

The section below presents the sequence to allow the slave to detect the end of a read block transfer (n = 0..3):

- MSEBI master read a block of data from the slave with CPU or DMA part of the MSEBI master controller.
  - See following figures:
    Figure 10.58, MSEBI Slave CPU FIFOs Example 1.
    Figure 10.62, Slave DMA FIFOs for Requests from Master MSEBI DMA RX FIFOs.
- At the end of the transfer, the firmware on master side sends an “end of block event” to the slave by writing the rMSEBIS_INT register (see **Section 10.4.7.4, MSEBI Slave: Register Access by Master**).
  - The value to write on bMSEBIS_SET_INT_CPURX or bMSEBIS_SET_INT_DMARX in the rMSEBIS_INT register depends on:
    1) The initiator of the transfer (CPU/DMA)
    2) The side of the transfer (RX)
    3) The MSEBI_CS[n]_N used for the transfer
- MSEBI slave sets the interrupt on rMSEBIS_STATUS_INT0 register.
- Slave CPU manages the interrupt by clearing the interrupt with rMSEBIS_CLR_INT register.

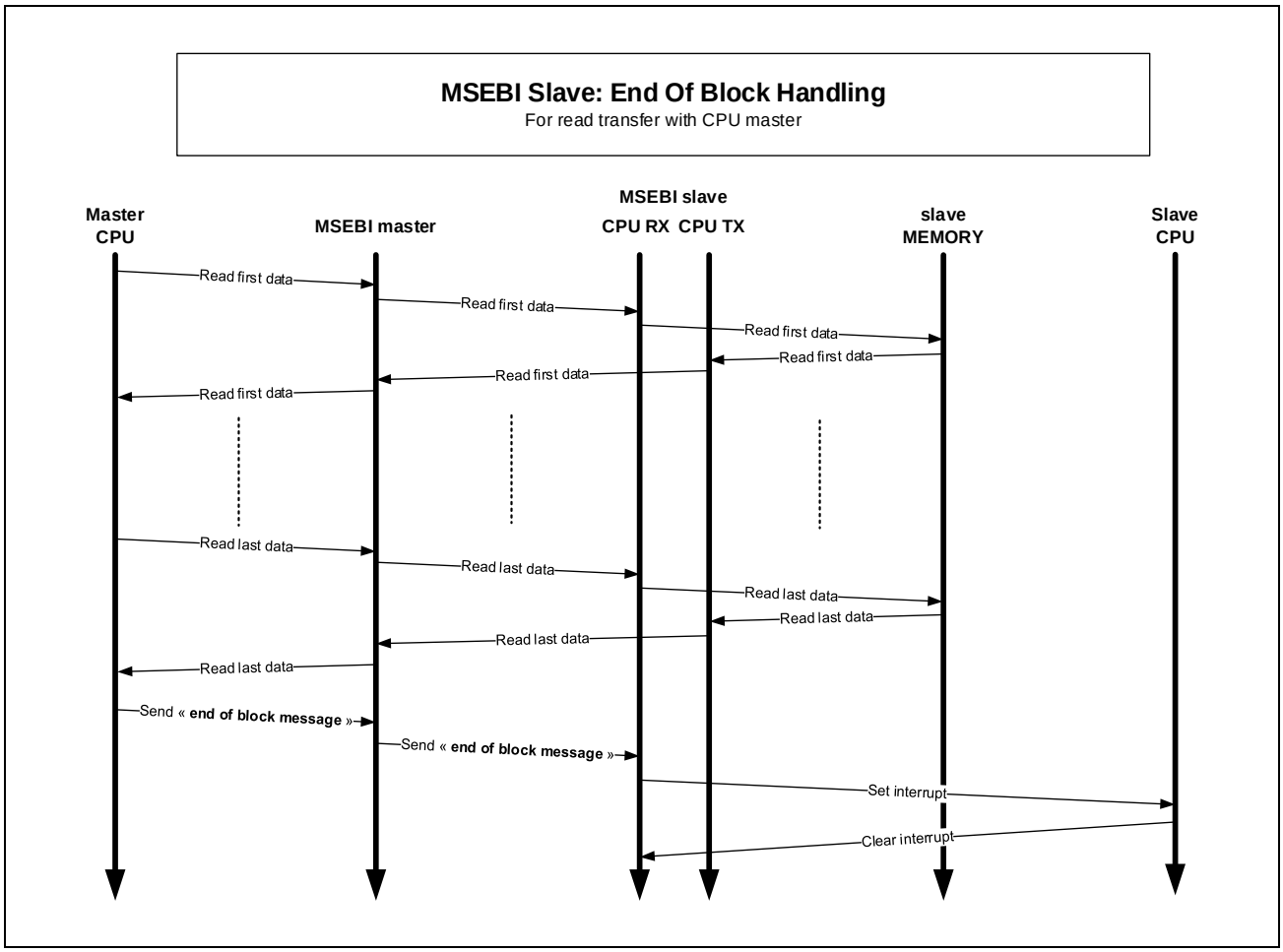

Figure 10.43 MSEBI Slave: End of Read Block Transfer from MSEBI CPU Master

10.4.6 MSEBI Master Mode

10.4.6.1 Master Mode Overview

The main features are:

- Asynchronous and synchronous mode
- MSEBI interface width selectable from 8, 16 and 32 bits
- Multi DLE mode
- Burst mode on MSEBI Interface
- Support for burst access 1-4-8-16 on AHB
- Prefetch capability for CPU receive FIFO
- DMA coupling with 4 DMA channels (external request reception capability)
- CPU transmit and receive FIFO for master: 2 × 32 Words × 32 bits
- DMA transmit and receive FIFO for master: 4 × 32 Words × 64 bits with configurable watermark level
- Up to 4 chip select Lines
- Programmable address capability from 2 B...4 GB
- Programmable setup time on read/write
- Programmable hold time on read/write
- External wait request (can be enabled or disabled)

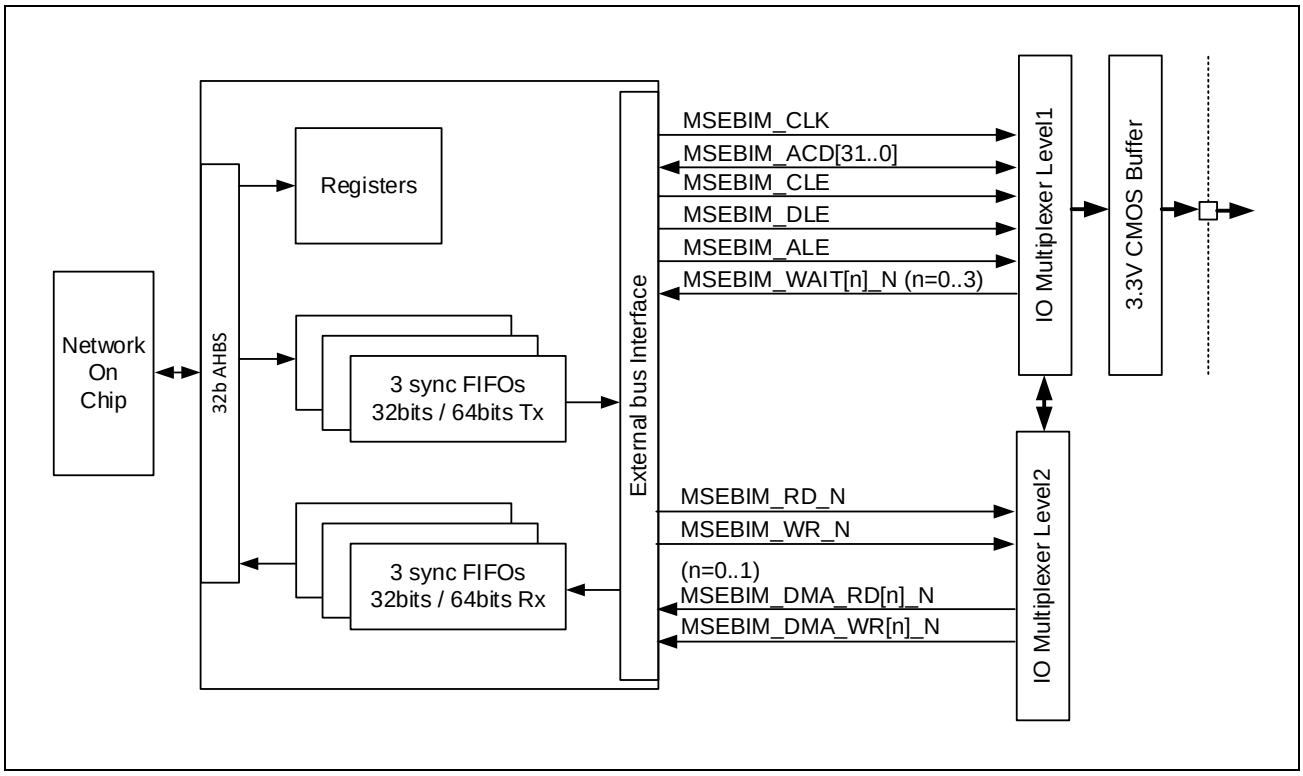

Figure 10.44 Master Mode Overview

10.4.6.2 MSEBI Master: Burst Mode

To manage the burst access, MSEBI interface must be configured with following features:

- Synchronous mode enabled (Set bMSEBIM_CONFIG bits).
- Burst mode enabled (Set bMSEBIM_BURST_ENABLE bit to 1’b1)
- Burst is allowed on each CS[n]_N (n = 0..3)
- Burst is allowed depending on the chip select configuration. Access size must match with the following conditions:
  - MSEBI configured in Mode32: Access Word
  - MSEBI configured in Mode16: Access Word and Half Word.
  - MSEBI configured in Mode8: All type of access.
- The burst cycle can be generated by five potential requesters
  - CPU
  - DMA Transmit FIFO on CS0_N
  - DMA Transmit FIFO on CS1_N
  - DMA Receive FIFO on CS0_N
  - DMA Receive FIFO on CS1_N
- Between each requester, the arbiter manages a Round Robin Priority
- Prefetch mode is only available if:
  - Burst mode is enable with bMSEBIM_BURST_ENABLE for each CS[n]_N (n = 0..3)
  - In CPU mode: bMSEBIM_BURST_SIZEMAX_CPUREAD is set to a value greater than 1
  - In DMA mode: bMSEBIM_BURST_SIZEMAX_DMAREAD is set to a value greater than 1
- MSEBI will generate a burst if possible
- The burst size is undefined
  - MODE32: From 4..1 kbytes and never cross a 1 kbyte boundary
  - MODE16: From 2..1 kbytes and never cross a 1 kbyte boundary
  - MODE8: From 1..1 kbyte and never cross a 1 kbyte boundary
- The max burst size is configured by
  - CPU: bMSEBIM_BURST_SIZEMAX_CPUREAD and bMSEBIM_BURST_SIZEMAX_CPUWRITE bits

    CPU prefetch: bMSEBIM_BURST_SIZEMAX_CPUREAD is also used to control the maximum number of words to be read during a prefetch operation.
  - DMA: bMSEBIM_BURST_SIZEMAX_DMAREAD and bMSEBIM_BURST_SIZEMAX_DMAWRITE bits
- FIFO size
  - CPU: 32 words × 32 bits as Transmit and Receive FIFO
  - DMA: For each CS[n]_N (n = 0..1), 32 words × 64 bits as Transmit and Receive FIFO

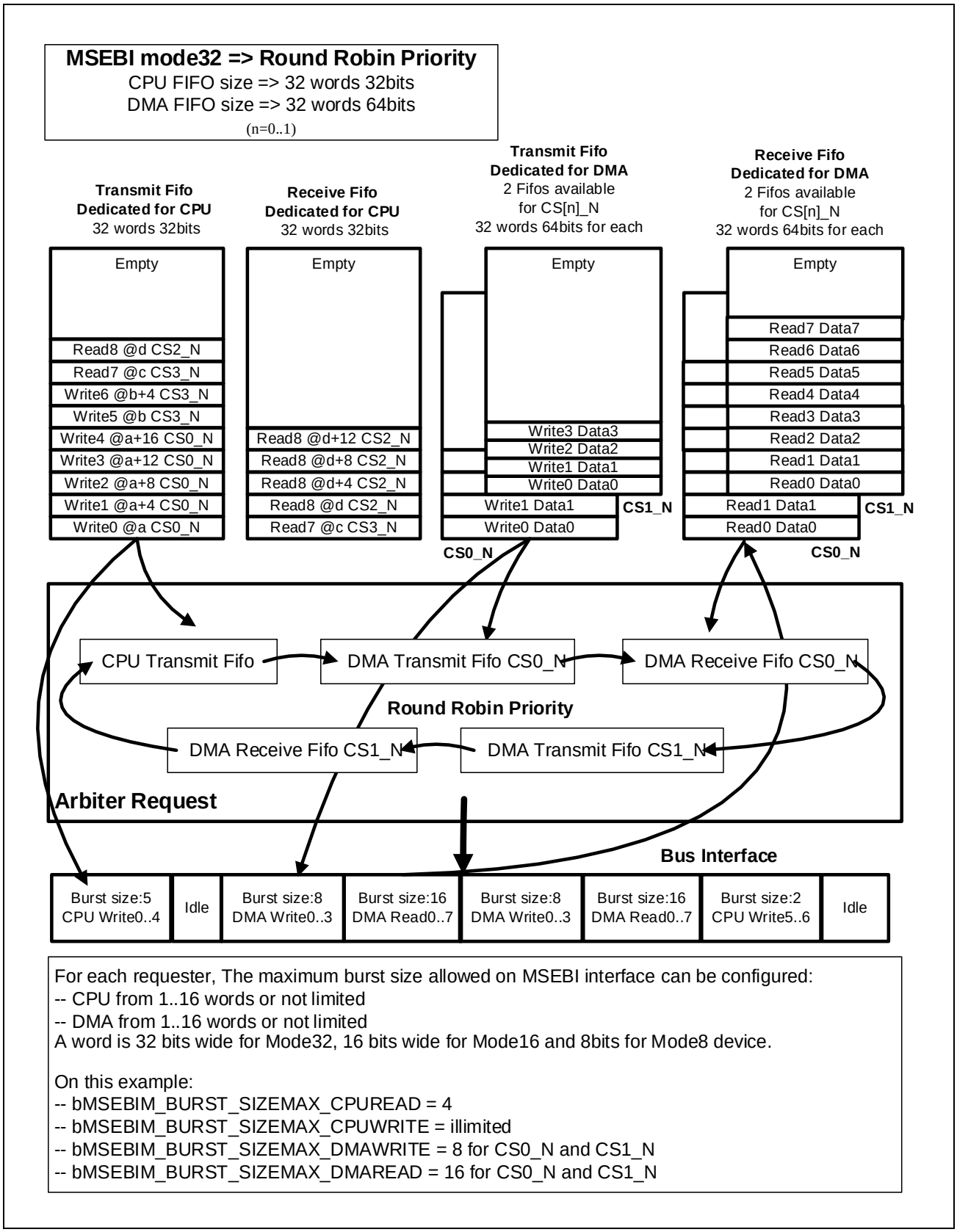

Figure 10.45 MSEBI: Round Robin Priority

The main features of “Burst access” are:

- For CPU FIFO, the order of access must be strictly respected, whatever the chip select that is accessed.
- For DMA Transmit FIFO (CS0_N dedicated), the order of access must be strictly respected on chip select CS0_N.
- For DMA Transmit FIFO (CS1_N dedicated), the order of access must be strictly respected on chip select CS1_N.
- For DMA Receive FIFO (CS0_N dedicated), the order of access must be strictly respected on chip select CS0_N.
- For DMA Receive FIFO (CS1_N dedicated), the order of access must be strictly respected on chip select CS1_N.
- Between each FIFO (CPU, 4 DMA FIFO), the orders of access are not respected.
- The bus interface can stop a burst access at any time.
- The burst on the external bus is generated on data cacheable or not cacheable, the information in the AHB burst is not used.
- A FIFO stores all the requests of the CPU (single, burst, read, write).
- On Write, the bus interface is always trying (depending on the contents of the FIFO) to generate a burst size as large as possible (Limited by burst size max allowed).
- On Read, when after the current read request, the FIFO is empty, the bus interface generates a burst access in prefetch mode to anticipate prefetch the following CPU read request until FIFO full or limited by burst size max allowed: bMSEBIM_BURST_SIZEMAX_CPUREAD.
  - Address is supposed to be linearly incremented by 1/2/4 in 8/16/32 bits mode respectively and never crosses a 1 kB boundary.

(1) Master CPU FIFOs

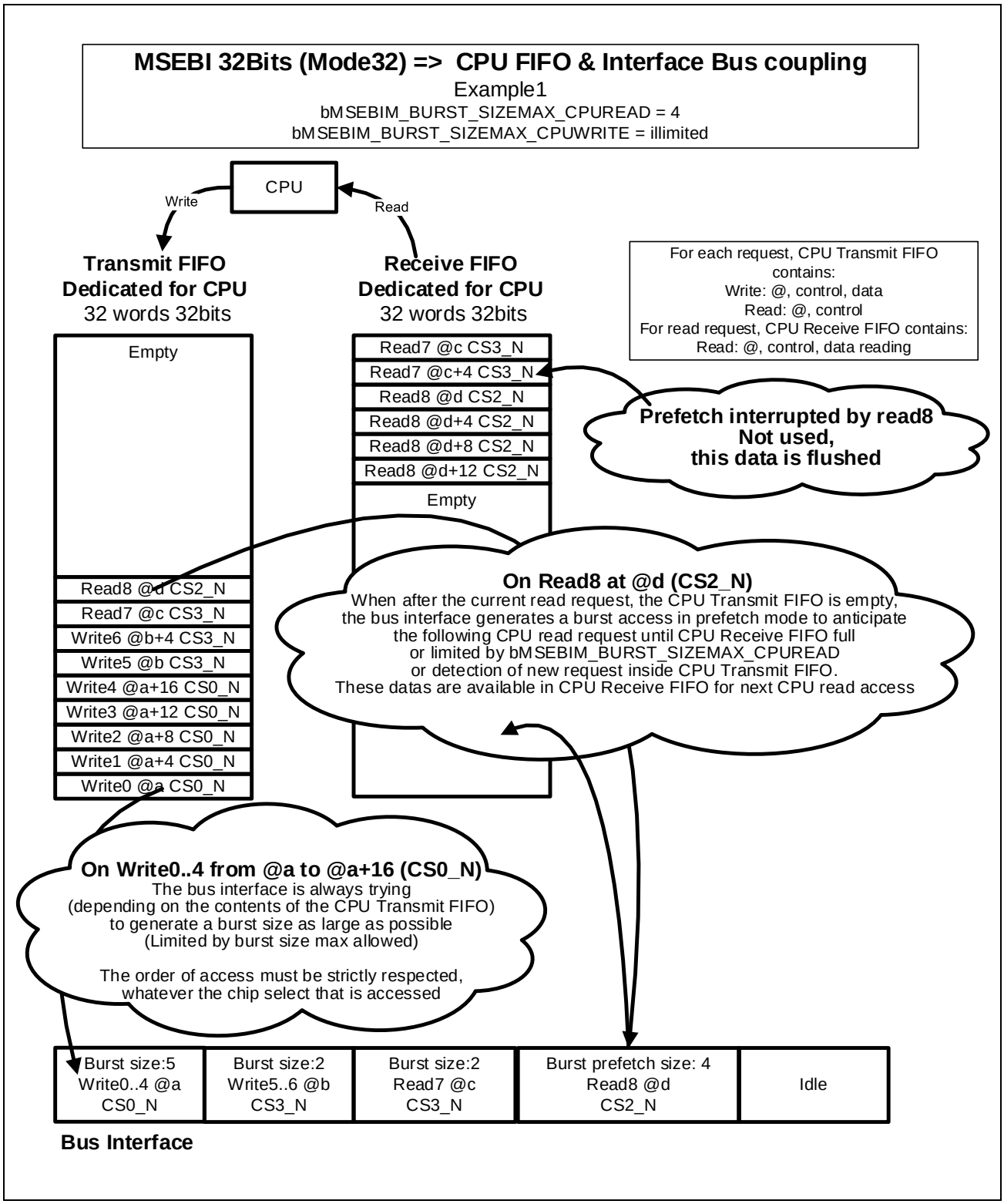

Figure 10.46 MSEBI: Burst Mode, CPU Transmit and Receive FIFO and Bus Interface Coupling, Example1

