BA22

32-bit RISC Processor

External Peripherals Specification

January 2012

IP Product Version
4.0p1

Document Signature
BA22–EPS–4.0p1–101

CAST, Inc.

Document Version

This document with its associated Release Notes applies to the version(s) of the core specified on the cover. See the Release Notes for any updates and additional information not included here.

Document Version History

Version	Date	Person	Changes from Previous Version
1.00	28, July 2011	N.S	First draft ; Released for use with early customers
1.01	17, Jan 2012	N.S.	SPI updates from v 2.3

CAST, Inc.
11 Stonewall Court, Woodcliff Lake, NJ 07677
Phone: +1 201-391-8300 Fax: +1 201-391-8694
info@cast-inc.com www.cast-inc.com

Table of Contents

1. Overview 9
1.1 BA22 – Peripherals Interface 9

2. General Purpose Input Output 10
2.1 Overview 10
2.2 Block Diagram 10
2.3 Signal Descriptions 10
2.4 Register Descriptions 11
2.5 Functional Description 13
2.6 I/O Control 13
2.7 Timing 13

3. Universal Asynchronous Receiver/Transmitter 16
3.1 Overview 16
3.2 Block Diagram 16
3.3 Signal Descriptions 17
3.4 Register Descriptions 17
3.5 Functional Description 20
3.5.1 Transmitter Operation 20
3.5.2 Receiver Operation 20
3.5.3 Baud Rate Generator 20
3.5.3.1. Optional Divisor for Baud Rate Generator 21
3.5.4 Interrupts 21
3.5.4.1. Enabling Interrupts 21
3.5.4.2. Interrupt Types 21
3.5.4.3. Interrupt Identification 22
3.5.5 FIFO Control 22
3.5.6 Receiver FIFO 22
3.5.7 Transmitter FIFO 23
3.5.8 Asynchronous Serial Data Format 23
3.5.9 Diagnostic Mechanisms 23
3.5.10 Line Status 24
3.5.11 Modem Control / Status 25
3.5.12 Modem Control 25
3.5.13 Modem Status 25
3.5.14 DMA modes 25
3.5.15 Mode 0 (16450 and 16550) 25
3.5.16 Mode 1 (16550 only) 25
3.5.17 Loopback Operation 25
3.5.18 General Purpose Outputs 26
3.6 Timing Diagrams 26
3.6.1 APB Interface Timing 26
3.6.2 Example Transmitter Timing 28
3.6.3 Example Receiver Timing 28
3.6.4 PCLK & clkBaud 29
3.6.5 sout & clkBaud 29
3.7 Interrupt Generation 29
3.8 Minimal Implementation 29

4. Timer (Event Counter) 31
4.1 Overview 31
4.2 Block Diagram 31
4.3 Signal Descriptions 31
4.4 Register Descriptions 32
4.5 Functional Description 32
4.6 Timing 33

5. Processor WatchDog Timer 36

5.1 Overview ... 36
5.2 Block Diagram ... 36
5.3 Signal Descriptions ... 36
5.4 Programming Interface ... 37
5.4.1 Register Descriptions ... 37
5.5 Functional Description ... 39
5.6 Timing ... 40
5.6.1 Interface and Interrupt Timing ... 40
5.6.2 Reset Timing ... 42

6. Real Time Clock (RTC) ... 43
6.1 Overview ... 43
6.2 Block Diagram ... 43
6.3 Register Descriptions ... 43

7. I2C Master ... 50
7.1 Overview ... 50
7.2 Block Diagram ... 50
7.3 Signal Descriptions ... 51
7.4 Programming Interface ... 51
7.4.1 Register Descriptions ... 51
7.5 Functional Description ... 52
7.5.1 Clock Divider ... 52
7.6 Command Interface ... 53
7.6.1 Command Interface Examples ... 54
7.6.2 I2C Write Example ... 54
7.6.3 I2C Read Example ... 54
7.7 Interrupts ... 55
7.7.1 Enabling Interrupts ... 55
7.7.2 Clearing the Interrupt ... 56
7.8 FIFOs ... 56
7.9 Timing Diagrams ... 57

8. Serial Peripheral Interface (SPI) Controller ... 58
8.1 Overview ... 58
8.2 Block Diagram ... 59
8.3 Signal Descriptions ... 60
8.4 Build Options ... 60
8.5 Register Description ... 60
8.6 APB Interface Timing ... 63
8.7 SPI Master Data Transfer ... 64
8.8 SPI Slave Data Transfer ... 66
8.9 Programming ... 67
8.9.1 Reset ... 67
8.9.2 Setup Sequence ... 67
8.9.3 Transmit Sequence Non-FIFO mode ... 67
8.9.4 Transmit Sequence FIFO mode ... 68
8.9.5 Receive Sequence Non-FIFO mode ... 68
8.9.6 Receive Sequence FIFO mode ... 68
8.9.7 Transmit or Receive Error ... 68