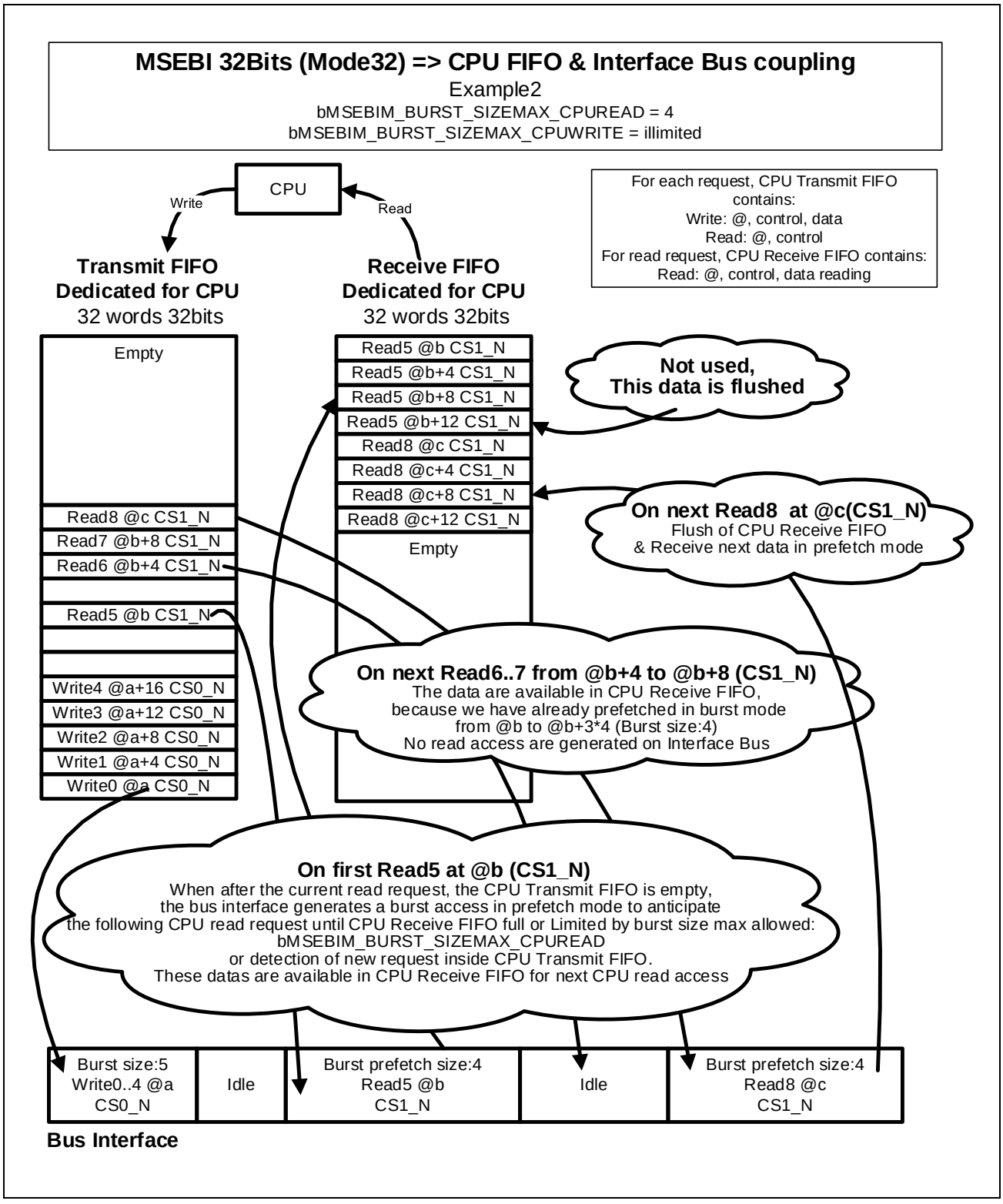

Figure 10.47 MSEBI: Burst Mode, CPU Transmit and Receive FIFO and Bus Interface Coupling, Example2

(2) Master DMA FIFOs

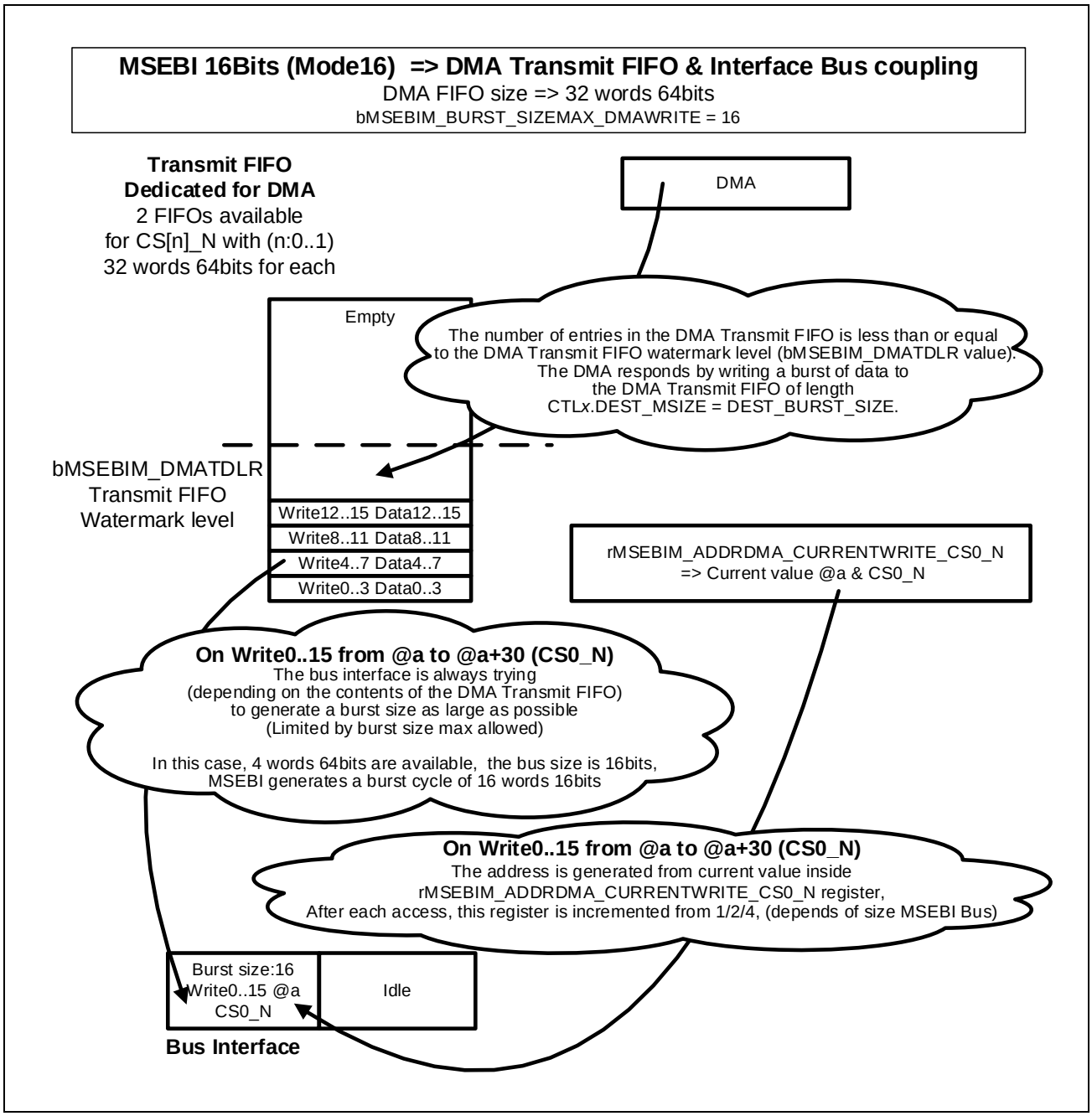

Figure 10.48 MSEBI: Burst Mode, DMA Transmit FIFO and Bus Interface Coupling

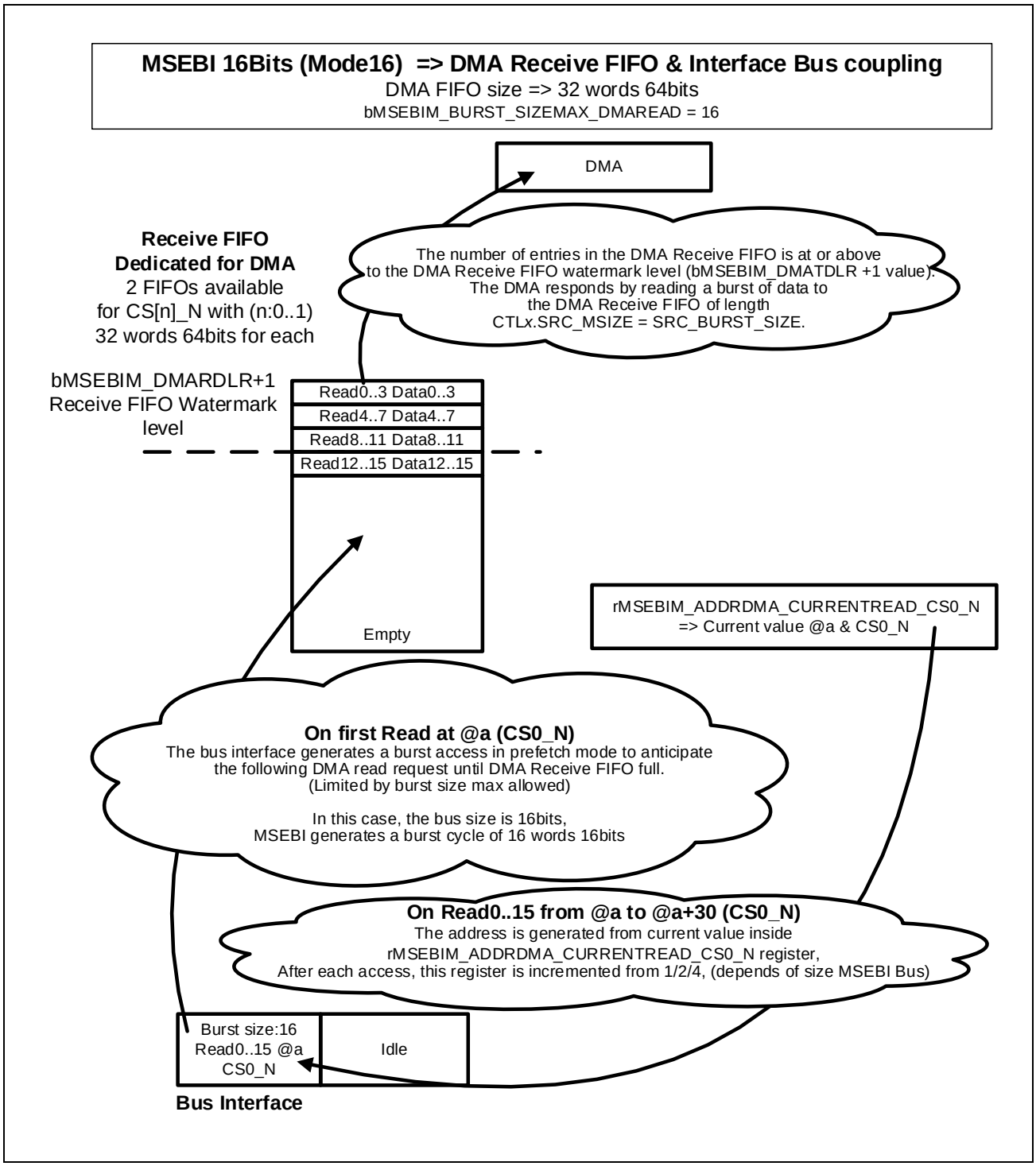

Figure 10.49 MSEBI: Burst Mode, DMA Receive FIFO and Bus Interface Coupling

(3) Master Multi DLE

Multi DLE (bMSEBIM_MULTI_DLE) mode can be set for each chip select independently and presents the following limitations:

- Write only
- No wait management
- WRDLEDATA_NB / WRDLEDATA_B must be greater or equal to 2 MSEBIM_CLK

This mode can be used for external FPGA configuration with the following possibilities:

- With an external flash: expensive and hard to maintain
- With JTAG or Passive Serial: slow
- With Parallel Access: can be driven by an MSEBI bus, if MSEBI_DLE signal is used as a clock.
  - Advantage of this method is:
    <Faster than JTAG method>
    In standard mode, FPGA needs to communicate with RZ/N1. MSEBI may be use for standard communication between RZ/N1 and FPGA so, if it is also used for programming of the FPGA, no other bus needs to be implemented (save some pins/place on the PCB).
  - Inconvenience of this method, over standard use of MSEBI, is that it supports only single access.

The main idea of Multi DLE mode is to permit to use external FPGA configuration with an MSEBI in burst mode. In the figure below, the Multi DLE mode allows to manage the Configuration Interface of the FPGA. The signal MSEBIM_DLE will be used as a Configuration Clock Input, and will also provide a rising edge for each valid signal of data. The signals MSEBIM_CLK, MSEBIM_ALE and MSEBIM_CLE are not used. Communication may be in 8 or 16 bits depending on the target FPGA.

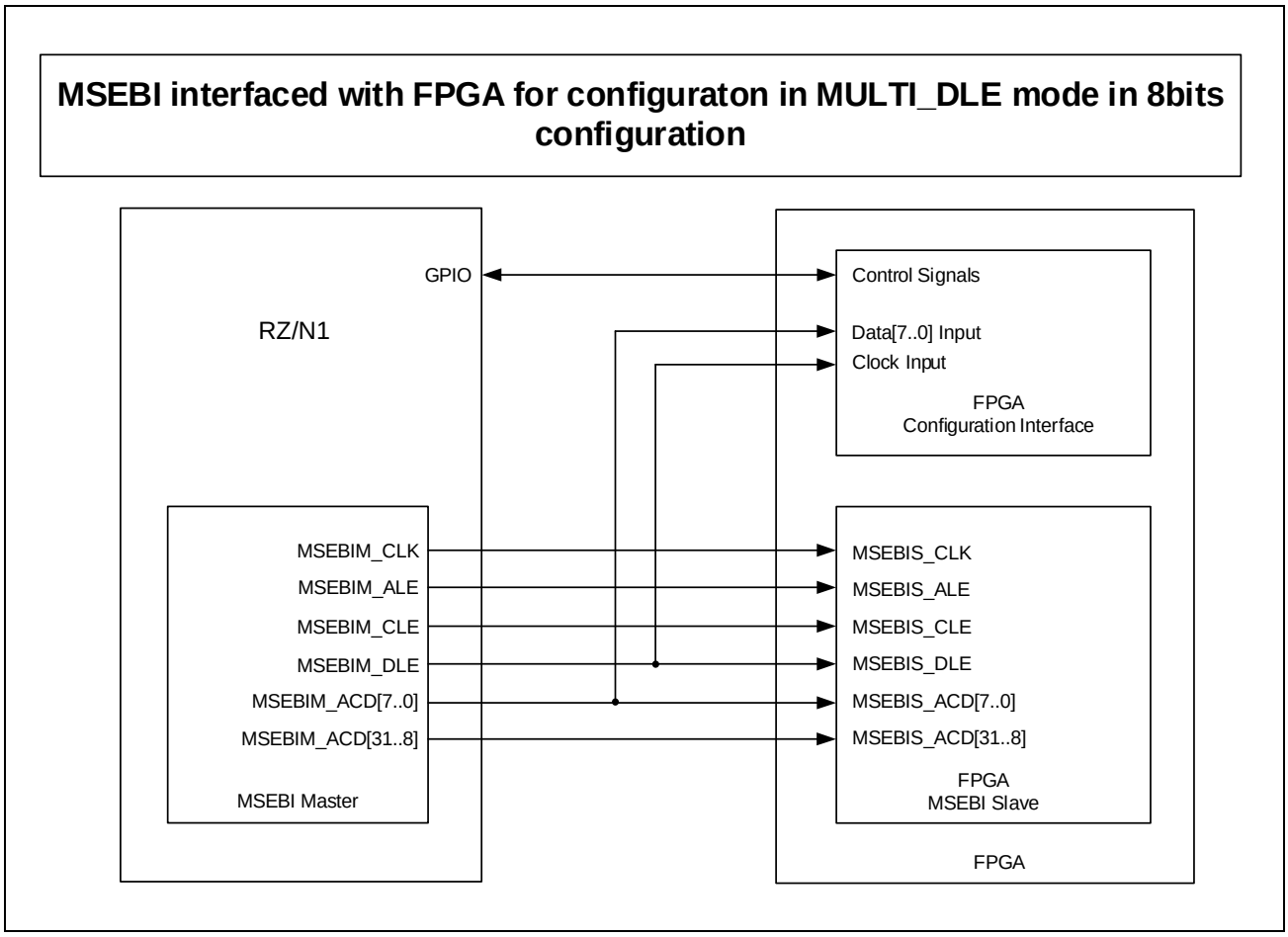

Figure 10.50 MSEBI Interfaced with FPGA for Configuration in MULTI_DLE Mode in 8 bits Configuration

CAUTION

- To be able to use MSEBI in standard mode after the configuration of the FPGA, signal MSEBI_DLE must be routed in parallel to another IO of the FPGA as the configuration clock input is a dedicated configuration pin.
- Depending on targeted FPGA, some MSEBI_ACD bits must be routed in parallel to others IO of the FPGA as some configuration data pins are dedicated to configuration.

In Multi DLE mode, MSEBI_DLE signal is set on last cycle of WRDLEDATA_NB or WRDLEDATA_B, see figures below with WRDLEDATA_NB and WRDLEDATA_B configured to 2 and 3.

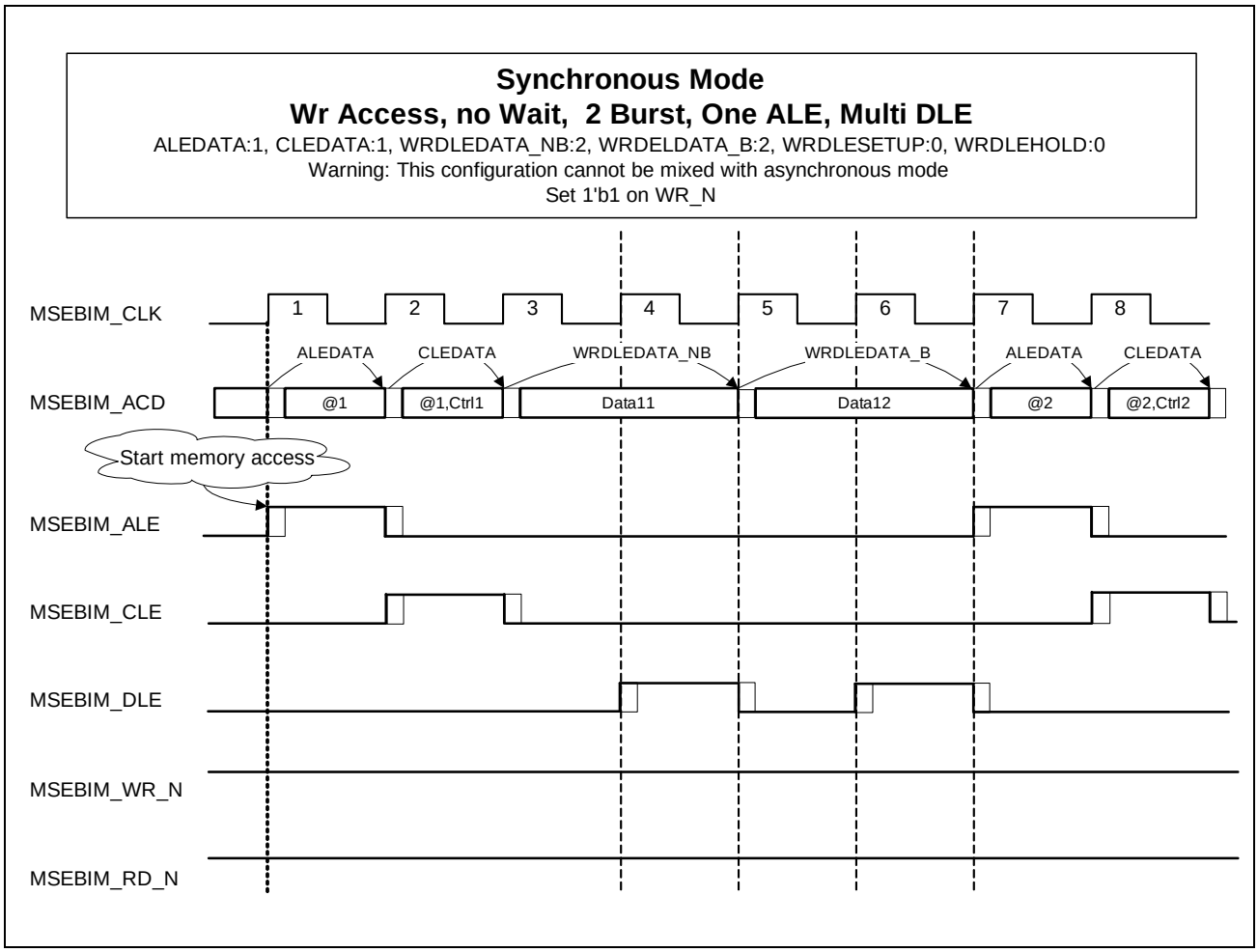

Figure 10.51 Timing1, Synchronous Mode, Burst, Multi DLE

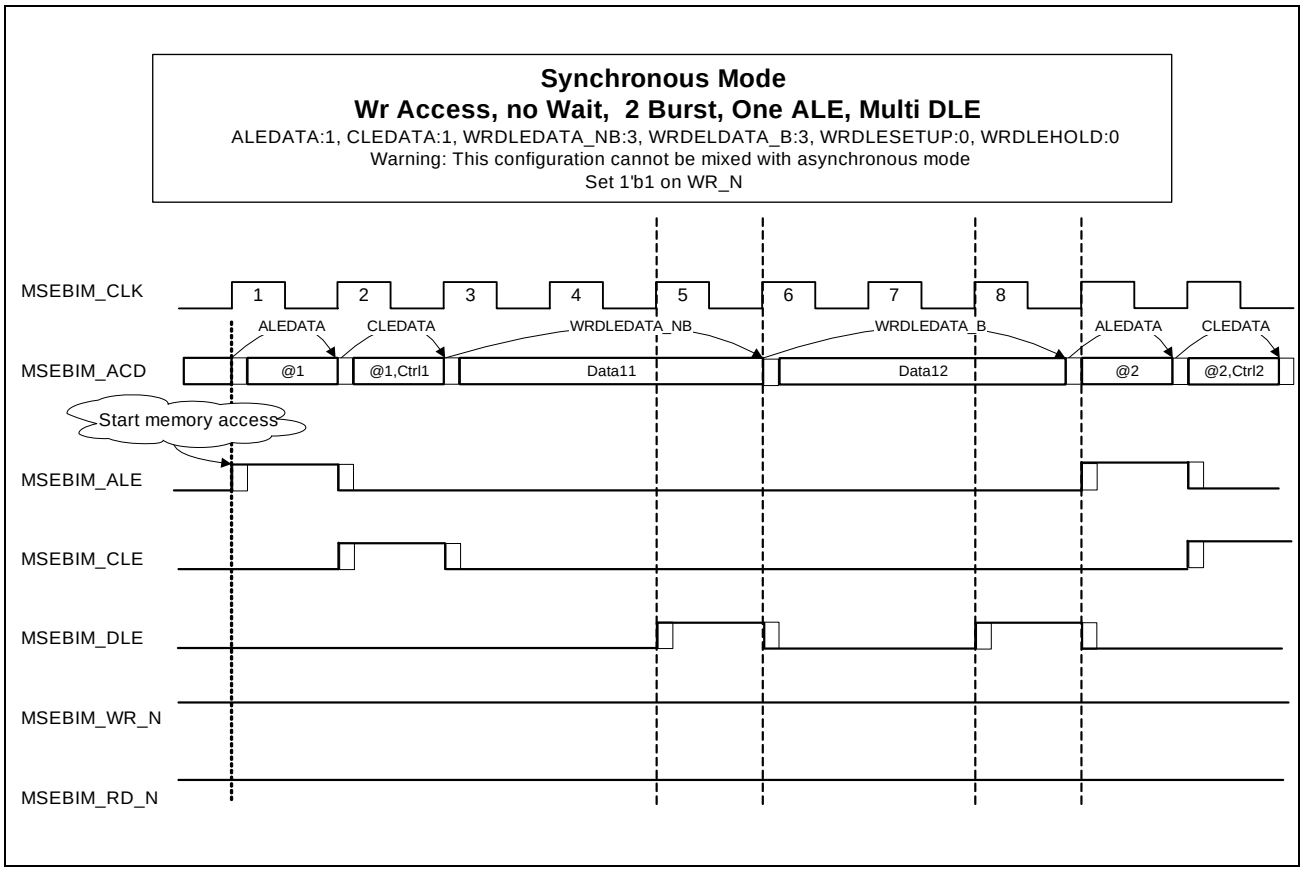

Figure 10.52 Timing2, Synchronous Mode, Burst, Multi DLE

10.4.6.3 MSEBI Master: DMA Control

The MSEBI controller has DMA capability. It has a handshaking interface to a DMA Controller to request and control transfers. The AHB bus is used to perform the data transfer to or from the DMA. In this mode, DMA controller must be configured in peripheral flow controller mode. The DMA always transfers data using DMA burst transactions if possible, for efficiency.

The MSEBI controller uses four DMA channels, two for the transmit data and two for the receive data (one for each of CS0_N and CS1_N).

The MSEBI has these following DMA registers:

- bMSEBIM_USE_EXT_RDDMA_REQ and bMSEBIM_USE_EXT_WRDMA_REQ bits: Control register to enable DMA control operation by external pins
- rMSEBIM_TDMACR_CS0_N, rMSEBIM_RDMACR_CS0_N, rMSEBIM_TDMACR_CS1_N, and rMSEBIM_RDMACR_CS1_N registers: Control register to configuration DMA operation (block size and burst request size) and start/stop DMA transfer.
- bMSEBIM_DMATDLR bit (Transmit watermark level): Register to set the DMA Transmit FIFO level at which a DMA request is made.
- bMSEBIM_DMARDLR bit (Receive watermark level): Register to set the DMA Receive FIFO level at which a DMA request is made.
- bMSEBIM_BURST_SIZEMAX_DMAWRITE and bMSEBIM_BURST_SIZEMAX_DMAREAD bits: Burst Size Max Allowed on Write and Read access

To manage the burst access, MSEBI interface must be configured with following features:

- Synchronous mode enable (Set bMSEBIM_CONFIG bits).
- Burst mode enable (Set bMSEBIM_BURST_ENABLE bit to 1’b1)

The max burst size is configured by:

- DMA: bMSEBIM_BURST_SIZEMAX_DMAREAD and bMSEBIM_BURST_SIZEMAX_DMAWRITE bits

To enable the DMA Controller interface on the MSEBI and enables the transmit handshaking interface (with n = 0..1):

- MSEBI supports 64 bits single transaction size. (refer to the bMSEBIM_SINGLE_DEST_WIDTH field in the rMSEBIM_TDMACR_CS[n]_N registers)
- The MSEBI must be programmed by the processor with the number of single transactions (block size) that are to be transmitted. This is programmed into the bMSEBIM_DEST_BLOCK_SIZE field in the rMSEBIM_TDMACR_CS[n]_N registers of MSEBI for DMA Transmit FIFO.
- The MSEBI use all incoming AHB request with an address included in the memory range reserved for the DMA FIFO as a push on the FIFO (see **Section 10.3.2.2, rMSEBIM_DMA_FIFOWRITE_CS[n]_N — DMA Transmit FIFO (64 KB) (n = 0..1)**). In order to be able to transfer max number of elements (max DEST_BLOCK_SIZE = 8191 single elements), the destination pointer of the DMAC register (DAR[ch]) must be set to the base address of the FIFO.
- The MSEBI must be programmed by the processor with the size of burst transfer (burst size). This is programmed into the bMSEBIM_DEST_BURST_SIZE field in the rMSEBIM_TDMACR_CS[n]_N registers of MSEBI for DMA Transmit FIFO.

- Enable external DMA request, writing a 1 into the bMSEBIM_USE_EXT_WRDMA_REQ bit field of rMSEBIM_DMATDLR_CS[n]_N registers.
- Writing DMA Write Access address (rMSEBIM_ADDRDMA_WRITE_CS[n]_N registers). First block address used by DMA controller to start a DMA transfer from DMA Transmit FIFO to MSEBI bus when the firmware set “1” on bMSEBIM_TDMAE1 (rising edge).
- Writing Transmit watermark level (bMSEBIM_DMATDLR). This bit field controls the level at which a DMA burst request is made by the Transmit logic.
- Writing a 1 into the bMSEBIM_TDMAE1 bit field of rMSEBIM_TDMACR_CS[n]_N registers. Now, the DMA Transmit channel is running until all data have been transferred (DEST_BLOCK_SIZE single transactions). The DMA controller is stopped (the bMSEBIM_TDMAE1 bit is reset to 0) when all data (DEST_BLOCK_SIZE single transactions) have been transferred from DMA Transmit FIFO to MSEBI bus (FIFO Empty)

NOTE

A rising edge on bMSEBIM_TDMAE1 flushes the DMA Transmit FIFO

Case with external DMA request enable, writing a 1 into the bMSEBIM_USE_EXT_WRDMA_REQ:

When bMSEBIM_TDMAE1 is set from 0 to 1 (start DMA), the MSEBI controller then waits MSEBIM_DMA_WR0_N or MSEBIM_DMA_WR1_N (depends on CS_N used) is reset to 0.

At this moment, the DMA Transmit FIFO requests are made to the DMA controller whenever the number of entries in the DMA Transmit FIFO is less than or equal to the watermark level bMSEBIM_DMATDLR value.

The DMA responds by writing a burst of data to the DMA Transmit FIFO buffer.

At this moment, the DMA Transmit FIFO is read and the read data is output to MSEBI Bus in Burst mode.

Case with external DMA request disable, writing a 0 into the bMSEBIM_USE_EXT_WRDMA_REQ:

Same function, but not use external DMA request (MSEBIM_DMA_WR0_N or MSEBIM_DMA_WR1_N is reset to 0).

The DMA transfer starts immediately.

To enable the DMA Controller interface on the MSEBI and enables the receive handshaking interface (with n = 0..1):

- MSEBI supports 64 bits single transaction size. (refer to the bMSEBIM_SINGLE_SRC_WIDTH field in the rMSEBIM_RDMACR_CS[n]_N registers)
- The MSEBI must be programmed by the processor with the number of single transactions (block size) that are to be received. This is programmed into the bMSEBIM_SRC_BLOCK_SIZE field in the rMSEBIM_RDMACR_CS[n]_N registers of MSEBI for DMA Receive FIFO.
- The MSEBI use all incoming AHB request with an address include in the memory range reserved for the DMA FIFO as a read on the FIFO (see **Section 10.3.2.1, rMSEBIM_DMA_FIFOREAD_CS[n]_N — DMA Receive FIFO (64 KB) (n = 0..1)**). In order to be able to transfer max number of elements (max DEST_BLOCK_SIZE = 8191 single elements), the source pointer of the DMAC register (SAR[ch]) must be set to the base address of the FIFO.
- The MSEBI must be programmed by the processor with the size of burst transfer (burst size). This is programmed into the bMSEBIM_SRC_BURST_SIZE field in the rMSEBIM_RDMACR_CS[n]_N registers of MSEBI for DMA Receive FIFO.
- Enable external DMA request, writing a 1 into the bMSEBIM_USE_EXT_RDDMA_REQ bit field of rMSEBIM_DMARDLR_CS[n]_N registers.

- Writing DMA Read Access address (rMSEBIM_ADDRDMA_READ_CS[n]_N registers). First block address used by DMA controller to start a DMA transfer from MSEBI bus to DMA Receive FIFO when the firmware set “1” on bMSEBIM_RDMAE1 (rising edge).
- Writing Receive watermark level (bMSEBIM_DMARDLR). This bit field controls the level at which a DMA burst request is made by the Receive logic.
- Writing a 1 into the bMSEBIM_RDMAE1 bit field of rMSEBIM_RDMACR_CS[n]_N registers. Now, the DMA Receive channel is running until all data have been transferred (SRC_BLOCK_SIZE single transactions). When all data have been transferred, the bMSEBIM_RDMAE1 bit is reset to 0, the DMA controller is stopped and DMA Receive FIFO is flushed.

NOTE

A rising edge on bMSEBIM_RDMAE1 flushes the DMA Receive FIFO

Case with external DMA request enable, writing a 1 into the bMSEBIM_USE_EXT_RDDMA_REQ:

When bMSEBIM_RDMAE1 is set from 0 to 1 (start DMA), the MSEBI controller then waits MSEBIM_DMA_RD0_N or MSEBIM_DMA_RD1_N (depends on CS_N used) is reset to 0.

At this moment, the data is input from the MSEBI bus and MSEBI controller writes data to DMA Receive FIFO.

When the number of entries in the DMA Receive FIFO is at or above the watermark level bMSEBIM_DMARDLR+1, the DMA responds by reading a burst of data from the DMA Receive FIFO buffer.

Case with external DMA request disable, writing a 0 into the bMSEBIM_USE_EXT_RDDMA_REQ:

Same function, but not use external DMA request (MSEBIM_DMA_RD0_N or MSEBIM_DMA_RD1_N is reset to 0).

The DMA transfer starts immediately.

(1) Overview on DMA Operation

DMA controller must be configured in peripheral flow controller mode, because MSEBI is a peripheral flow controller and must know the size of block transferred (see detail below with n = 0..1).

The MSEBI must be programmed by the processor with the number of data items (block size) that are to be transmitted or received. This is programmed into the bMSEBIM_DEST_BLOCK_SIZE / bMSEBIM_SRC_BLOCK_SIZE field in the rMSEBIM_TDMACR_CS[n]_N and rMSEBIM_RDMACR_CS[n]_N registers of MSEBI for DMA Transmit FIFO and Receive FIFO, respectively. The block is broken into a number of transactions, each initiated by a request from the MSEBI.

The DMA Controller and the MSEBI must also be programmed with the number of single transactions by burst to be transferred for each DMA request. This is also known as the burst transaction length, and is programmed into:

- DMA controller (ch = 0..7): The DEST_MSIZE / SRC_MSIZE fields of the DMAC.CTL[ch] register for Transmit FIFO and Receive FIFO, respectively.
- MSEBI: Programmed into the bMSEBIM_DEST_BURST_SIZE / bMSEBIM_SRC_BURST_SIZE field in the rMSEBIM_TDMACR_CS[n]_N and rMSEBIM_RDMACR_CS[n]_N registers of MSEBI for Transmit FIFO and Receive FIFO, respectively.

Be careful to take into account the size of AHB bus managed by DMA controller configured through DST_TR_WIDTH/SRC_TR_WIDTH fields of the DMAC.CTL[ch] register for DMA Transmit FIFO and DMA Receive FIFO respectively.

MSEBI supports 64 bits width (refer to bMSEBIM_SINGLE_DEST_WIDTH and bMSEBIM_SINGLE_SRC_WIDTH bits).

Recommended value to manage correctly all transactions in burst and single mode.

- Size of AHB Bus: 64 bits
- Size of Burst line: 4×64 bits

CAUTION

The burst size transaction must have the same values on DMAC and MSEBI.

(2) External DMA Request

4 inputs are available to start a DMA transfer frame.

- MSEBIM_DMA_RD0_N: Start and Stop Receive frame on CS0_N.
- MSEBIM_DMA_RD1_N: Start and Stop Receive frame on CS1_N.
- MSEBIM_DMA_WR0_N: Start and Stop Transmit frame on CS0_N.
- MSEBIM_DMA_WR1_N: Start and Stop Transmit frame on CS1_N.

All external requests are asynchronous.

Before use (n = 0..1):

Each external request (MSEBIM_DMA_RD[n]_N) must be enabled for DMA read.

- With bMSEBIM_USE_EXT_RDDMA_REQ in rMSEBIM_DMARDLR_CS[n]_N registers
- With bMSEBIM_RDMAE1 in rMSEBIM_RDMACR_CS[n]_N registers

After channel enabled and when a level “0” is detected on MSEBIM_DMA_RD[n]_N, a DMA read cycle is started from MSEBI bus to DMA Receive FIFO.

During a DMA transfer, if MSEBIM_DMA_RD[n]_N is set to “1”, the DMA read cycle is suspended. It means that the DMA is still allowed to read data from the DMA Receive FIFO but no read request is sent to the MSEBI bus.

If MSEBIM_DMA_RD[n]_N is reset to “0” during a suspended transfer, the transfer will continue and next read command is sent to the MSEBI bus.

The bMSEBIM_RDMAE1 bit is automatically cleared by hardware to disable the DMA in Receive mode after the last transfer in DMA Receive FIFO has completed (SRC_BLOCK_SIZE data item read in DMA Receive FIFO). Software can therefore poll this bit to determine when this channel is free for a new DMA transfer.

Each external request (MSEBIM_DMA_WR[n]_N) must be enabled for DMA write.

- With bMSEBIM_USE_EXT_WRDMA_REQ in rMSEBIM_DMATDLR_CS[n]_N registers
- With bMSEBIM_TDMAE1 in rMSEBIM_TDMACR_CS[n]_N registers

After channel enabled and when a level “0” is detected on MSEBIM_DMA_WR[n]_N, a DMA write request cycle is started from DMA Transmit FIFO to DMA Controller.

During a DMA transfer, if MSEBIM_DMA_WR[n]_N is set to “1”, the DMA write cycle is suspended. It means that the DMA is still allowed to write data on the DMA Transmit FIFO but no write request is send to the MSEBI bus.

If MSEBIM_DMA_WR[n]_N is reset to “0” during a suspended transfer, the transfer will continue and next write command is send to the MSEBI bus.

The bMSEBIM_TDMAE1 bit is automatically cleared by hardware to disable the DMA in Transmit mode after the last transfer in DMA Transmit FIFO has completed (DEST_BLOCK_SIZE data item read in DMA Transmit FIFO). Software can therefore poll this bit to determine when this channel is free for a new DMA transfer.

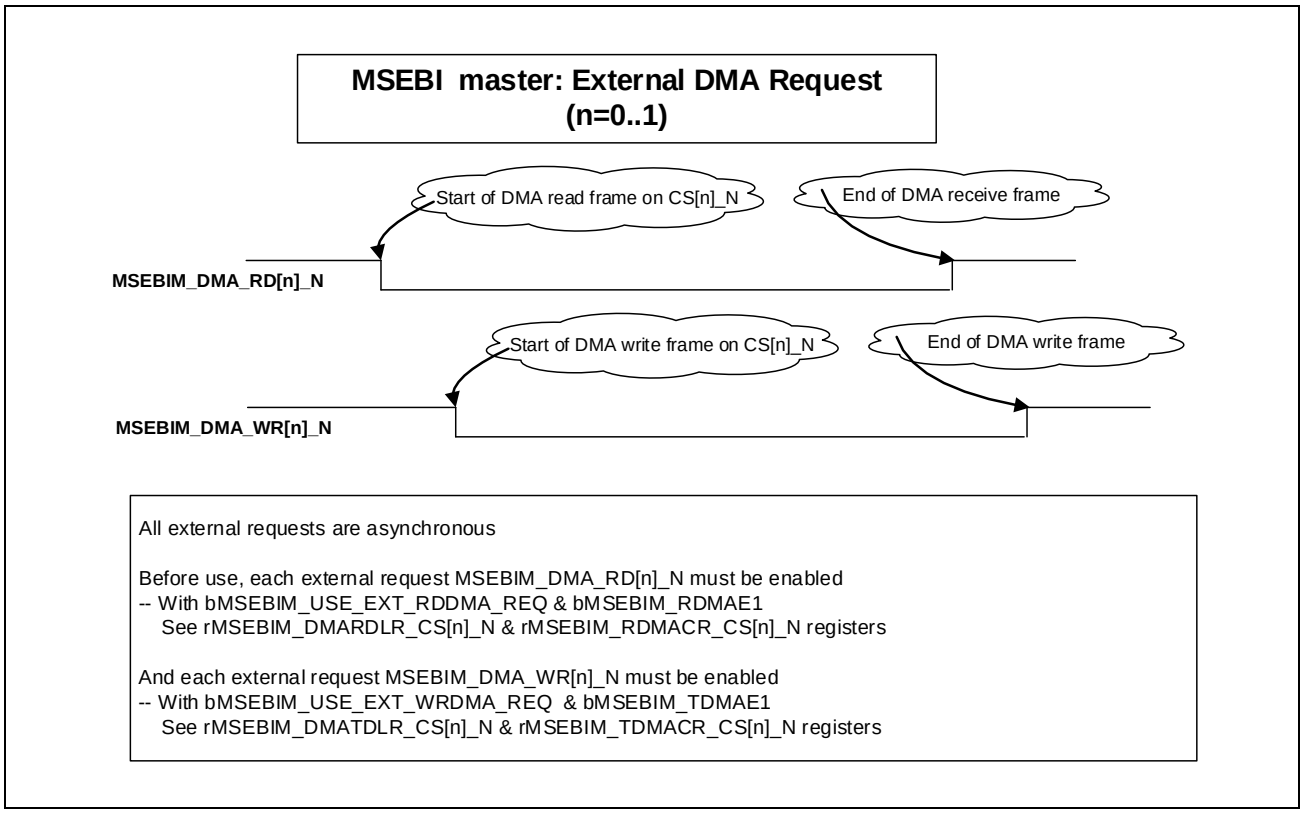

Figure 10.53 MSEBI Master: External DMA Request

(3) Transmit Watermark Level and Transmit FIFO Underflow

During MSEBI bus transfers, transmit FIFO requests are made to the DMA controller (ch = 0..7) whenever the number of entries in the transmit FIFO is less than or equal to the bMSEBIM_DMATDLR value of DMA transmit data level register, this is known as the watermark level. The DMA responds by writing a burst of data to the transmit FIFO buffer, of length DMAC.CTL[ch].DEST_MSIZE = DEST_BURST_SIZE. 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 will run out of data (underflow). To prevent this condition, the user must set the watermark level correctly.

(4) Choosing the Transmit Watermark Level

Consider the example where the assumption is made:

DEST_BURST_SIZE = DMAC.CTL[ch].DEST_MSIZE = FIFO_DEPTH – bMSEBIM_DMATDLR

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

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

DEST_BLOCK_SIZE / DEST_BURST_SIZE = 120/30 = 4

The number of burst transactions in the DMA block transfer is 4. But the watermark level, bMSEBIM_DMATDLR, is quite low. Therefore, the probability of an MSEBI underflow is high where the MSEBI bus 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.