9. External Bus Interface ... 69
9.1 Overview ... 69
9.2 Block Diagram ... 69
9.3 Signal Descriptions ... 69
9.4 Programming Interface ... 70
9.4.1 Register Descriptions ... 70
9.5 Functional Description ... 71
9.5.1 APB Register Interface ... 71

9.5.2 Wait State Generation ... 71
9.5.3 Address Generation ... 71
9.5.4 Control Signal Generation ... 71
9.5.5 Data Steering ... 72
9.6 Timing Diagrams ... 72
9.6.1 Configuration Register Timing ... 72
9.6.2 AHB and EBI Timing ... 73

10. DMA-AHB Direct Memory Access Controller ... 78
10.1 Overview ... 78
10.2 Block Diagram ... 78
10.3 Functional Description ... 78
10.3.1 General ... 78
10.3.2 AMBA (AHB Master/AHB Slave) Interfaces ... 79
10.3.3 DMA Finite State Machine (FSM) and FIFO Control ... 79
10.3.4 Memory to Memory Transfer ... 80
10.3.5 Memory to Peripheral Transfer ... 80
10.3.6 Peripheral to Memory Transfer ... 80
10.3.7 Peripheral to Peripheral Transfer ... 80
10.3.8 Control and Status Registers ... 80
10.3.9 Interrupt Controller ... 81
10.3.10 Peripheral Handshaking ... 81
10.3.11 Peripheral HRESP=ERROR response ... 82
10.3.12 Peripheral HRESP=SPLIT and HRESP=RETRY responses ... 82
10.3.13 Linked List Support ... 82
10.4 Register Descriptions ... 83
10.5 Signal Descriptions ... 86
10.6 Timing Diagrams ... 87
10.6.1 AHB Master Interface Timing ... 87
10.6.2 Peripheral Interface Timing ... 88

11. SSRAM Controller ... 89
11.1 Overview ... 89
11.2 Block Diagram ... 89
11.3 Signal Descriptions ... 89
11.4 Functional Description ... 90
11.5 Timing ... 91

12. AHB to APB Bridge ... 94
12.1 Overview ... 94
12.2 Block Diagram ... 94
12.3 Signal Descriptions ... 94
12.4 Functional Description ... 95
12.5 Timing ... 95

13. AHB Arbiter ... 97
13.1 Overview ... 97
13.2 Block Diagram ... 97
13.3 Signal Descriptions ... 97
13.4 Functional Description ... 99
13.4.1 Determining the Next AHB Master ... 99
13.4.2 Keeping Track of Address Phase and Data Phase ... 100
13.4.3 Distribution (Fanout) ... 100
13.5 Timing ... 100

List of Figures

Figure 1-1: BA22 Peripherals ... 9

Figure 2-1: GPIO Block Diagram........ 10
Figure 2-2: APB Interface Timing 13
Figure 2-3: Input and Interrupt Timing 14
Figure 4-1: Interface and Interrupt Timing 33
Figure 5-1: WatchDog Timer Block Diagram 36
Figure 5-2: Interface and Interrupt Timing 40
Figure 5-3: Reset Timing 42
Figure 6-1: RTC Block Diagram 43
Figure 7-1: I2C Block Diagram 50
Figure 7-2: Example I2C Bus Events 57
Figure 7-3: Example I2C Clock Relationships........ 57
Figure 8-1: SPI Master - Slave Data Transmission 58
Figure 8-2: SPI Master - Slave interconnect 58
Figure 8-3: SPI Master to 3 Slaves interconnect........ 58
Figure 8-4: SPI Block Diagram 59
Figure 8-5: Typical IPC-SPI-APB I/O Connections 59
Figure 8-6: APB Interface Timing 63
Figure 8-7: SPI Master Data Transfer for LET = 0, SD = 0. 64
Figure 8-8: SPI Master Data Transfer for LET = 1, SD = 0. 65
Figure 8-9: SPI Master Data Transfer for LET = 0, SD = 1. 65
Figure 8-10: SPI Master Data Transfer for LET = 1, SD = 1. 66
Figure 8-11: SPI Slave Clock Relationships......... 67
Figure 9-1: EBI Bock Diagram........ 69
Figure 9-2: APB Interface Timing 72
Figure 9-3: AHB, Memory Interface Timing 74
Figure 10-1: IPC-DMA-AHB Block Diagram........ 78
Figure 10-2: AHB Master Interface Timing 88
Figure 10-3: Peripheral Interface Timing 88
Figure 11-1: AHB SRAM Controller Block Diagram 89
Figure 11-2: AHB SRAM Controller Timing 91
Figure 12-1 AHB to APB Bridge Block Digaram........ 94
Figure 12-2: AHB to APB Bridge Interface Timing........ 95
Figure 13-1: AHB Arbiter Block Diagram 97
Figure 13-2: AHB Bus Changeover from Master 0 to Master 1 100