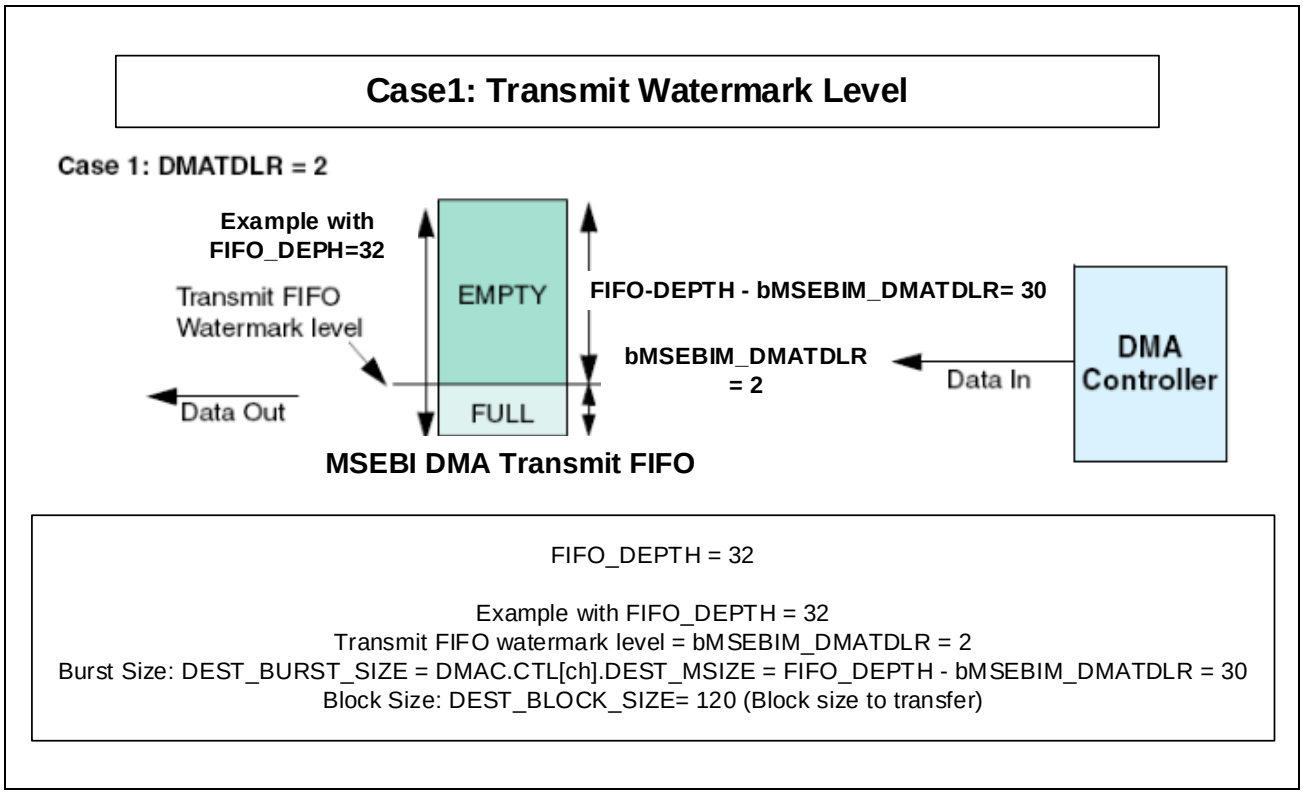

Figure 10.54 MSEBI Case1: Transmit Watermark Level

In the second case, the number of burst transactions in Block:

DEST_BLOCK_SIZE / DEST_BURST_SIZE = 120/2 = 60

In this block transfer, there are 60 destination burst transactions in a DMA block transfer. But the watermark level, bMSEBIM_DMATDLR is very high. Therefore, the probability of an MSEBI underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the MSEBI 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 request bursts per block and worse bus utilization than the former case.

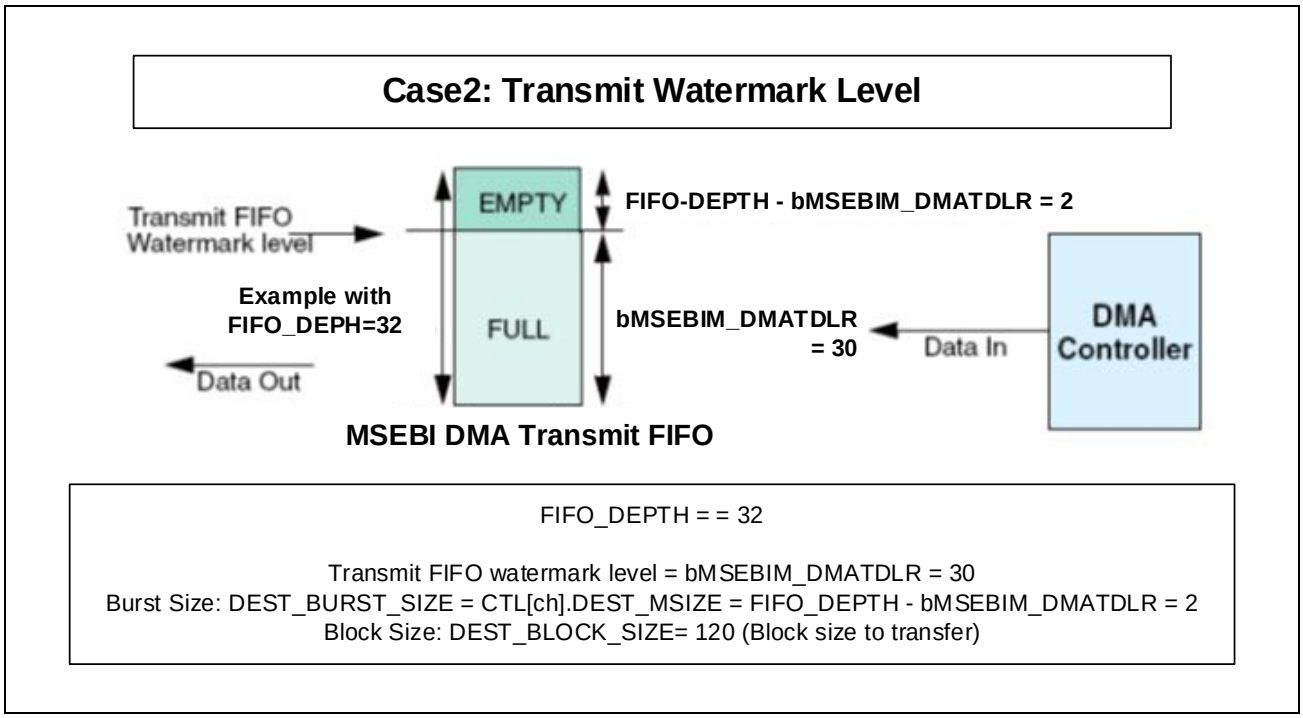

Figure 10.55 MSEBI Case2: Transmit Watermark Level

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 MSEBI 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 bus layer, increases the rate at which the DMA controller can respond to burst transaction requests. This in turn allows the user to decrease the watermark level, which improves bus utilization without compromising the probability of an underflow occurring.

(5) Selecting DEST_MSIZE and Transmit FIFO Overflow

It may cause overflow when there is not enough space in the MSEBI transmit FIFO to service the destination burst request.

Therefore, for optimal operation, we must configure (ch = 0..7):

- DMAC.CTL[ch].DST_TR_WIDTH = 3 (64bits)
- DMAC.CTL[ch].DEST_MSIZE = 1 (4 single transactions)
- bMSEBIM_SINGLE_DEST_WIDTH = 1 (64bits)
- bMSEBIM_DEST_BURST_SIZE = 1 (4 single transactions)
- bMSEBIM_DMATDLR = 28

(6) Receive Watermark Level and Receive FIFO Overflow

During MSEBI bus transfers, receive FIFO requests are made to the DMA controller whenever the number of entries in the receive FIFO is at or above the DMA Receive Data Level Register, that is bMSEBIM_DMARDLR+1.

This is known as the watermark level. The DMA controller (ch = 0..7) responds by reading a burst of data to the receive FIFO buffer of length SRC_BURST_SIZE= DMAC.CTL[ch].SRC_MSIZE.

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

(7) Choosing the Receive Watermark Level

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

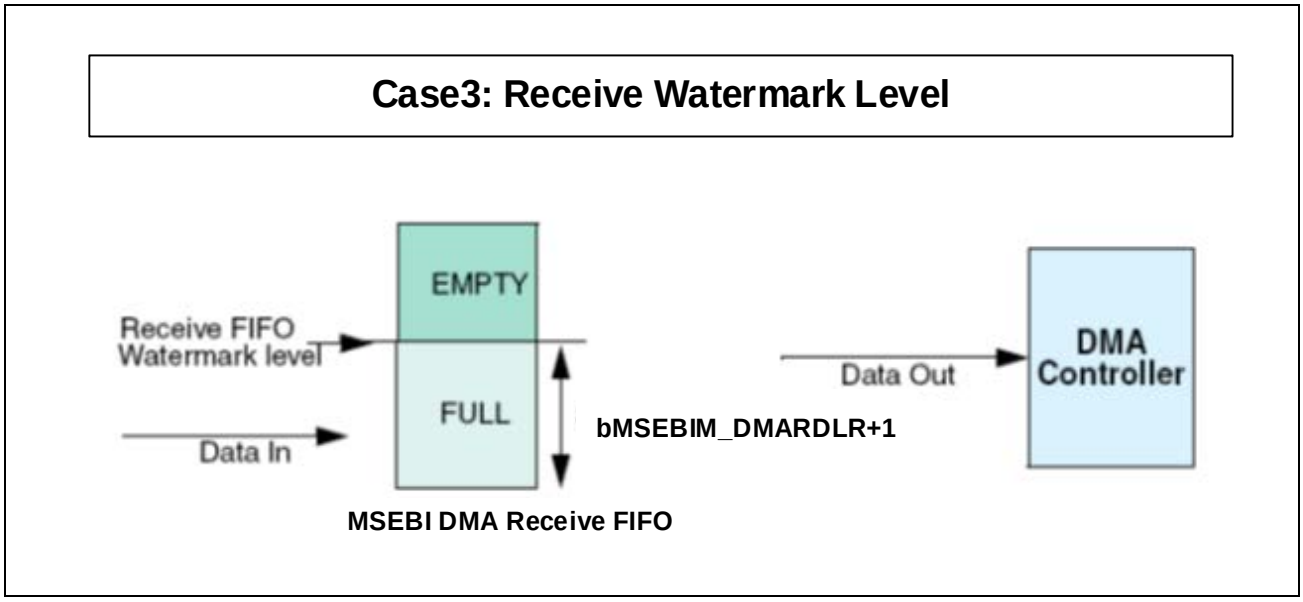

Figure 10.56 MSEBI Case3: Receive Watermark Level

(8) Selecting SRC_MSIZE and Receive FIFO Underflow

It may cause underflow when there is not enough data on the MSEBI receive FIFO to service the source burst request.

Therefore, for optimal operation, we must configure (ch = 0..7):

- DMAC.CTL[ch].SRC_TR_WIDTH = 3 (64bits)
- DMAC.CTL[ch].SRC_MSIZE = 1 (4 single transactions)
- bMSEBIM_SINGLE_SRC_WIDTH = 1 (64bits)
- bMSEBIM_SRC_BURST_SIZE = 1 (4 single transactions)
- bMSEBIM_DMARDLR = 3

10.4.7 MSEBI Slave Mode

10.4.7.1 Slave Mode Overview

Main features on slave mode are:

- Synchronous only
- MSEBI interface width selectable from 8, 16 and 32 bits
- Support for burst access 1-4-8-16 on AHB
- 4 DMA flow control signals (external request transmission) capability
- 1 interrupt line for the detection of "end of block"
- 2 addressing mode
  - Direct
  - MMU
- Write protect bit to avoid write on the device
- 6 FIFOs of 32 bits to manage requests from MSEBI
- Prefetch capability for CPU transmit FIFO and DMA RX FIFO
- Optimized burst capability for DMA TX FIFO
- Up to 4 chip select Lines
- 6 shared registers accessed by the master of the bus
  - 1 register to manage errors
  - 1 register to manage end of block interrupt
  - 1 ID register for each chip select
    Used by the master to determine if the chip select is ready.

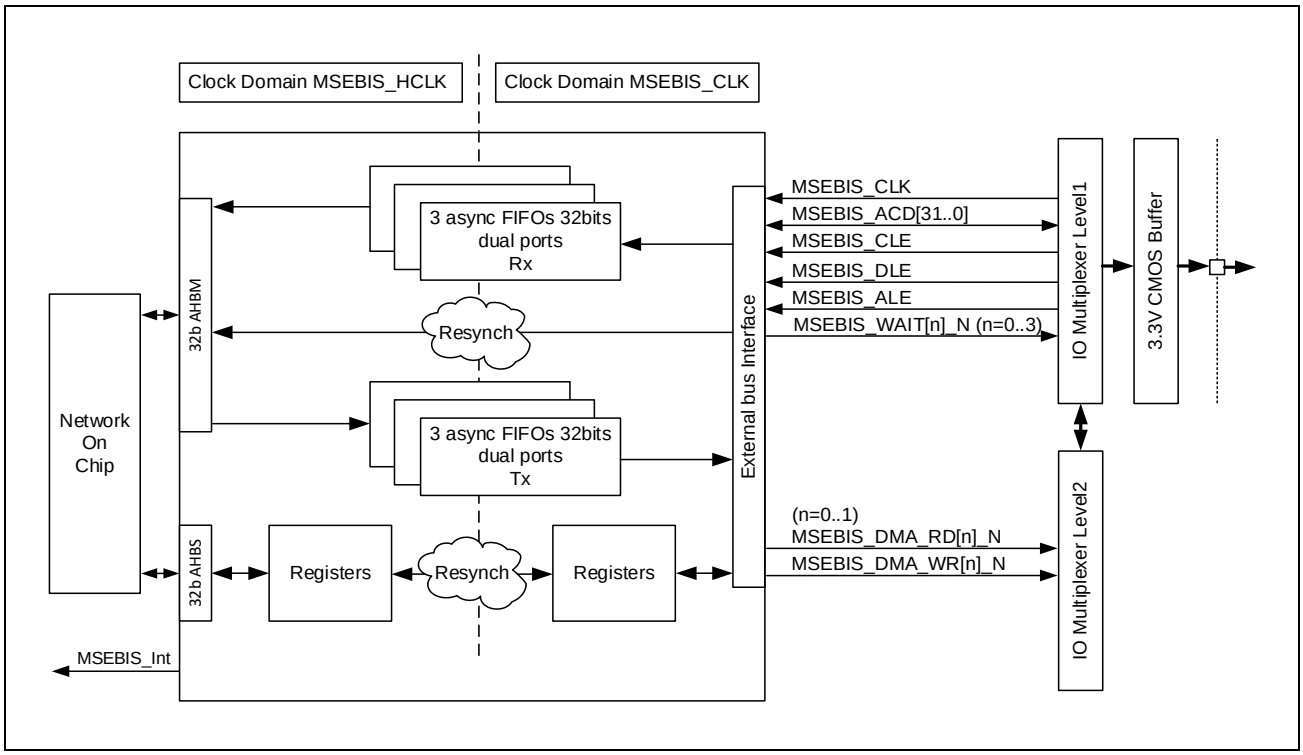

Figure 10.57 Slave Mode Overview

10.4.7.2 MSEBI Slave: Burst Mode

To manage the burst access on AHB bus, MSEBI slave interface uses the following set of rules:

- Burst mode enabled (Set bMSEBIS_BURST_ENABLE bit to 1'b1) for each MSEBI_CS[n]_N (n = 0..3).
- The burst cycle can be generated by five potential requesters:
  - CPU receive FIFO
    Contains command from the CPU part of the master of the MSEBI bus
  - DMA Transmit FIFO on CS0_N
    Contains command from the DMA TX 0 part of the master of the bus
  - DMA Transmit FIFO on CS1_N
    Contains command from the DMA TX 1 part of the master of the bus
  - DMA Receive FIFO on CS0_N
    Manage a command from the DMA RX 0 part of the master of the bus
  - DMA Receive FIFO on CS1_N
    Manage a command from the DMA RX 1 part of the master of the bus
- A specific logic is used to detect the initiator of the request. See **Section 10.4.7.3, MSEBI Slave: Detection of Request Initiator**.
- Between each request, the arbiter manages a Round Robin Priority
- Prefetch mode is only available if all of the following conditions are met:
  - Burst mode is enabled with bMSEBIS_BURST_ENABLE for each MSEBI_CS[n]_N (n = 0..3).
  - In CPU mode: bMSEBIS_BURST_SIZEMAX_CPUREAD is set to a value greater than 1.
  - In DMA mode: bMSEBIS_DMARX_MAX_BURST is set to a value greater than 1.
- The max burst size is configured by:
  - CPU: bMSEBIS_BURST_SIZEMAX_CPUWRITE and bMSEBIS_BURST_SIZEMAX_CPUREAD registers.
    CPU prefetch: bMSEBIS_BURST_SIZEMAX_CPUREAD is also used to control the maximum number of words to be read during a prefetch operation.
  - DMA: bMSEBIS_DMARX_MAX_BURST (rMSEBIS_DMARDLR_CS[n]_N register, n = 0..1) and bMSEBIS_DMATX_MAX_BURST (rMSEBIS_DMATDLR_CS[n]_N register, n = 0..1)
    DMA prefetch: bMSEBIS_DMARX_MAX_BURST (rMSEBIS_DMARDLR_CS[n]_N register, n = 0..1) is also used to control the maximum number of words to be read during a prefetch operation.
- MSEBI will generate a write burst only
- FIFO size
  - CPU: 32 words × 32 bits as Transmit and Receive FIFO
  - DMA: For each MSEBI_CS[n]_N (n = 0..1), 32 words × 32 bits as Transmit and Receive FIFO
- An AHB burst access will never cross a 1 kB boundary.

(1) Slave CPU FIFOs

Receive FIFO:

- All transaction from MSEBI bus with id=CPU are stored on the receive FIFO.
- Order of transaction is strictly respected.
- The MSEBI slave controller is always trying (depending on the contents of the FIFO) to generate a burst on the AHB bus with a size as large as possible (Limited by burst size max allowed: bMSEBIS_BURST_SIZEMAX_CPUWRITE).
  - MSEBI Slave controller will generate a burst only
  - Address is supposed to be linearly incremented by 4 and never cross a 1 kB boundary.
- Access on MSEBI_CS[n]_N (n = 0..3) is allowed only if dedicated bMSEBIS_CS_ENABLE bit is set. If not (bit cleared), the access on MSEBI_CS[n]_N is ignored

Transmit FIFO:

- Data from read requests are stored on the transmit FIFO.
- On Read command from the MSEBI bus, if prefetch is enabled and when the FIFO is empty after the current read request, the bus interface generates a burst access in prefetch mode (Limited by parameter: bMSEBIS_BURST_SIZEMAX_CPUREAD) to anticipate the following MSEBI read request.
  - MSEBI Slave controller will generate a prefetch operation only
  - Address is supposed to be linearly incremented by 4 and never cross a 1 kB boundary.
- Content of the FIFO is flushed when a read request appends with a non-incremental address
- Access on MSEBI_CS[n]_N (n = 0..3) is allowed only if dedicated bMSEBIS_CS_ENABLE bit is set. If not (bit cleared), the access on MSEBI_CS[n]_N is ignored

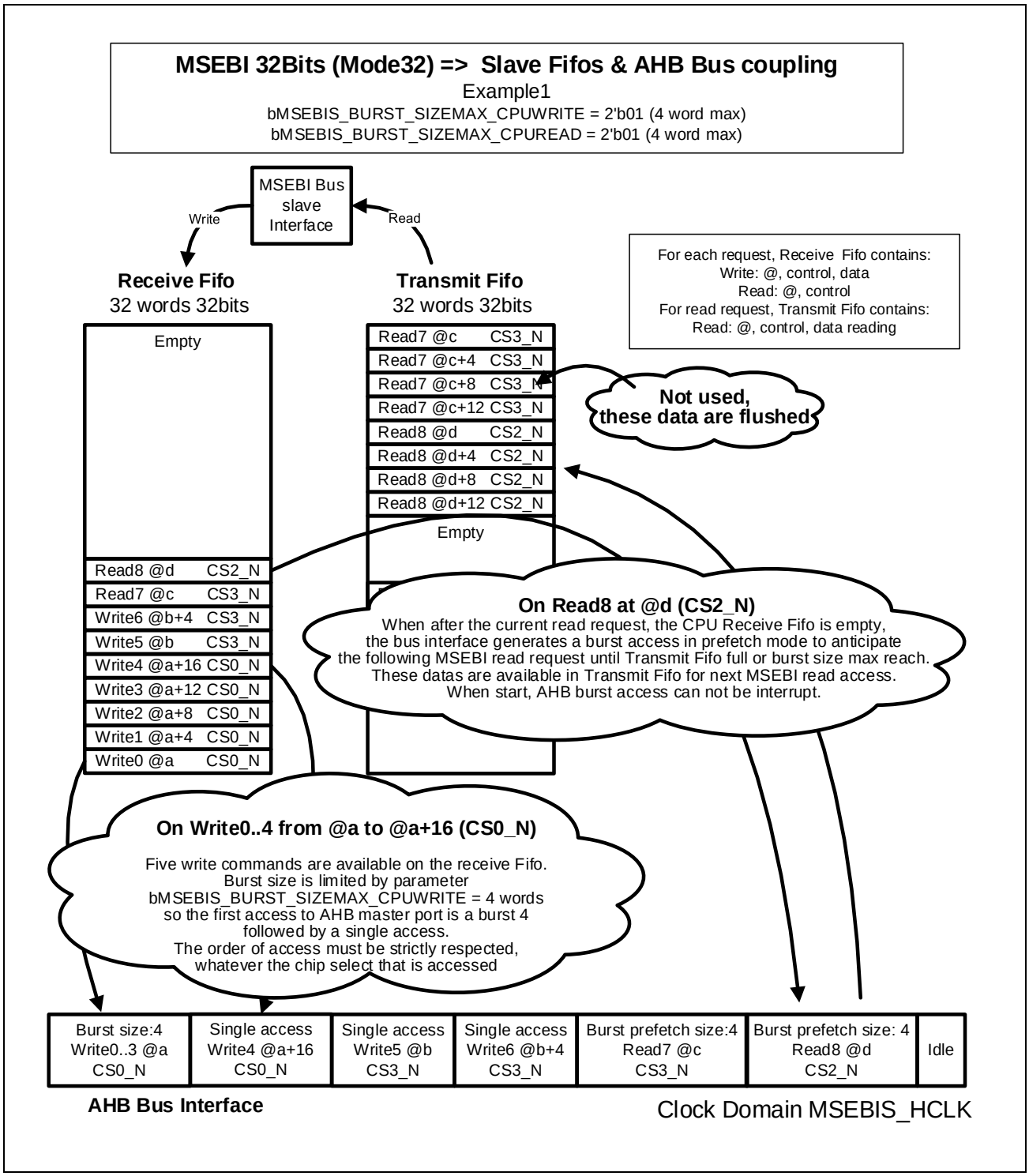

Figure 10.58 MSEBI Slave CPU FIFOs Example 1

NOTE

On the figure above, all accesses are supposed to be aligned 32 bits.

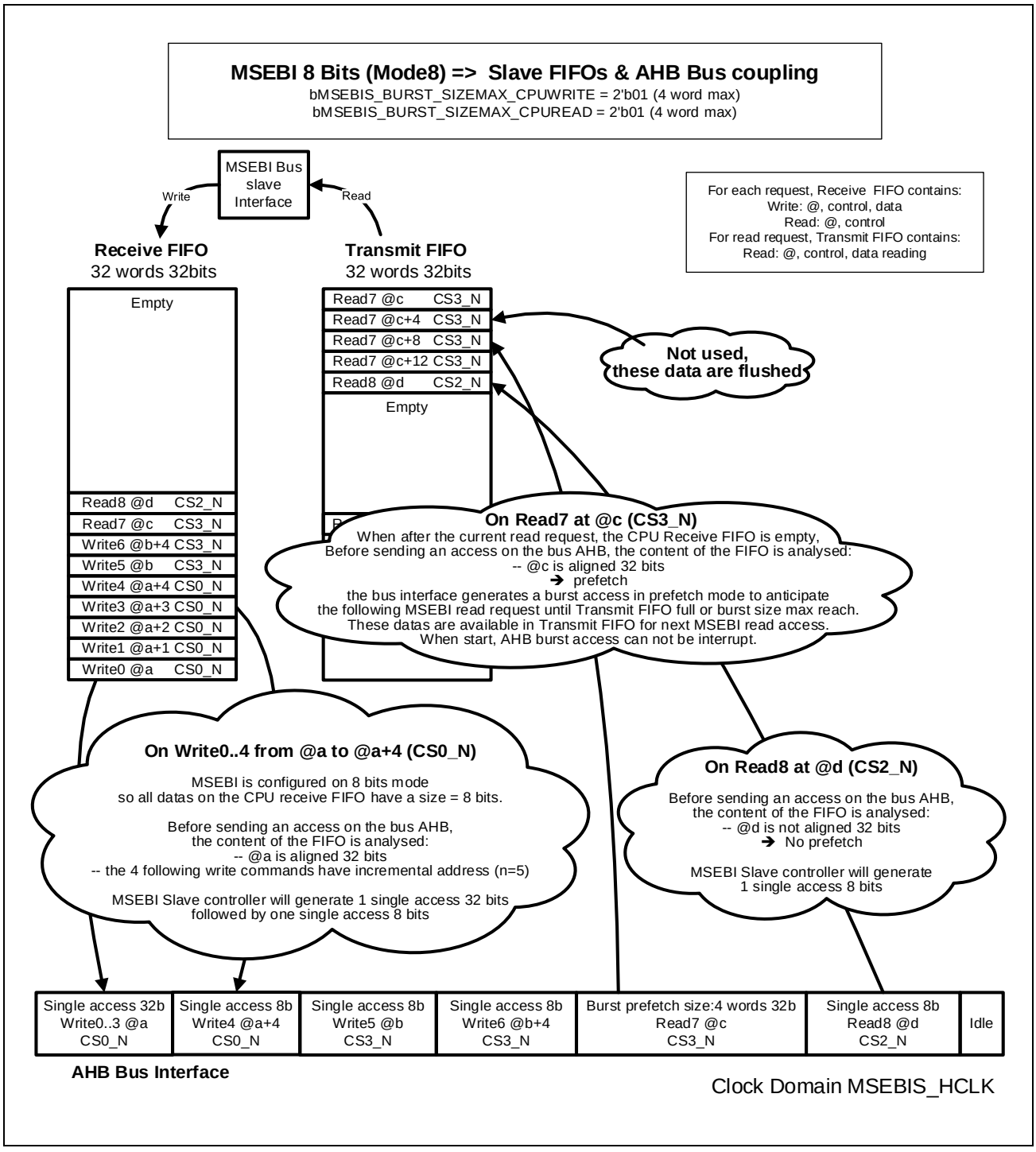

Figure 10.59 MSEBI Slave CPU FIFOs Example 1 Mode8

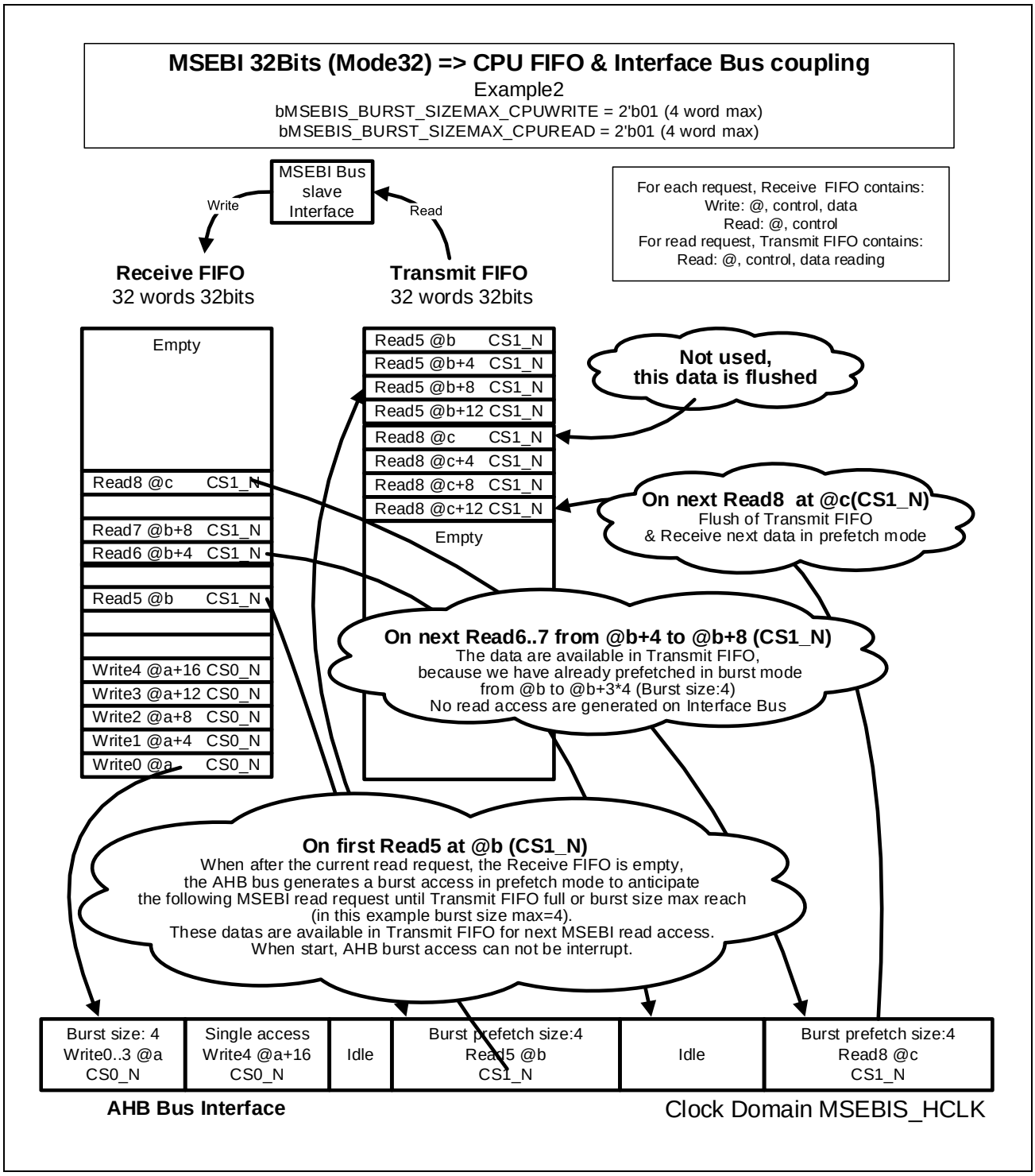

Figure 10.60 MSEBI Slave CPU FIFOs Example 2

NOTE

On the figure above, all accesses are supposed to be aligned 32 bits.

(2) Slave DMA FIFOs

(a) Slave DMA FIFOs for Requests from Master MSEBI DMA TX FIFOs

- All write request from MSEBI bus with id=DMA are stored on a DMA Transmit FIFO.
- Order of transaction is strictly respected.
- The MSEBI slave controller is always trying (depending on the contents of the FIFO) to generate a burst on the AHB bus with a size as large as possible (Limited by burst size max allowed: bMSEBIS_DMATX_MAX_BURST (rMSEBIS_DMATDLR_CS[n]_N register, n = 0..1)).
  - MSEBI Slave controller will generate a burst only
  - Address is supposed to be linearly incremented by 1/2/4 and never cross a 1 kB boundary.
- MSEBI can optimize burst request by waiting to have enough requests on the FIFO to send a burst size (bMSEBIS_DMATX_MAX_BURST (rMSEBIS_DMATDLR_CS[n]_N register, n = 0..1)). This mode is selected by bMSEBIS_DMATX_OPT_BURST (rMSEBIS_DMATDLR_CS[n]_N register, n = 0..1).
- Content of the FIFO is read until it becomes empty when bMSEBIS_DMATX_ENABLE in rMSEBIS_DMATX_REQ_CS[n]_N register (n = 0..1) is set to 0 .

CAUTION

When MSEBI slave is optimized for a low occupation of the NoC (bMSEBIS_DMATX_OPT_BURST = 1), write request are grouped on DMA TX FIFOs. If the size of the block to write is not a multiple of burst size word 32 bits, the block transfer will not be completed until the “end of block event” is received (optimization is considered disabled on the channel after the reception of the “end of block event” and until the FIFO is empty: see **Section 10.4.5.3, MSEBI Interrupt: End of Block Detection by the Slave**).

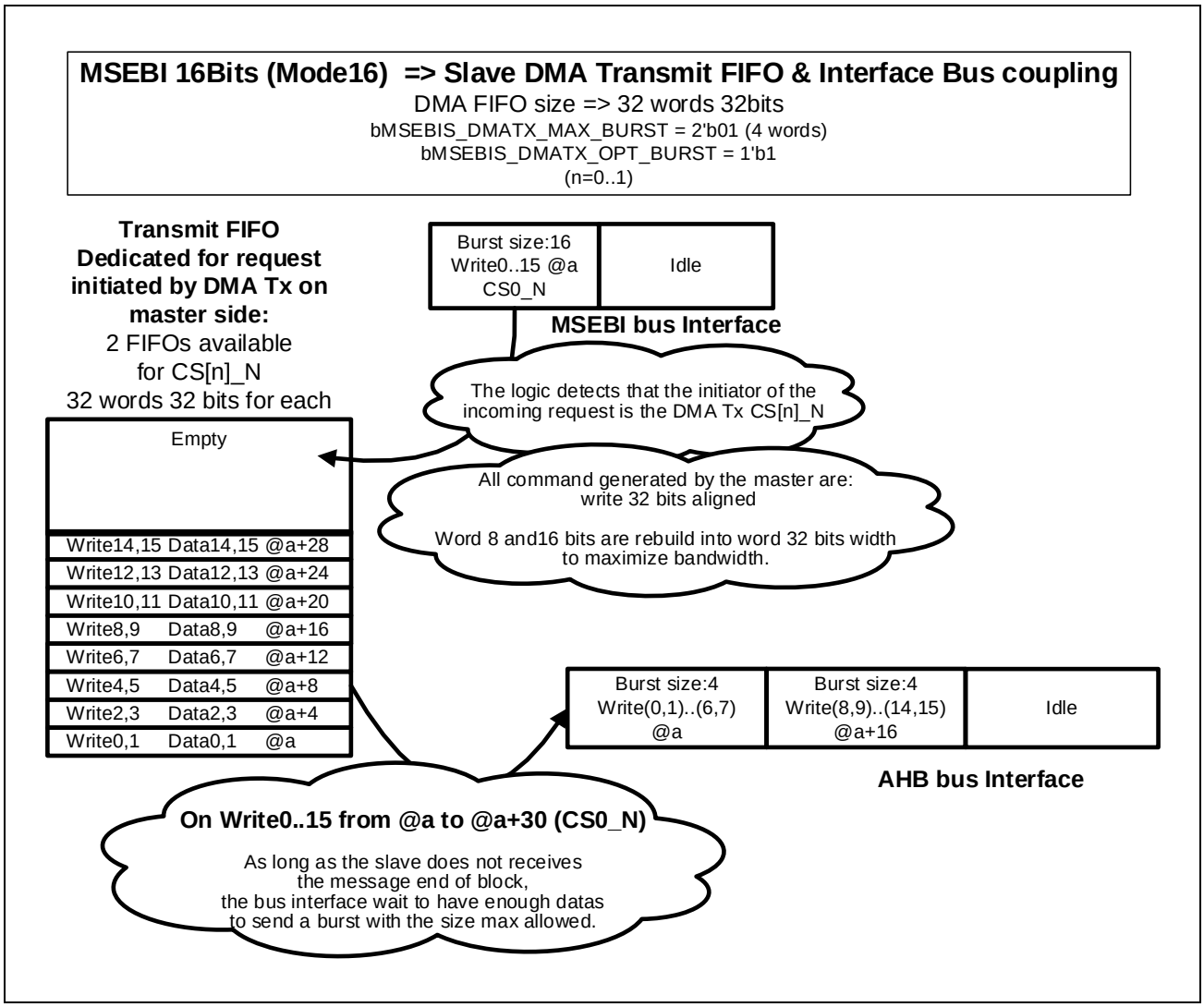

Figure 10.61 Slave DMA FIFOs for Requests from Master MSEBI DMA TX FIFOs

(b) Slave DMA FIFOs for Requests from Master MSEBI DMA RX FIFOs

- All read request from MSEBI bus with id=DMA are stored on a DMA Receive FIFO.
- Order of transaction is strictly respected.
- On Read command from the MSEBI bus, if prefetch is enabled and when the FIFO is empty after the current read request, the bus interface generates a burst access in prefetch mode (Limited by parameter: bMSEBIS_DMARX_MAX_BURST on rMSEBIS_DMARDLR_CS[n]_N register (n = 0..1)) to anticipate the following MSEBI read request on this channel.
  - MSEBI Slave controller will generate a prefetch operation only
  - Address is supposed to be linearly incremented by 4 and never cross a 1 kB boundary.
- After a prefetch operation, MSEBI slave controller will send another burst in prefetch mode as soon as there are enough places on the FIFO to generate a read burst on AHB bus (to avoid wait cycle on MSEBI bus between two prefetch operations when FIFO is empty). This type of prefetch is generated when all of the following conditions are met:
  - A first prefetch operation has been generated.
  - FIFO has not been flushed since last prefetch operation.
  - There are at least bMSEBIS_DMARX_MAX_BURST (rMSEBIS_DMARDLR_CS[n]_N register, n = 0..1) places available on the FIFO.
- Content of the FIFO is flushed when a read request appends with a non-incremental address or when bMSEBIS_DMARX_ENABLE in the rMSEBIS_DMARX_REQ_CS[n]_N register (n = 0..1) is set to 0 .

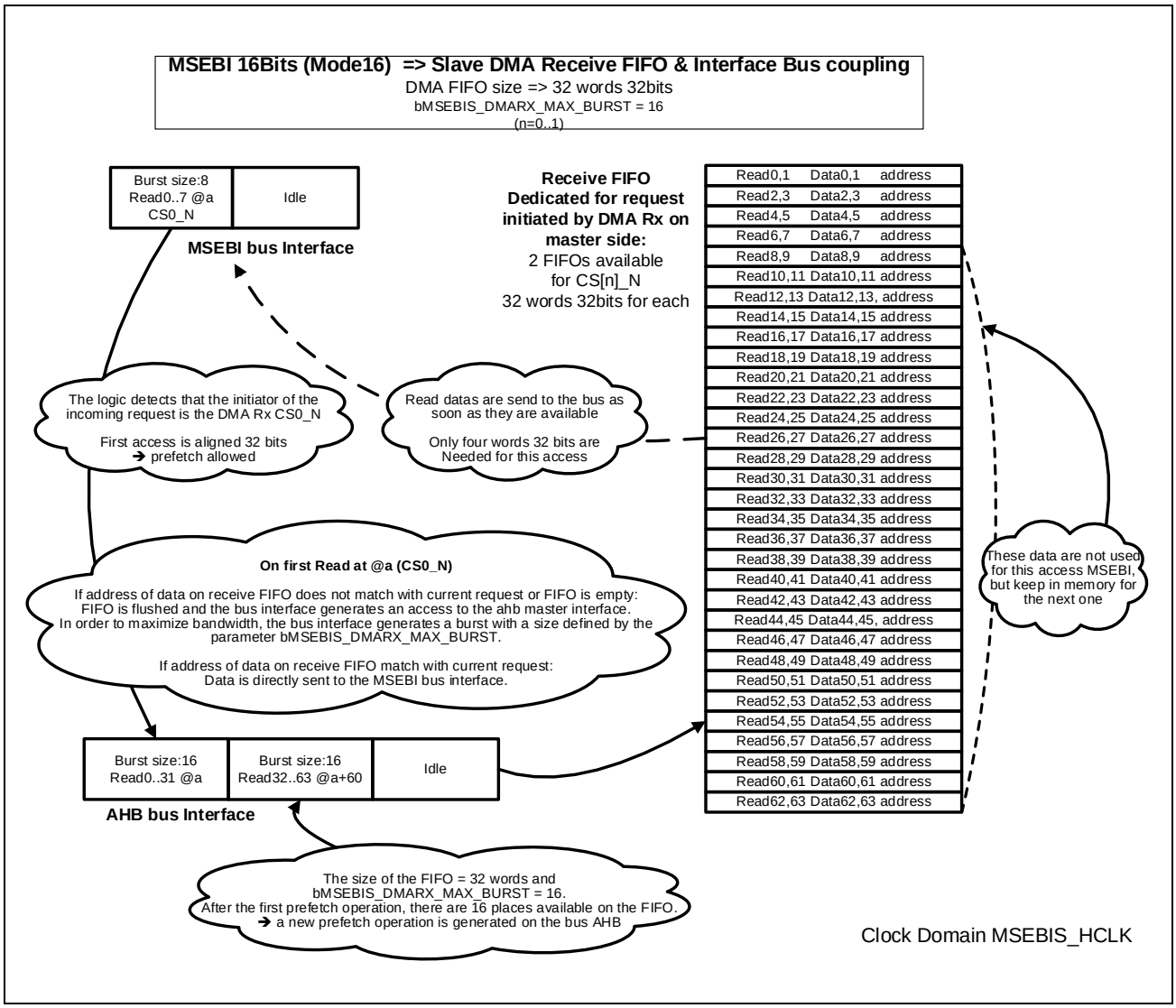

Figure 10.62 Slave DMA FIFOs for Requests from Master MSEBI DMA RX FIFOs

(3) Slave DMA Flow Control Signals

MSEBI Slave controller can manage DMA flow control signals:

- MSEBIS_DMA_RD0_N
- MSEBIS_DMA_RD1_N
- MSEBIS_DMA_WR0_N
- MSEBIS_DMA_WR1_N

Flow control of MSEBIS_DMA_RD0_N and MSEBIS_DMA_RD1_N pins can be enabled by bMSEBIS_DMARX_FLOW_CTRL field in rMSEBIS_DMARDLR_CS[n]_N (n = 0..1) register.

Flow control of MSEBIS_DMA_WR0_N and MSEBIS_DMA_WR1_N pins can be enabled by bMSEBIS_DMATX_FLOW_CTRL field in rMSEBIS_DMATDLR_CS[n]_N (n = 0..1) register.

MSEBI controller provides one DMA flow control signal for each DMA FIFO.

When flow control is enabled, MSEBIS_DMA_RD[n]_N and MSEBIS_DMA_WR[n]_N pins are driven by following bits in rMSEBIS_DMARX_REQ_CS[n]_N and rMSEBIS_DMATX_REQ_CS[n]_N (n = 0..1) registers:

- bMSEBIS_DMARX_FORCE
- bMSEBIS_DMATX_FORCE

For more information about the behavior of flow control signals, please refer to **Section 10.4.6.3(2), External DMA Request** on the master part of the documentation.