List of Tables

Table 1-1: List of External Peripherals 9
Table 2-1: GPIO Interface Signals........ 10
Table 2-2: GPIO Registers Summary........11
Table 2-3: State Register (offset 0x00 11
Table 2-4: Set Register (offset 0x00)........11
Table 2-5: Clear Register (offset 0x04)........11
Table 2-6: Interrupts Register (offset 0x04 11
Table 2-7: Out Register (offset 0x08) 11
Table 2-8: In Register (offset 0x0C)........ 12
Table 2-9: Enable Set Register (offset 0x10) 12
Table 2-10: Enable Clear Register (offset 0x14) 12

Table 2-11: Edge Register (offset 0x18) 12
Table 2-12: Level Register (offset 0x1C) 12
Table 2-13: Set Level Register (offset 0x20) 12
Table 2-14: Clear Level Register (offset 0x24) 12
Table 2-15: Any Edge Set Register (offset 0x28) 12
Table 2-16: Any Edge Clear Register (offset 0x2C) 12
Table 2-17: Interrupt Clear Register (offset 0x30) 12
Table 2-18: Control Register (offset 0x34) 12
Table 3-1: UART Interface Signals 17
Table 3-2: UART Registers Summary 17
Table 3-3: Receiver Buffer Register (offset 0x00) 18
Table 3-4: Transmitter Holding Register (offset 0x00) 18
Table 3-5: Interrupt Enable Register (offset 0x04) 18
Table 3-6: Interrupt Identification Register (offset 0x08) 18
Table 3-7: FIFO Control Register (offset 0x08) 18
Table 3-8: Line Control Register (offset 0x0C) 19
Table 3-9: Modem Control Register (offset 0x10) 19
Table 3-10: Line Status Register (offset 0x14) 19
Table 3-11: Modem Status Register (offset 0x18) 19
Table 3-12: Scratch Register (offset 0x1C) 19
Table 3-13: Divisor Latch (LS) (offset 0x00) 20
Table 3-14: Divisor Latch (MS) (offset 0x04) 20
Table 3-15: UART Interrupts 21
Table 3-16: UART Interrupt Types 22
Table 3-17: UART FIFO Trigger Levels 22
Table 3-18: UART Character Length 23
Table 3-19: UART Stop Bits 23
Table 3-20: UART Parity Type 23
Table 3-21: UART Parity Functionality 24
Table 3-22: UART Loopback Operation 26
Table 3-23: APB Interface Timing 27
Table 3-24: UART Example Transmitter Timing 28
Table 3-25: UART Example Receiver Timing 29
Table 3-26: UART PCLK to clkBaud Relationship 29
Table 3-27: UART sout to clkBaud Relationship 29
Table 4-1: Timer Block Diagram 31
Table 4-2: Timer Interface Singals 31
Table 4-3: Timer Registers Summary 32
Table 4-4: Load (offset 0x00) 32
Table 4-5: Value (offset 0x04) 32
Table 4-6: Control (offset 0x08) 32
Table 4-7: Clear (offset 0x0C) 32
Table 5-1: WatchDog Timer Interface Singals 36
Table 5-2: WatchDog Timer Configurable Parameters 37
Table 5-3: Register Summary 37
Table 5-4: Load (offset: 0x0) 37
Table 5-5: Value (offset: 0x4) 37
Table 5-6: Control (offset: 0x8) 38