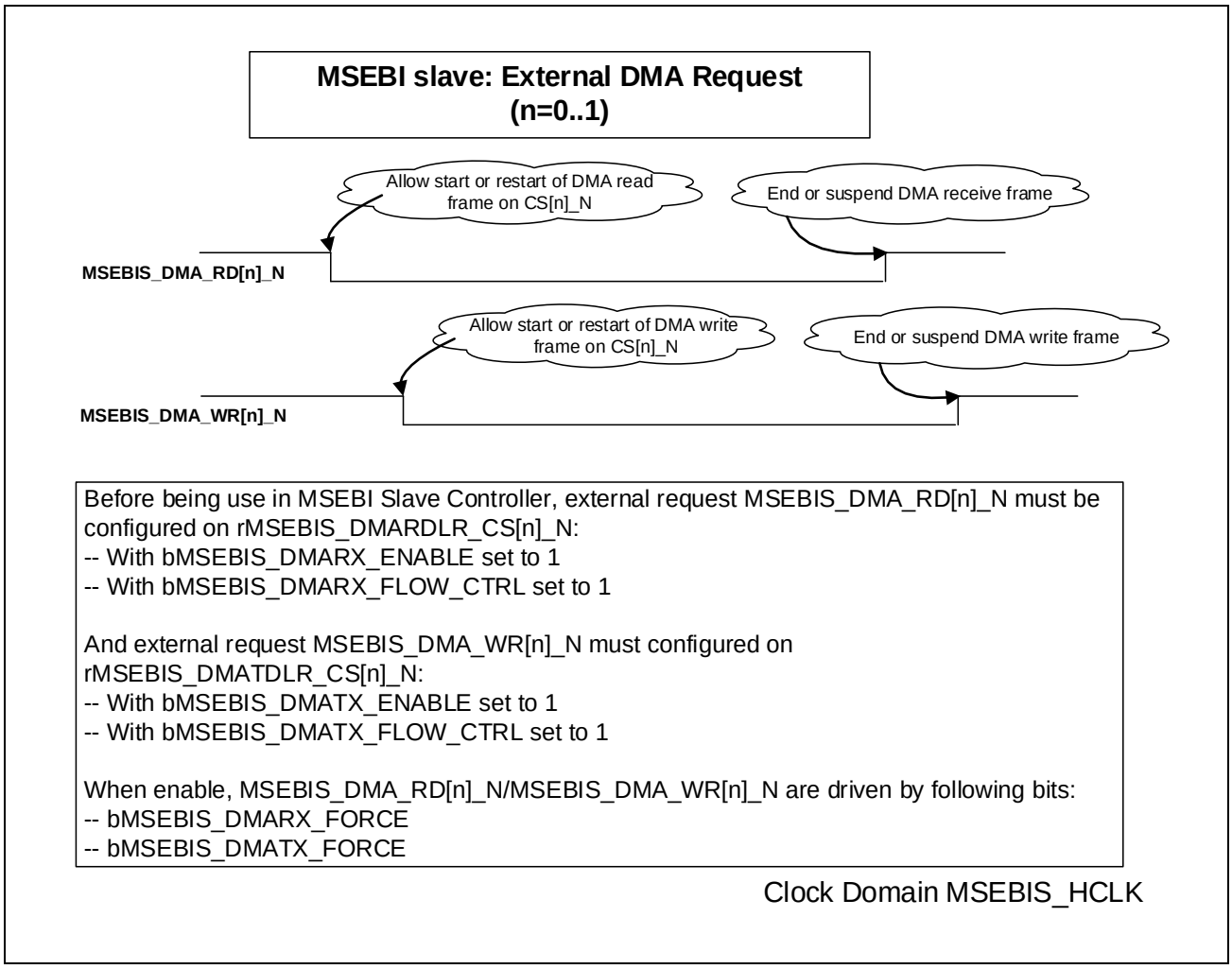

Figure 10.63 MSEBI Slave: External DMA Request

(4) Slave FIFOs Arbiter: Round Robin

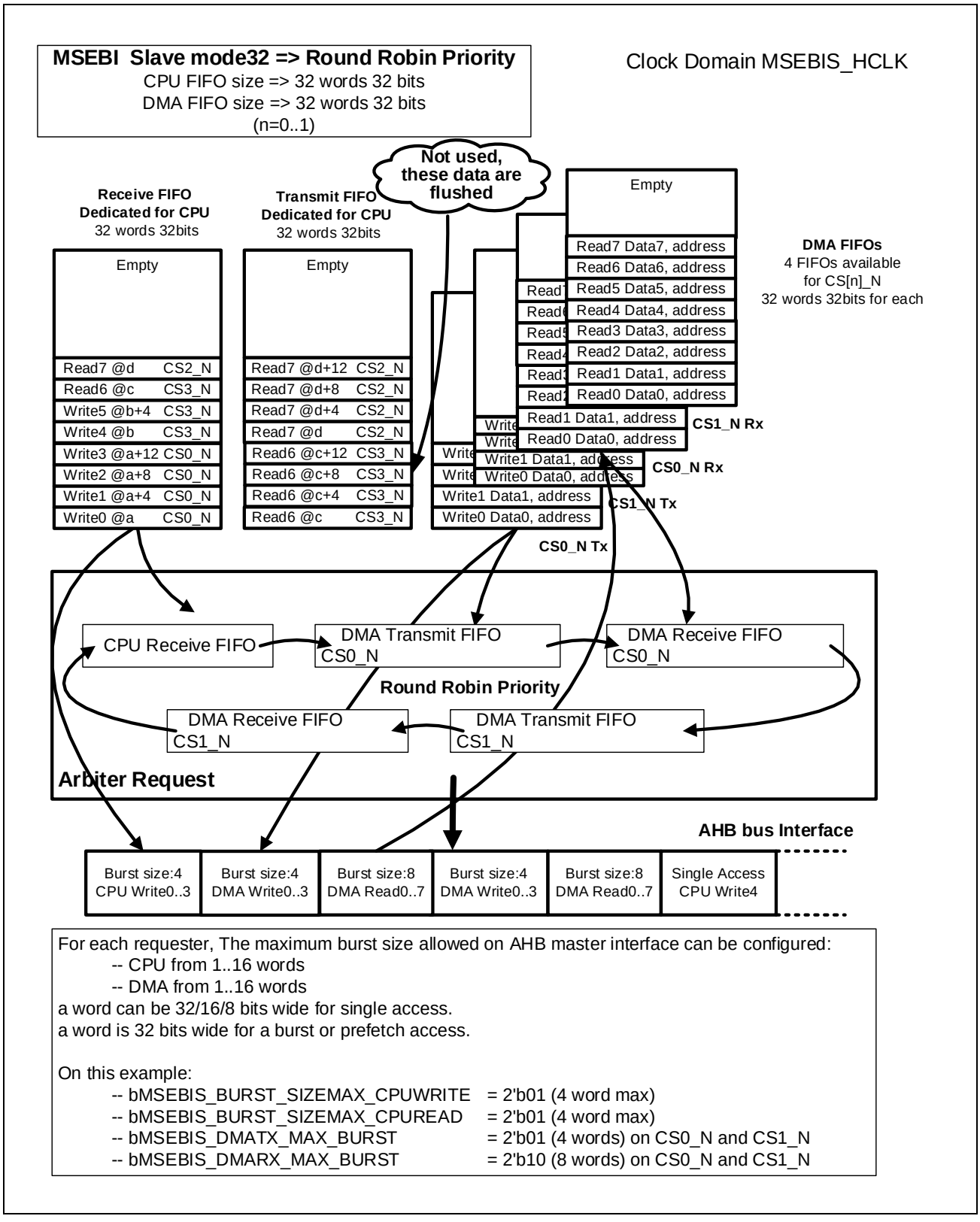

Figure 10.64 Slave FIFOs Arbiter: Round Robin

10.4.7.3 MSEBI Slave: Detection of Request Initiator

The detection of the initiator of the request is done with the MSEBI_DMA_N bit during the CLE phase.

This information is used to route the incoming request to the correct FIFO in order to optimize transfers (grouping the request from the same initiator increase the probability to generate a burst).

See:

- For **Table 10.4, MSEBI Mode32, Multiplexer Function on ACD31..0**.
- For **Table 10.6, MSEBI Mode16, Multiplexer Function on ACD15..0**.
- For **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

Table 10.47 Slave Detection of Request Initiator

Initiator	MSEBI_DMA_N	MSEBI_CS1_N, MSEBI_CS0_N	Request Type	Targeted FIFO
CPU	1	X, X	X	CPU receive
DMA RX CS0	0	1, 0	read	DMA RX CS0
DMA TX CS0	0	1, 0	write	DMA TX CS0
DMA RX CS1	0	0, 1	read	DMA RX CS1
DMA TX CS1	0	0, 1	write	DMA TX CS1

10.4.7.4 MSEBI Slave: Register Access by Master

MSEBI master on the bus can access six slave's shared registers. The MSEBI access corresponding to MSEBI_CS[n]_N (n = 0..3) is generated according to the address of each register.

- rMSEBIS_INT
- rMSEBIS_STATUS
- rMSEBIS_ID_CS[n]_N (n = 0..3)

Access to the shared registers of a slave where MSEBI_CS[n]_N is an active chip select is available when following conditions are reach:

- The command set simultaneously MSEBI_CSREG_N and MSEBI_CS[n]_N to 0 during the phase MSEBI_CLE of an access to the chip select [n].
- The request is managed on the device where the chip select is active:
  - bMSEBIS_CS_ENABLE is set to 1.

The following table explains the conditions on MSEBI_CSREG_N and MSEBI_CS[n]_N to access the slave's shared registers.

Table 10.48 MSEBI Slave Shared Register Access

MSEBI_CS[n]_N	MSEBI_CSREG_N	CS[n]_N Access Type
0	0	Access CS[n]_N shared registers
0	1	Access CS[n]_N memory space
1	0	Reserved
1	1	No access

See table below:

- For **Table 10.4, MSEBI Mode32, Multiplexer Function on ACD31..0**.
- For **Table 10.6, MSEBI Mode16, Multiplexer Function on ACD15..0**.
- For **Table 10.8, MSEBI Mode8, Multiplexer Function on ACD7..0**.

10.4.7.5 MSEBI Slave: Chip select Configuration Status

All chip select of MSEBI slave controller can present their configuration status to the master of the MSEBI bus.
(n = 0..3)

- This feature is mainly used after a change of the MSEBI bus configuration to inform the master of the MSEBI bus that the chip selects (on slave device) are ready to receive requests from MSEBI bus.
- Status is available by reading the slave's shared rMSEBIS_ID_CS[n]_N registers through the MSEBI bus
  - For more information about slave's shared registers, see **Section 10.4.7.4, MSEBI Slave: Register Access by Master**.
- The master of the MSEBI bus can access to the slave's shared registers with all chip select.
  - With MSEBI_CSREG_N and MSEBI_CS[n]_N driven to 1'b0
  - See **Section 10.4.7.4, MSEBI Slave: Register Access by Master**.
- A read of the correct value on the corresponding slave's shared register validate the configuration of the chip select.
  - Read 0x1234_FEDn on rMSEBIS_ID_CS[n]_N validates the configuration of the chip select [n].
  - Any others value indicates that the chip select [n] is not ready for communication.

10.4.7.6 MSEBI Slave: Addressing Mode

MSEBI slave can manage the address of the new incoming transaction in two different ways:

- Direct access: address decoded by the slave is directly used to generate a new transaction on the AHB master port.
- MMU mode access: A base offset (aligned 4KB) can be added to the address decoded by the slave before being used to generate a new transaction on the AHB master port.

See following registers:

- **Section 10.3.3.4, rMSEBIS_MMU_ADDR_MASK_CS[n]_N — MMU Address Mask Register (n = 0..3)** and **Section 10.3.3.3, rMSEBIS_MMU_ADDR_CS[n]_N — MMU Base Address Register (n = 0..3)**.

MMU mode increases performance as it may save some ALE phases.

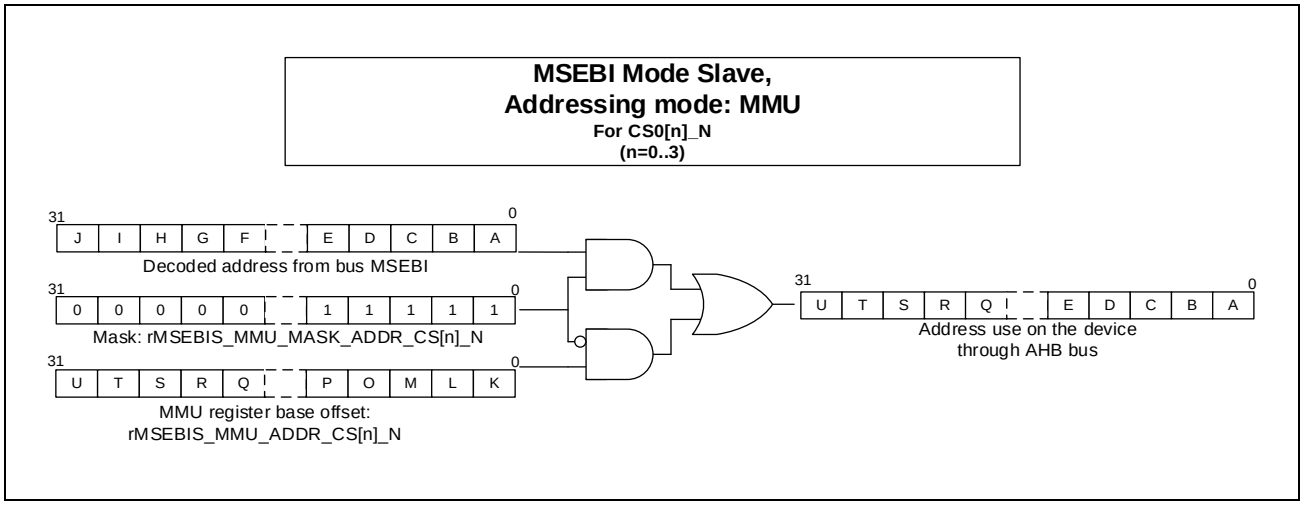

Figure 10.65 Slave Addressing on MMU Mode

The selection of the addressing mode is made with bMSEBIS_ADDR_MODE bit

10.4.7.7 MSEBI Slave: Write Protect

If this bit is set, all writes on the slave memory space are forbidden.

Setting of write protect is done with the bMSEBIS_WEN and can be managed separately for each MSEBI_CS[n]_N (n = 0..3).

If a write request arrives from the MSEBI on a chip select where write protect is enable (bMSEBIS_WEN = 0), an error flag (bMSEBIS_ERROR_WEN) is set on the corresponding rMSEBIS_STATUS register reachable by the master of the bus.

bMSEBIS_ERROR_WEN is clear when the master of the bus writes 1'b1 to the corresponding bit in the rMSEBIS_STATUS register.

rMSEBIS_STATUS register cannot be access by the CPU. It is reserved for the MSEBI master bus accesses.

10.4.7.8 MSEBI Slave: Configuration Registers & Synchronization

All registers are clocked on the AHB clock domain, MSEBIS_HCLK, but some configuration fields (phases delay, routing capability, wait configuration, etc.) are on the MSEBI clock domain, MSEBIS_CLK.

For a configuration update on all registers managed by MSEBI_CS[n]_N, these fields need to be synchronized as follows:

- On the first AHB access, bMSEBIS_CS_ENABLE is cleared by CPU.
- Until the end of the synchronization mechanism.
  - Data from AHB bus is not copied on MSEBI slave registers
  - AHB is in wait state (delay max controlled by timeout)
- Once the synchronization mechanism completes, the write is enabled on the configuration registers and the value of the fields to synchronize on MSEBIS_CLK clock are set to a default value (to avoid side effect).
- After the end of the configuration from AHB (set bMSEBIS_CS_ENABLE), write access to registers are locked and the value of the fields to synchronize on MSEBIS_CLK clock are driven by registers.

NOTE

Configuration registers managed on a different MSEBIS_CS[m]_N with m < > n are not changed, and all accesses in MSEBIS_CS[m]_N continue to execute normally

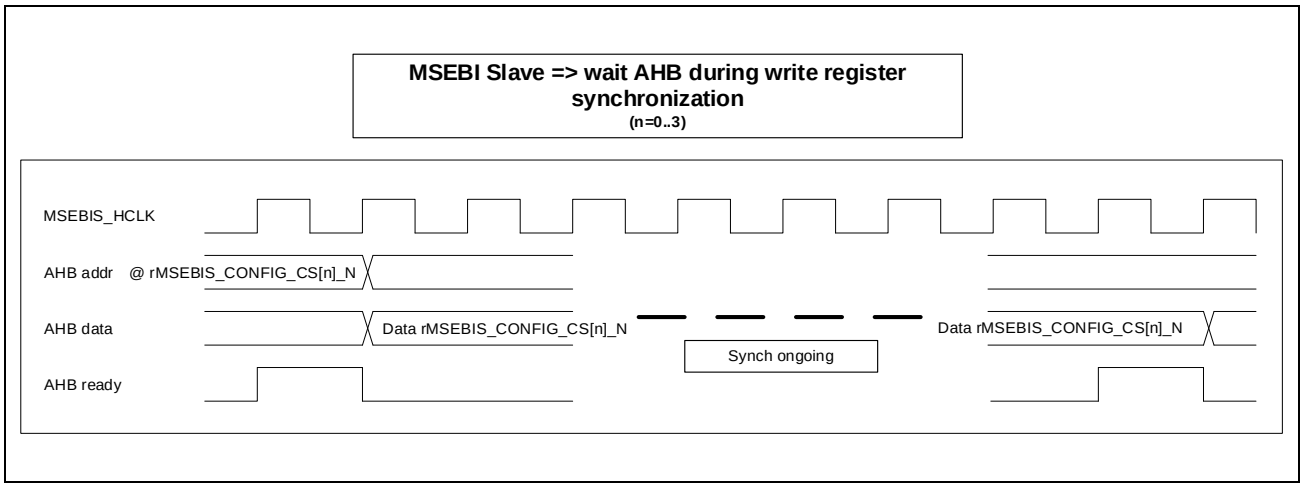

Figure 10.66 Wait AHB during Register Synchronization Mechanism

10.5 Usage Notes

In case DMA controller goes down during a DMA transfer, a recovery procedure shall be implemented:

1. Stop current DMA transfers in MSEBI Master
2. Reset burst size value in DMA RX control register (rMSEBIM_RDMACR_CS[n]_N n = 0..1) to allow single access only
3. Reset DMARDLR registers (rMSEBIM_DMARDLR_CS[n]_N n = 0..1)
4. Restart DMA controller with initial parameters if possible to unlock DMA MSEBI FIFOs. Burst size value (SRC_MSIZE or DEST_MSIZE) is tied to 1 (single access)
5. Wait and check DMA receive FIFOs are empty
6. Disable/stop DMA controller
7. Restart DMA transfers in MSEBI Master

REVISION HISTORY	RZ/N1D Group, RZ/N1S Group, RZ/N1L Group User's Manual: Peripherals

Rev.	Date	Description	
		Page	Summary
0.50	Jun 30, 2017	—	First Edition issued
0.80	Oct 31, 2017	20 to 21	1.1, description, modified
		70, 71	1.4.30, Table 1.38, revised
		72, 73	1.4.31, Table 1.39, revised
		75	1.5.1.2, chapter title, corrected
		81	1.5.1.7, Figure 1.6: figure title, corrected
		90 to 95	1.5.1.10, description, modified
		99 to 100	1.6, chapter, deleted
		99	2.1, description, revised
		108	2.4.1, "Value after reset" of b2-b0, corrected (0 → 7)
		134, 135	2.4.24, Table 2.30, revised
		136, 137	2.4.25, Table 2.31, revised
		173, 174, 210	3.3 and 3.4.35, I2C SS, Register Name: FS Spike Suppression Limit → I2C Sm, Fm Spike Suppression Limit, corrected
		223	3.6.2.1, Table 3.40: Note, added
		229 to 237	4.3 and 4.4, Register Name, modified (Port GPIO Port A Data Ouput Register → GPIO Port A Data Output Register, Port GPIO Port A Data Direction Register → GPIO Port A Data Direction Register, Port GPIO Port B Data Ouput Register → GPIO Port B Data Output Register, Port GPIO Port B Data Direction Register → GPIO Port B Data Direction Register, Port GPIO Port A Data Input Register → GPIO Port A Data Input Register, Port GPIO Port B Data Input Register → GPIO Port B Data Input Register)
		231 and 236	4.4.1, 4.4.2 and 4.4.12, the external pins GPIO[n] → the external pins BGPIO[m]A[n], corrected
		232 to 237	4.4.3, 4.4.4 and 4.4.13, the external pins GPIO[n] → the external pins BGPIO[m]B[n], corrected
		239	4.5.1.3, Figure 4.4, revised (CortexA7 → Cortex-A7)
		241	4.5.1.4, Table 4.19: BGPIO Pins Name, corrected (BGPIO1a → BGPIO1A, BGPIO1b → BGPIO1B, BGPIO2a → BGPIO2A, BGPIO2b → BGPIO2B, BGPIO3a → BGPIO3A, BGPIO3b → BGPIO3B)
		241	4.5.1.4, Table 4.20: Status Pins of header line, corrected (GPIO[m]a → BGPIO[m]A, GPIO[m]b → BGPIO[m]B)
		243	5.1, description, revised
		244	5.2, Timer[m]_Int[7:0] → TIMER[m]_Int[7:0], corrected
		245	5.3.1 and 5.3.2, Timer[n] → Sub-timer[n], modified
		246 to 253	5.4.1 to 5.4.9, Timer[n] → Sub-timer[n], modified
		249	5.4.5, Table 5.7: bTimerMaskInt: Function, modified
		251, 252	5.4.7 and 5.4.8, iTimerInt → TIMER_Int[n], timer → Sub-timer, corrected
		263	5.5.4, description, revised
		265	5.6, description, revised
		269	6.3, Register Name: Error Warning Interrupt Register → Error Warning Limit Register, corrected
		269	6.3.1, Table 6.1: Address of rCan_ReceiveFifo, corrected (5210 4080h → 5210 4080h to 5210 417Ch)
		269	6.3.1, Table 6.1: Address of rCan_RdTransmitBuffer, corrected (5210 4180h to 5120 41B0h → 5210 4180h to 5210 41B0h)
		270	6.3.2, Table 6.2: Address of rCan_ReceiveFifo, corrected (5210 5080h → 5210 5080h to 5210 517Ch)

Rev.	Date	Description	
		Page	Summary
0.80	Oct 31, 2017	279	6.4.5, chapter title, corrected
		286	6.4.11, Register Name: Error Warning Interrupt Register → Error Warning Limit Register, corrected
		290	6.4.14, NOTE, corrected
		290	6.4.14, Address, corrected (5210 4040h - 5120 4070h (CAN1) → 5210 4040h - 5210 4070h (CAN1), 5210 5040h - 5120 5070h (CAN2) → 5210 5040h - 5210 5070h (CAN2))
		314 to 315	6.5.7, Table 6.32 to 6.35: header line, corrected (Acceptange → Acceptance)
		342	7.1, description, revised
		346 to 377	7.3 and 7.4, corrected (ADC-1 → ADC1, ADC-2 → ADC2)
		365	7.4.1.16, Table 7.18: bADC_Priority, modified
		370	7.4.1.20, Table 7.22: bADC2_Enable, modified
		409	8.3.1, chapter title, corrected
		410	8.4.1, Table 8.3: bLcd_FBP: packet mode → packed mode, corrected
		423	8.4.10, Table 8.12: bLcd_DBAR, modified
		439	8.5.2, Table 8.27 and 8.28, modified
		450, 451	8.5.9, Figure 8.7 and 8.8, figure title, corrected
		452	8.5.10, chapter title, corrected
		454	8.5.13, unpacket → unpacked, packet → packed, corrected
		457	8.5.16, packet → packed, corrected
		460	9.4.1, chapter title, corrected
		461	9.4.2, chapter title, corrected
		464	9.6, description, modified
		465, 467	10.1, description, revised
		483	10.3.1.3, Table 10.15: bMSEBIM_SINGLE_DEST_WIDTH, modified
		485	10.3.1.4, Table 10.16: bMSEBIM_SINGLE_SRC_WIDTH, modified
		496	10.3.1.11, Table 10.23: Bit Name, corrected (bMSEBIM_CS[n]_N_ROUTING_CS3_N → bMSEBIM_CS[n]N_ROUTING_CS3_N, MSEBIM_CS[n]_N_ROUTING_CS2_N → bMSEBIM_CS[n]N_ROUTING_CS2_N)
		497	10.3.1.11, Table 10.23: bMSEBIM_MODE_WAIT, corrected (MSEBIM_WAIT[n] → MSEBIM_WAIT[n]_N, MSEBIM_WAIT0[n]_N → MSEBIM_WAIT[n]_N)
		591	10.4.6.1, description, revised
		603, 605	10.4.6.3, description, corrected
		607	10.4.6.3(2), description, corrected
		608	10.4.6.3(2), Figure 10.55, corrected
		613	10.4.7.1, description, revised
		624	10.4.7.2(3), Figure 10.65, corrected
		631	10.5, MSEBIS_WAIT_[n]_N → MSEBIS_WAIT[n]_N, corrected
0.90	Dec 28, 2017	all	All sections, corrected English spelling and syntax errors
		20, 226, 266	1.1, 4.1, and 6.1, add trademarks
		39	1.4.7, Table 1.15: Function of bUart_STOP, revised
		53	1.4.16, Table 1.24: Function of bUart_BUSY, revised
		56	1.4.19, Table 1.27: Function of bUart_UR, revised
		67	1.4.28, Table 1.36: Function of bUart_EnableDE, external pads → external pin, modified
		71	1.4.30, Table 1.38: Function of bUart_TDMAE, Uart_DEST_BURST_SIZE → bUart_DEST_BURST_SIZE
		85, 86	1.5.1.9(3), Figure 1.9 and 1.10, Size block → Block size, corrected
		86	1.5.1.9(3), description, revised

Rev.	Date	Description	
		Page	Summary
0.90	Dec 28, 2017	90	1.5.1.10, description, revised
		97	1.5.1.11(1), description, external pad → external pin, modified
		99 to 170	Section 2, SPI (indicating the module name) → SPI controller, revised
		108	2.4.1, Table 2.7: bSpi_SLV_OE, high impedance → floating, corrected
		109, 146, 163, 164, 168, 169	2.4.1, 2.5.8, 2.5.8.2 (chapter title), 2.6.1.1, and 2.6.1.3, Transmit Mode Only → Transmit Only Mode, corrected
		109, 141, 146, 163, 164, 168	2.4.1, 2.5.4, 2.5.8, 2.5.8.3 (chapter title), 2.6.1.1, and 2.6.1.3, Receive Mode Only → Receive Only Mode, corrected
		133	2.4.23, Table 2.29: Function of bSpi_RX_Sample_Delay, revised
		143	2.5.5, description, corrected
		144	2.5.6, description, revised
		147	2.5.8.4, master configurations → master mode, modified
		148 to 150	2.5.9, description, revised
		152	2.5.11, description, no data are present (high impedance) → no data are present, corrected
		156	2.5.11, description, no data are driven (high impedance) → no data are driven, corrected
		157	2.5.11, SPI master configuration → SPI master, SPI slave configuration → SPI slave, modified
		160	2.5.12.3, Figure 2.23 and 2.24, Size block → Block size, corrected
		161	2.5.12.3, description, revised
		164	2.6.1.1 5., see Warning on polling Busy → see CAUTION of 2.6.1, corrected
		166	2.6.1.2 5., see Warning on polling Busy → see CAUTION of 2.6.1, corrected
		171 to 225	Section 3, I2C (indicating the module name) → I2C controller, revised
		171	3.1, description, revised
		173, 174, 201	3.3 and 3.4, Register Name of IC_RXFLR, Receive FIFO Level Register → I2C Receive FIFO Level Register, changed
		186	3.4.10, description, revised. 3.4.10, Table 3.12: Function, revised
		215	3.5.1.4, chapter title, corrected
		220	3.5.4.1, M_TX_ABRT → TX_ABRT, corrected
		221	3.6.1, I2C_clock → I2C_SCLK, corrected
		225	3.6.3.1, below maximum value → above maximum value,corrected
		226, 227	4.1, Figure 4.1 to 4.3, corrected
		238	4.5.1.1, CAUTION, revised
		239	4.5.1.3 Figure 4.4, corrected
		239, 240	4.5.1.4, description, revised. 4.5.1.4, Figure 4.5, corrected
		241	4.5.1.4, Table 4.19, header line, corrected. 4.5.1.4, Table 4.20, revised
		242	4.6.1, CAUTION, revised
		256	5.4.12, Table 5.14: Function, revised
		259	5.5.1, description, revised
		259	5.5.2, description, revised.
		260, 261	5.5.2, Figure 5.2 to 5.4, figure title, modified
		266 to 341	Section 6, CAN (indicating the module name) → CAN controller, revised
		266	6.1, description, revised
		269, 270, 298	6.3 and 6.4, Register Name of rCan_SyncTransmitBuffer, Sync Transmit Buffer Register → Sync Frame Transmit Buffer Register, changed
		271, 272	6.4.1, Table 6.3: Function of bCan_SM and bCan_RM, revised
		273, 274	6.4.2, Table 6.4: Function, revised

Rev.	Date	Description	
		Page	Summary
0.90	Dec 28, 2017	275, 276	6.4.3, Table 6.5: Function of bCan_TS, bCan_RS, bCan_TBS, bCan_DOS, and bCan_RBS, revised
		277, 278	6.4.4, Table 6.6: Function, revised
		294	6.4.18 Table 6.20, Function, revised
		298	6.4.22, Table 6.24, Function, revised
		298, 299, 301, 306	6.4.22, 6.4.23, 6.4.24, and 6.4.27, loss arbitration → arbitration loss, modified
		305	6.4.27, description, revised
		305	6.4.27, Table 6.29: Function of bCan_TimerOnlyMode, Reduce mode → CAN Controller is running in TimerOnlyMode, revised
		305	6.4.27, Table 6.29: Function of bCan_TimerOnlyIfBusOff, CAN Controller is running in full mode → Disable all the “Sync frame” system if the CAN Controller detect a “Bus off” condition
		307	6.4.28, Table 6.30: Function, revised
		308	6.4.29, Table 6.31, CAN Full module → CAN module, corrected
		314, 315	6.5.7, Table 6.32 to 6.35: Receive Buffer Field colum, Not matched → No filtering, modified
		316, 319, 320, 332	6.5.8, 6.5.8.9, 6.5.8.10, and 6.5.15, Only for CAN full (with transmit “Sync frame” mechanism) → Only when using “Sync frame” transmission mechanism, corrected
		325	6.5.11.1, Table 6.38 and 6.39, (1) → *1 and (2) → *2, corrected. 6.5.11.1, first Note → Note 1, second Note → Note 2, corrected
		331	6.5.14, description, revised
		332	6.5.15, description, revised
		337, 338	6.5.15.2, description, revised
		340	6.5.16, description, revised
		356, 357	7.4.1.9, Table 7.11: Function of bADC_DMA1_RUNNING and bADC_PENDING_VC, revised
		364	7.4.1.16, Table 7.18: Function of bADC_RR_Pointer, revised
		366	7.4.1.17, Table 7.19: Function of bADC_POWER_DOWN, revised
		369	7.4.1.19, Table 7.21: Function of bADC_MASKLOCK, revised
		370	7.4.1.20, Table 7.22: Function of bADC_DMA_Request, corrected
		370, 371	7.4.1.20, Table 7.22: Function of bADC_DMA_Request, bADC2_Enable, and bADC_Continuous, revised
		371	7.4.1.20, Table 7.22: Function of bADC_TrigEnable, ADC1 Trigger Enable → Trigger Enable, corrected
		381	7.5.1, “bADC_TrigSel field to 5’h13” → “bADC_TrigSel field to 5’h14” at example of single conversion, corrected
		382	7.5.1, “on a use iADC_EOC_VC3 event” → “on a use iADC_EOC_VC4 event” at example of oversampling 4X, corrected
		384	7.5.1, bADC_TSHSAMP: 5’h12 → bADC_TSHSAMP: 5’h0C at example of sample & hold setting, corrected
		385	7.5.2, “3.3 DC characteristics” → “ADC characteristics described on Electrical Characteristics”, corrected
		393	7.5.6, bADC_TSHSAMP: 6’h6 (Minimum value) → bADC_TSHSAMP: 5’h0C
		404	7.5.10, bADC_DMA_Request bits in rADC_VC[n] registers with n = 0..1 → bADC_DMA_Request bits in rADC_VC[n] registers with n = 0..15, corrected.
		404	7.5.10, The ADC DMA channel selected depends onf bADC_DMA_Request[1] bit status → The ADC DMA channel selected depends onf bADC_DMA_Request[1:0] bit status, corrected
		405	7.5.10.1, description, revised
		410	8.4.1, Table 8.3: Function of bLcd_FBP, modified
		443	8.5.5, Table 8.30: Encoded Pixel Data → Pixel Data, corrected

Rev.	Date	Description	
		Page	Summary
0.90	Dec 28, 2017	443	8.5.5, Table 8.31 and 8.32: 32'h20, joined P0 and P1 cells, revised
		444	8.5.5, Table 8.33: 32'h40, joined P0 and P1 cells, revised
		444	8.5.5, Table 8.34: 32'h400, joined P0 and P1 cells, revised
		445	8.5.6 and 8.5.7, description, revised
		451	8.5.9, CAUTION, revised
		454	8.5.13, description, modified. CAUTION, revised
		457	8.5.16, description, revised
		458	9.2, Table 9.1, table title, corrected
		465	10.1, description, revised
		468	10.1.1, chapter title, changed. Table 10.1, table title, changed
		468	10.1.1, Table 10.1: Description of MSEBIM_ACD[31..0] and MSEBIS_ACD[31..0], modified
		469	10.1.2, chapter title, changed. Table 10.2, table title, changed
		469	10.1.2, Table 10.2: Description of MSEBI_CS[n]_N, modified
		469	10.1.2, Table 10.2: Description of MSEBI_CSREG_N, changed MSEBI_CSREG_N from 1 to 0 on "CS[n]_N Access Type" is" Access CS[n]_N shared registers", corrected
		480	10.3.1.1, Table 10.13: Function of bMSEBIM_CLEDATA and bMSEBIM_ALEDATA, revised
		483	10.3.1.3, Table 10.15: Function of bMSEBIM_SINGLE_DEST_WIDTH, revised
		484	10.3.1.3, Table 10.15: Function of bMSEBIM_TDMAE1, revised
		485	10.3.1.4, Table 10.16: Function of bMSEBIM_SINGLE_SRC_WIDTH, revised
		486	10.3.1.4, Table 10.16: Function of bMSEBIM_RDMAE1, revised
		487	10.3.1.5, Table 10.17: Function of bMSEBIM_ADDRDMA_READ_2, modified
		488	10.3.1,6, Table 10.18: Function of bMSEBIM_ADDRDMA_CURRENTREAD, modified
		489	10.3.1.7, Table 10.19: Function of bMSEBIM_ADDRDMA_WRITE_2, modified
		490	10.3.1.8, Table 10.20: Function of bMSEBIM_ADDRDMA_CURRENTWRITE, modified
		492	10.3.1.9, Table 10.21: Funciton of bMSEBIM_DMATDLR, DEST_BURST_SIZE → bMSEBIM_DEST_BURST_SIZE, corrected
		494	10.3.1.10, Table 10.22: Function of bMSEBIM_DMARDLR, SRC_BURST_SIZE → bMSEBIM_SRC_BURST_SIZE, corrected
		496	10.3.1.11, Table 10.23: Function of b14 to b12, revised
		504, 505	10.3.3.1, b0, bMSEBIS_ALEDATA → Reserved, modified
		514, 515	10.3.3.9, Table 10.36: Function of b14 to b12, revised
		517	10.3.3.9, Table 10.36: Function of bMSEBIS_BUSY, revised
		518	10.3.3.10, b30, Reserved → bMSEBIS_WAIT_CONF, modified
		532	10.4.2, Command phase → Control phase, CMD sub-phase → VALID sub-phase, modified
		542 to 547	10.4.3.1, description, revised
		548	10.4.3.3, chapter title, changed
		548 to 559	10.4.3.3, CMD sub phase → VALID sub phase, modified
		548 to 550	10.4.3.3, high Impedance → floating, modified
		561 to 576	10.4.4 (including sub-section), CMD sub phase → VALID sub phase, modified
		555	10.4.4.1, Figure 10.14, modified
		567 to 569	10.4.4.4, Figure 10.24 to 10.26, modified
		571	10.4.4.5, Figure 10.28, modified
		580	10.4.4.7, Figure 10.36, modified
		592	10.4.6.2, description, revised
		603 to 605	10.4.6.3, description, revised
		606	10.4.6.3 (1), description, revised

Rev.	Date	Description	
		Page	Summary
0.90	Dec 28, 2017	609	10.4.6.3 (4), 120/30 = 3 → 120/30 = 4, DMA block transfer is 3 → DMA block transfer is 4, corrected
		609 to 610	10.4.6.3 (4), Figure 10.56 to 10.57, modified (spell error)
		610	10.4.6.3 (4), description, modified
		620	10.4.7.2 (2) (a), bMSEBIS_DMATX_ENABLE in rMSEBIS_CONFIG_CS[n]_N → bMSEBIS_DMATX_ENABLE in rMSEBIS_DMATX_REQ_CS[n]_N, corrected
		622	10.4.7.2 (2) (b), bMSEBIS_DMARX_ENABLE in the rMSEBIS_CONFIG_CS[n]_N → bMSEBIS_DMARX_ENABLE in the rMSEBIS_DMARX_REQ_CS[n]_N, corrected
		624	10.4.7.2 (3), description, revised
		628	10.4.7.5, Read 0x1234_FEDk on rMSEBIS_ID_CS[n]_N → Read 0x1234_FEDn on rMSEBIS_ID_CS[n]_N, corrected
		631	10.5, description, modified
0.95	Oct 19, 2018	—	All sections, spelling, syntax errors and appearances are corrected, and expressions are modified properly
		—	All sections, unified clock notation
		4	How to Use This Manual, 1. Objective and Target Users, Documents related to RZ/N1 (R18DS0026 → R01DS0323), table modified
		6	How to Use This Manual, 3. List of Abbreviations and Acronyms, INTC, OTP, description modified
		22	1.2 Signal Interfaces, UART[m]_SCLK (External Internal → Serial), description modified
		23 to 30	1.3.1..8 Register Map UART 1..8, Table 1.1..8 Register Map UART 1..8, note added
		38	1.4.7 rUart_LCR — Line Control Register, bUart_StickParity, expression modified
		43	1.4.9 rUart_LSR — Line Status Register, bUart_PE, description modified
		50	1.4.14 rUart_TFR — Transmit FIFO Read, bUart_TFR, expression modified
		56	1.4.19 rUart_SRR — Software Reset Register, bUart_XFR, bUart_RFR, description modified
		75	1.5.1.2 Baud Rate Tolerance to 19200 baud, description deleted
		76	1.5.1.3 FIFO Management (byte → bits), description modified
		76	1.5.1.4 Clock Management, description modified
		78	1.5.1.6 Interrupts, Table 1.41 Interrupt Control Functions, bUart_IID (4'b0100 → 4'b1100), value modified
		79	1.5.1.7 Auto Flow Control (rUart_MCR bits → rUart_MCR register), expression modified
		86	1.5.1.9 DMA Management (Only UART4, 5, 6, 7, 8), (4) Selecting DEST_MSIZE and Transmit FIFO Overflow (underflow → overflow, others), description modified
		86	1.5.1.9 DMA Management (Only UART4, 5, 6, 7, 8), (7) Selecting SRC_MSIZE and Receive FIFO Underflow, description modified
		98	2.1 Overview, Slave selects number, (delete: Baud rate reference clock), description modified
		100	2.2 Signal Interfaces, description modified
		107	2.4.1 rSpi_CTRLR0 — Control Register 0 (by bSpi_SSIENR), description, added 2.4.2..8 Same as 2.4.1 rSpi_CTRLR0
		108	2.4.1 rSpi_CTRLR0 — Control Register 0, bSpi_TMOD, description modified
		109	2.4.2 rSpi_CTRLR1 — Control Register 1, caution modified
		112, 113	2.4.5 rSpi_SER — Slave Enable Register, bSpi_SoftwareSS, bSpi_HardwareSS, description modified
		114	2.4.6 rSpi_BAUDR — Baud Rate Select, bSpi_SCKDV, description modified
		117	2.4.9 rSpi_TXFLR — Transmit FIFO Level Register, bSpi_TXTFL (b3 to b0 → b4 to b0), description modified
		118	2.4.10 rSpi_RXFLR — Receive FIFO Level Register, bSpi_RXTFL (b3 to b0 → b4 to b0), description modified

Rev.	Date	Description	
		Page	Summary
0.95	Oct 19, 2018	131	2.4.22 rSpi_DR — Data Register, NOTE, description modified
		138	2.5.2 Typical Connection between SPI Master & Slave (The serial bit rate clock → The serial clock), term modified
		139	2.5.3 Control Slave Select Line by Hardware or Software Mode, caution modified
		139	2.5.3 Control Slave Select Line by Hardware or Software Mode, Figure 2.4 Control Slave Select Line by Hardware or Software Mode, description deleted
		140	2.5.4 Programmable Prescaler Clock, description modified
		141	2.5.4 Programmable Prescaler Clock, Figure 2.5 SPI Master Mode, Maximum Clock Ratio, figure deleted
		141	2.5.4 Programmable Prescaler Clock, Figure 2.6 SPI Slave Mode, Maximum clock Ratio, figure deleted
		141	2.5.5 Data Input Sample Delay (SPI_MOSI → SPI_MISO, others), description modified
		142	2.5.6 Transmit & Receive FIFO & Control, description modified
		144	2.5.8.3 Receive Only Mode (SPI_MOSI → SPI_MISO, others), description modified
		146 to 148	2.5.9 Motorola Serial Peripheral Interface, description modified
		154	2.5.11 National Semiconductor Microwire, Figure 2.19 National Semiconductor Mode, Single Transfer, Receive Data, SPI controller in Slave Mode, figure modified
		155	2.5.11 National Semiconductor Microwire, Figure 2.20 National Semiconductor Mode, Single Transfer, Transmit Data, SPI controller in Slave Mode, figure modified
		156	2.5.12 DMA Control, (handshaking → request), expression modified
		158	2.5.12.3 Choosing the Transmit Watermark Level, Figure 2.22 SPI Case2: Transmit Watermark Level (UART → SPI), figure modified
		159	2.5.12.4 Selecting DEST_MSIZE and Transmit FIFO Overflow, description modified
		160	2.5.12.6 Choosing the Receive Watermark Level, Figure 2.23 SPI Case3: Receive Watermark Level (UART → SPI), figure modified
		160	2.5.12.7 Selecting SRC_MSIZE and Receive FIFO Underflow, description modified
		161, 162	2.6.1.1 Programming Master SPI in Motorola & Texas Mode, description modified
		166	2.6.1.3 Programming Slave SPI in Motorola & Texas Mode, description modified
		174	3.4.1 IC_CON — I2C Control Register, SPEED ((100 kb/s) → (≤100 kb/s)), description modified
		176	3.4.3 IC_SAR — I2C Slave Address Register, IC_SAR (The default values → This value), description modified
		196	3.4.26 IC_STATUS — I2C Status Register, (Bits 3 and 10 → Bits 3 and 4), MST_ACTIVITY, description modified
		202	3.4.30 IC_TX_ABRT_SOURCE — I2C Transmit Abort Source Register, ABRT_GCALL_READ, description deleted
		204	3.4.32 IC_SDA_SETUP — I2C SDA Setup Register (tSU;DAT (note 4) → tSU;DAT), description modified
		208	3.4.35 IC_FS_SPKLEN — I2C Sm, Fm Spike Suppression Limit (tSP(table 4) → tSP), description modified
		211	3.5.1.1 Initial Configuration, (4) Enable the I2C controller by writing a “1” in bit 0 of the IC_ENABLE register., note deleted
		212	3.5.1.2 Slave Transmitter Operation for a Single Byte, (6) Software must clear the RD_REQ and TX_ABRT interrupts, description modified
		213	3.5.1.3 Slave Receiver Operation for a Single Byte, (4) I2C controller asserts the RX_FULL interrupt (IC_TX_TL → IC_RX_TL, described in → similarly to), description modified
		213	3.5.1.4 Slave Transfer Operation for Bulk Transfer (IC_INTR_STAT → IC_INTR_MASK, R_RD_REQ → RD_REQ, others), description modified
		215	3.5.2.1 Initial Configuration, (2), (3), description modified
		217	3.5.3 Disabling the I2C controller (hardware → I2C controller, others), description modified

Rev.	Date	Description	
		Page	Summary
0.95	Oct 19, 2018	219	3.6.1 Spike Suppression ((tSP, Table 4) → (tSP) , for low frequencies → by a long pulse), description modified
		221	3.6.2.1 Minimum High and Low Counts, Table 3.40 Minimum High and Low Counts (than equal I2C_PCLK. → than or equal to I2C_PCLK.), note modified
		224	4.1 Overview, (pin → signal), expression modified
		224	4.1 Overview, Figure 4.1 BGPIO Summary Synoptic (32b AHBS → 32b APBS), figure modified
		226	4.2 Signal Interfaces, Table 4.1 BGPIO Signal Interface (by GPIO multiplexing logic. → by IO Multiplexing logic.), note modified
		234	4.4.12 rGPIO_ext_porta — GPIO Port A Data Input Register, bGPIO_ext_port (data register → data output register), description modified
		235	4.4.13 rGPIO_ext_portb — GPIO Port B Data Input Register, bGPIO_ext_port (data register → data output register), description modified
		236	4.5.1.1 Data & Control Flow (the output → the external output, I/O pins → I/O, external pins → external inputs), description modified
		236	4.5.1.2 Interruption (Only Port A), description deleted
		237, 238	4.5.1.4 Trigger Synchronous Operation (CFG_GPIOT_PTEN_mj[n] → The bit in System Control CFG_GPIOT_PTEN_mj[n], others), description modified
		238	4.5.1.4 Trigger Synchronous Operation, Figure 4.5 Synchronization Principle and Capture on Event (INT_REQ[31:0] → INT_REQ[3:0], [k] → [n]), figure modified
		239	4.5.1.4 Trigger Synchronous Operation, Table 4.19 Trigger Synchronous Operation Allocation Line, (pin → signal), expression modified
		240	4.6.1 Programming Consideration (Programming → In order to prevent glitches, programming), caution modified
		244	5.4.1 rTimerLoadCount_[n] — Preset Value of Sub-timer[n] (n = 0..5), bTimerLoadCount, description added
		245	5.4.2 rTimerLoadCount_[n] — Preset Value of Sub-timer[n] (n = 6..7), bTimerLoadCount, description added
		256	5.4.14 rTimer_DMA_PendingClrOvf — TIMER DMA Overflow Clear, bTimer_DMA_RunningClrOvf_7 (Always Reserved → When read, this bit returns “0”, others), description modified
		256	5.4.14 rTimer_DMA_PendingClrOvf — TIMER DMA Overflow Clear, b7, b6 (R/W → W), modified
		256	5.4.14 rTimer_DMA_PendingClrOvf — TIMER DMA Overflow Clear (bTimer_DMA_RunnningClrOvf_6 → bTimer_DMA_RunningClrOvf_6), phrase modified
		257	5.5.2 Counter 16 or 32 Bits, Two events can clear the timer: (Timer is enabled after being reset or disabled → Enable timer after reset or disabled), description modified
		259	5.5.2 Counter 16 or 32 Bits, [Enable] (Restart the timer in increment mode. → Restart timer increment.), description modified
		260	5.5.3 Interruption (count up from → count up to), description modified
		260	5.5.3 Interruption, Figure 5.5 Timer Interruption (Timer Interrupt rise edge), figure modified
		261	5.5.4 DMA Control, description modified
		—	Overall, Section 6 CAN, (record → hold, recorded → held), phrase modified
		263	6.1 Overview, (with record of bit position → with data of bit position), expression modified
		273	6.4.3 rCan_SR — Controller Status Register, bCan_RBS (Full → Not Empty), description modified
		289	6.4.16 rCan_ACR[n] — Acceptance Code Filter [n] Register (n = 0..3), bCan_ACR (rCan_AMR0 → rCan_AMR[n]), description modified
		290	6.4.17 rCan_AMR[n] — Acceptance Mask Filter [n] Register (n = 0..3), bCan_AMR (rCan_ACR0 → rCan_ACR[n]), description modified
		320	6.5.10 Error Handling, Table 6.36 Increment/Decrement of Transmit and Receive Error Counte (error → error flag, Overload error → Overload flag), phrase modified

Rev.	Date	Description	
		Page	Summary
0.95	Oct 19, 2018	337	6.5.16 Difference between CAN Controllers and Reference Philips SJA1000 Devices, Sleep Mode, description modified
		339	7.1 Overview, description deleted
		358	7.4.1.13 rADC_FORCE — ADC Request, bADC_FORCE_VC, description modified
		365	7.4.1.18 rADC_ACQS — ADC Control Sample and Hold, Function (f → fADC_CLK, others), expression modified
		376	7.5 Operation, Figure 7.3 ADC Controller Synoptic (delete: iAdcTrig[7:0]), figure modified
		378	7.5.1 Virtual Channel ADC_VC Principle Operation, Figure 7.4 Virtual Channel ADC_VC Architecture (delete: iAdcTrig_Sync[7:0]), figure modified
		390	7.5.6 Simultaneous Sample and Hold, (ADC_VC1..4 → ADC_VC1..3), description modified
		402	7.5.10.1 Overview on DMA Operation (The source and destination transfer → The source transfer), caution modified
		407, 409	8.4.1 rLcd_CR1 — Control Register 1, bLcd_PSS, bLcd_LCE, description modified
		410	8.4.2 rLcd_HTR — Horizontal Timing Register (bLcd_HSW → bLcd_PPL, bLcd_HBP → bLcd_HFP), modified
		423	8.4.13 rLcd_PWMFR_0 — PWM0 Frequency Register, bLcd_PWMFCD_0, description modified
		424	8.4.14 rLcd_PWMDCR_0 — PWM0 Duty Cycle Register, bLcd_PWMDC_0, description modified
		427	8.4.17 rLcd_PWMFR_1 — PWM1 Frequency Register, bLcd_PWMFCD_1, description modified
		428	8.4.18 rLcd_PWMDCR_1 — PWM1 Duty Cycle Register, bLcd_PWMDC_1, description modified
		435	8.5.1 Main Features Description, description modified
		436	8.5.2 Bandwidth Limitation, description modified
		439	8.5.4 DMA Controller and Memory Interface, description deleted
		439	8.5.5 Frame Buffer Organization, description deleted
		439	8.5.5 Frame Buffer Organization (Between Table 8.29 LCD Frame Buffer Support for Palette Load to Table 8.34 LCD Frame Buffer Organization, bLcd_PSS = 1 and bLcd_BPP = 3'b011), table deleted
		441	8.5.7 Pixel Unpack, Figure 8.5 Pixel Unpack, Big Endian Frame Buffer Byte placed in Big Endian Pixel Byte (Lcd_EPO=1 → Lcd_EPO=x), figure modified
		442, 443	8.5.8 Palette Lookup Table, description modified
		445	8.5.9 Output FIFO and Formatter, Figure 8.8 LCD Output Formatting, bLcd_BPP: 16, 18, 24 (RGB defined in Palette → Frame Buffer Direct), figure modified
		450	8.5.15 Blink Function, Figure 8.10 Blink BL[1:0] Attribute Management (Slow speed blink: Clock → Clock/4, Fast speed blink: Clock/4 → Clock), figure modified
		451	8.5.16 Limitation, Concerning loading Palette by DMA:, description deleted
		452	9.1 Overview (Up to 64 → 64, othrers), description modified
		454	9.4.1 rSemaphoreLockCPU[m]_[n] — Semaphore Lock CPU[m] Register [n], description added
		455	9.4.2 rSemaphoreStatusCPU[m]_[n] — Semaphore Status CPU[m] Register [n] (CPU m = 1..4 → With CPU[m] (m = 1..4)), expression modified
		458	9.6 Usage Notes, caution modified
		462	10.1.2 MSEBI Master Address Mapping of CS[n] from CPU (Table 10.2 Address Mapping of CS[n] from CPU), subsection added
		463	10.1.3 Multiplexed Signal Interface, Table 10.3 Multiplexed Signal Interface (1/2), MSEBI_CS[n]_N, description modified
		471	10.2.2 Register Map MSEBI Master from DMA, Table 10.11 Register Map MSEBI Master from DMA (64KB → 32KB), description modified

Rev.	Date	Description	
		Page	Summary
0.95	Oct 19, 2018	489	10.3.1.11 rMSEBIM_CONFIG_CS[n]_N — Chip Select Config Register (n = 0..3) (b15: "–" → bMSEBIM_MULTI_DLE, b12:bMSEBIM_CS[n]N_ROUTING_CS1_N → bMSEBIM_CS0N_ROUTING_CS1_N), description modified
		494	10.3.1.12 rMSEBIM_CONFIG — Common Config Register, bMSEBIM_CLKENABLE, description modified
		524	10.3.4.3 rMSEBIS_ID_CS[n]_N — Slave ID Register (n = 0..3), bMSEBIS_ID (This register is pooled → This register is read, 0x1234_FED0 → 0x1234_FEDn (n = 0..3)), description modified
		525	10.4.1.1 AHB Slave Interface, description modified
		525	10.4.1.2 AHB Master Interface (MSEBI Slave only), title modified
		526	10.4.2 Use Case Device Connection, MSEBI(x)_CLK, description modified
		531	10.4.2.5 Three Devices, Mode8/16/32, Asynchronous, Figure 10.6 Three Devices, Mode8/16/32, Asynchronous, figure modified
		532	10.4.2.6 Three Devices, Mode8/16/32, Mixed Synchronous and Asynchronous, Figure 10.7 Three Devices, Mode8/16/32, Mixed Synchronous and Asynchronous, figure modified
		533	10.4.2.7 One Device, Mode8, Asynchronous, ALE in Parallel Mode, Figure 10.8 One Device, Mode8, Asynchronous, ALE in Parallel Mode (READY_N → READY), figure modified
		533	10.4.2.7 Two Devices, Mode8/16, Synchronous, Multi Master (Figure 10.8 Two Devices, Mode8/16, Synchronous, Multi-Master), subsection deleted
		533	10.4.2.8 Three Devices, Mode8, Synchronous/Asynchronous, Multi Master (Figure 10.9 Three Devices, Mode8, Synchronous/Asynchronous, Multi-Master), subsection deleted
		544 to 574 593 to 594	10.4.4.1 Asynchronous Mode, One ALE, Figure 10.9 MSEBI Timing, Asynchronous Mode, Write1, NoWait, NoBurst, One ALE (Delete MSEBI(x)_HCLK, Read figure: Changed setup data and hold data reference to MSEBI(x)_CLK) figure modified 10.4.4.1 to 10.4.4.7, 10.4.6.2 (Figure 10.10..36, Figure 10.51..52) Same as 10.4.4.1 Asynchronous Mode, One ALE, Figure 10.9 MSEBI Timing, Asynchronous Mode, Write1, NoWait, NoBurst, One ALE
		576 to 579	10.4.5.2 MSEBI Interrupt: End of Block Detection by the Master, For a transfer using DMA TX[n] FIFO (n = 0..1) (by pooling (on a deported task) → by polling), description modified
		580	10.4.5.3 MSEBI Interrupt: End of Block Detection by the Slave (by pooling (on a deported task) → by polling), description modified
		585	10.4.6.2 MSEBI Master: Burst Mode, Figure 10.45 MSEBI: Round Robin Priority, figure modified
		601	10.4.6.3 MSEBI Master: DMA Control, (4), Figure 10.54 MSEBI Case1: Transmit Watermark Level (Burst Size: 60 → 30), figure modified
		603	10.4.6.3 MSEBI Master: DMA Control, (5) Selecting DEST_MSIZE and Transmit FIFO Overflow, description modified
		604	10.4.6.3 MSEBI Master: DMA Control, (8) Selecting SRC_MSIZE and Receive FIFO Underflow, description modified
		615	10.4.7.2 MSEBI Slave: Burst Mode, (2), Figure 10.62 Slave DMA FIFOs for Requests from Master MSEBI DMA RX FIFOs (Add dotted line for "These data are not used"), figure modified
		617	10.4.7.2 MSEBI Slave: Burst Mode, (3), Figure 10.63 MSEBI Slave: External DMA Request (by register → by following bits), figure modified
1.00	Mar 29, 2019	—	All sections, spelling, syntax errors and appearances are corrected, and expressions are modified properly
		31	1.4.1 rUart_DLL — Divisor Latch (Low), bUart_DLL, description modified
		32	1.4.2 rUart_DLH — Divisor Latch (High), bUart_DLH, description modified
		37 to 38	1.4.6 rUart_FCR — FIFO Control Register, description modified
		39 to 40	1.4.7 rUart_LCR — Line Control Register, description modified
		41	1.4.8 rUart_MCR — Modem Control Register, bUart_LB, description modified
		54	1.4.16 rUart_USR — UART Status Register, bUart_BUSY, description modified

Rev.	Date	Description	
		Page	Summary
1.00	Mar 29, 2019	57	1.4.19 rUart_SRR — Software Reset Register, description modified
		60	1.4.22 rUart_SFE — Shadow FIFO Enable, bUart_SFE, description modified
		64	1.4.26 rUart_DMASA — DMA Software Acknowledge, bUart_DMASA, description modified
		65	1.4.27 rUart_TO — Time-Out Counter Configuration Register, bUart_TO3, description modified
		67	1.4.28 rUart_CTRLTO — Time-Out Control Register, bUart_TG, description modified
		69	1.4.29 rUart_STATUSTO — Time-Out Counter Status Register, bUart_TIMEOUTStatus3, description modified
		75 to 76	1.5.1.1 UART (RS232) Serial Protocol, description modified
		76	1.5.1.1 UART (RS232) Serial Protocol, Figure 1.4 Receiver Serial Data Sample, figure modified
		76	1.5.1.2 Baud Rate Tolerance to 19200 baud, description added
		78	1.5.1.5 Back to Back Character Stream Transmission, description modified
		82	1.5.1.8 Programmable THRE interrupt, Figure 1.7 Flowchart of Interrupt Generation, Programmable THRE Interrupt Mode & FIFO Enable, figure modified
		91 to 95	1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management, description modified
		94	1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management, Figure 1.15 Receiver & Transceiver Time-Out0..3, Timing Description, figure modified
		96	1.5.1.11 Half-Duplex Mode Management, description modified
		96	1.5.1.11 Half-Duplex Mode Management, Figure 1.17 Transceiver Time-Guard Synoptic, figure modified
		97	1.5.1.11 Half-Duplex Mode Management, Figure 1.18 Data Enable Auto Generation in Half-Duplex Mode, figure modified
		100	2.1 Overview, Figure 2.2 SPI Slave Synoptic, figure modified
		109	2.4.1 rSpi_CTRLR0 — Control Register 0, bSpi_SCPH, description modified
		111	2.4.3 rSpi_SSIENR — Enable Register, bSpi_SSIENR, description modified
		113	2.4.5 rSpi_SER — Slave Enable Register, description modified
		116	2.4.7 rSpi_TXFTLR — Transmit FIFO Threshold Level, description modified
		117	2.4.8 rSpi_RXFTLR — Receive FIFO Threshold Level, description modified
		132	2.4.22 rSpi_DR — Data Register, bSpi_DR, description deleted
		154	2.5.11 National Semiconductor Microwire, description deleted
		166	2.6.1.2 Programming Master SPI in National Semiconductor Mode, Figure 2.25 SPI Controller in Master Mode, National Semiconductor Mode, figure modified
		168	2.6.1.3 Programming Slave SPI in Motorola & Texas Mode, Figure 2.26 SPI Controller in Slave Mode, Motorola & Texas Mode, figure modified
		174	3.4.1 IC_CON — I2C Control Register, RX_FIFO_FULL_HLD_CTRL, description modified
		183	3.4.9 IC_INTR_STAT — I2C Interrupt Status Register, ALL_BITS, description modified
		184 to 185	3.4.10 IC_INTR_MASK — I2C Interrupt Mask Register, ALL_BITS, description modified
		186 to 188	3.4.11 IC_RAW_INTR_STAT — I2C Raw Interrupt Status Register, description modified
		198	3.4.26 IC_STATUS — I2C Status Register, IC_STATUS_ACTIVITY, description modified
		202	3.4.30 IC_TX_ABRT_SOURCE — I2C Transmit Abort Source Register, ABRT_USER_ABRT, description modified
		234	4.4.9 rGPIO_intstatus — GPIO Port A Interrupt Status, bGPIO_intstatus, description modified
		234	4.4.10 rGPIO_raw_intstatus — GPIO Port A Raw Interrupt Status (Premasking), bGPIO_raw_intstatus, description modified
		247	5.4.3 rTimerCurrentCount_[n] — Current Value of Sub-timer[n] (n = 0..5), bTimerCurrentCount, description modified
		247	5.4.4 rTimerCurrentCount_[n] — Current Value of Sub-timer[n] (n = 6..7), bTimerCurrentCount, description modified

Rev.	Date	Description	
		Page	Summary
1.00	Mar 29, 2019	339	6.6 Special Notice, description deleted
		340	7.1 Overview, DMA coupling, description modified
		404	7.6 Usage Notes, 7.6.1 Restriction, subsection added
		440	8.5.3 Timing and Control, Figure 8.2 LCD Horizontal Timing, figure modified
		464	10.1.1 Signal Interfaces, Table 10.1 Signal Interface, note added
		483	10.3.1.5 rMSEBIM_ADDRDMA_READ_CS[n]_N — DMA Read Address Register (n = 0..1), bMSEBIM_ADDRDMA_READ_2, description deleted
		527	10.4.1.1 AHB Slave Interface, For MSEBI Master, description added
1.10	Sep 30, 2020	474	10.2.4 Register Map MSEBI Slave from MSEBI, Table 10.13 Register Map MSEBI Slave from MSEBI, description added
		522	10.3.4.1 rMSEBIS_INT — Slave Interrupt Register, Address, description added
		524	10.3.4.2 rMSEBIS_STATUS — Slave Status Register, Address, description added
		526	10.3.4.3 rMSEBIS_ID_CS[n]_N — Slave ID Register (n = 0..3), Address, description added
		621	10.4.7.4 MSEBI Slave: Register Access by Master, description modified
1.20	Dec 29, 2021	—	All sections, spelling, syntax errors and appearances are corrected, and expressions are modified properly
		22	1.2 Signal Interfaces, UART[m]_RTS_N, description modified
		67, 68	1.4.28 rUart_CTRLTO — Time-Out Control Register, description modified
		69	1.4.29 rUart_STATUSTO — Time-Out Counter Status Register, bUart_DE, description modified
		90	1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management, description added
		93	1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management, (1) Receiver Time-Out, Figure 1.14 Receiver Time-Out Synoptic, figure modified
		93	1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management, (1) Receiver Time-Out, description modified
		95	1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management, (3) Time-out counter timing, Figure 1.15 Receiver & Transceiver Time-Out0..3, Timing Description, figure moved
		95	1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management, (2) Transceiver Time-Out, Figure 1.16 Transceiver Time-Out Synoptic, figure modified
		94, 95	1.5.1.10 Transceiver & Receiver Time-Out for MODBUS Management, (2) Transceiver Time-Out, description modified
		—	1.5.1.11 Half-Duplex Mode Management, subsection deleted
		96	1.5.2 Usage Notes, subsection added
		160	2.6.1 Programming Consideration, CAUTION, description modified

RZ/N1D Group, RZ/N1S Group, RZ/N1L Group
User's Manual: Peripherals

Publication Date: Rev.0.50 Jun 30, 2017
Rev.1.20 Dec 29, 2021

Published by: Renesas Electronics Corporation

RZ/N1D Group, RZ/N1S Group, RZ/N1L Group

Renesas Electronics Corporation

R01UH0752EJ0120