Table 5-7: Kick (offset: 0xC) 38
Table 5-8: Reset Occurred (offset: 0x10) 38
Table 5-9: Clear (offset 0x14) 39
Table 5-10: Interrupt Value (0x18) 39
Table 6-1: RTC Registers 44
Table 7-1: I2C Interface Signals 51
Table 7-2: I2C Register Summary 51
Table 7-3: Status Register (0ffset 0x00) 51
Table 7-4: Read Data Register (offset 0x04) 52
Table 7-5: Command Register (offset 0x08) 52
Table 7-6: Interrupt Enable Register (offset 0x0C) 52
Table 7-7: Control Register (offset 0x10) 52
Table 7-8: Prescale Register (offset 0x14) 52
Table 7-9: I2C Commands 53
Table 7-10: I2C Interrupts 55
Table 7-11: I2C Clock Domains 57
Table 8-1: Offset 0x00: Serial Data Transmit (to Tx FIFO) 60
Table 8-2: Offset 0x04: Serial Data Receive (from Rx FIFO) 60
Table 8-3: Offset 0x08: Serial Clock Divisor 60
Table 8-4: Offset 0x0C: Control Register 61
Table 8-5: Offset 0x10: Status Register (USE_SPI_FIFOS is defined) 61
Table 8-6: Offset 0x10: Status Register (USE_SPI_FIFOS is not defined) 61
Table 8-7: Offset 0x14: Slave Select Register 62
Table 8-8: Offset 0x18: Slave Select Polarity Register 62
Table 8-9: Offset 0x1C: Interrupt Enable Register 62
Table 8-10: Offset 0x20: Interrupt Status Register 62
Table 8-11: Offset 0x24: Interrupt Clear Register 62
Table 8-12: Interrupt Bit Mapping 62
Table 8-13: Offset 0x28: TX FIFO Watermark Level Register (only valid when USE_SPI_FIFOS is defined) 63
Table 8-14: Offset 0x2C: RX FIFO Watermark Level Register (only valid when USE_SPI_FIFOS is defined) 63
Table 8-15: Offset 0x30: FIFO Level Register 63
Table 8-16: Offset 0x34: FIFO Level Register 63
Table 9-1: EBI Interface Signals 69
Table 9-2: Register Summary 70
Table 9-3: Configuration Register (offset 0x00) 70
Table 10-1: Source and Destination with Same Transfer Width 79
Table 10-2: Source and Destination Different Same Transfer Widths 79
Table 10-3: Linked List Elements 83
Table 10-4: : IPC-DMA-AHB Register Descriptions 83
Table 10-5: IPC-DMA-AHB Interface Signals 86
Table 11-1: AHB SRAM Controller Interface Signals 89
Table 11-2: MWRITE Singal 90
Table 12-1: AHB to APB Bridge Interface Signals 94
Table 13-1: AHB Arbiter Interface SIgnals 97

1. Overview

The purpose of this document is to give detailed information about the BA22 external peripherals that are available. These peripherals may be easily integrated into the user's unique design via the AMBA interface.

Table 1-1: List of External Peripherals

Peripheral Name	Description	Bus Interface
GPIO	General Purpose Input and Output	APB
UART	Universal Asynchronous Receiver/Transmitter	APB
TIMER	Event counter	APB
WDOG	Watchdog Timer	APB
RTC	Real Time Clock	APB
I2C	I2C bus master	APB
SPI	Serial Peripheral Interface Controller	APB
EBI	External Bus Interface	AHB
DMA	Direct Memory Access Controller	AHB
SRAM-CTRL	Static RAM controller	AHB
AHB2APB	AHB to APB bus bridge	AHB/APB
AHB Arbiter	AHB Bus Arbiter	AHB

1.1 BA22 – Peripherals Interface

The external peripherals are connected to the BA22 via the AHB bus. An AHB system will in some cases require an AHB Arbiter to determine the priority of the bus masters (e.g. BA22 and DMA) controlling the bus. Peripherals with low bandwidth requirements use an APB interface. In order to use these peripherals, an AHB to APB bridge must be used. The following figure illustrates an example of the BA22 processor integrated with some of the external peripherals.

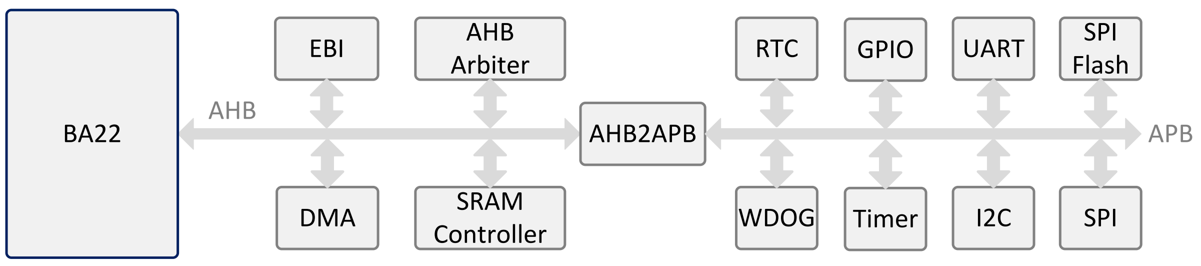


Figure 1-1: BA22 Peripherals

The external peripherals of your choice are integrated with the BA22 from CAST, and delivered to you in a single package with a testbench and a quick start application.

2. General Purpose Input Output

2.1 Overview

The GPIO is a configurable module allowing the use of up to 32 scalable I/O lines. If more than 32 I/Os are required, more than one GPIO module may be instantiated.

Each line can be configured independently resulting in a very useful I/O application. The GPIO module supports a wide variety of options concerning synchronization logic and an interrupt signal, which can be triggered by either a low level, high level, positive or negative edge. The GPIO is also fully compatible with the industry-standard APB bus interface.

The GPIO module is a standard APB Slave peripheral, and has a standard APB Slave Port. The APB Slave Port is typically connected to one of the various APB Mirrored-Slave Ports of an APB Channel module. The GPIO registers are accessed through its APB interface.

2.2 Block Diagram

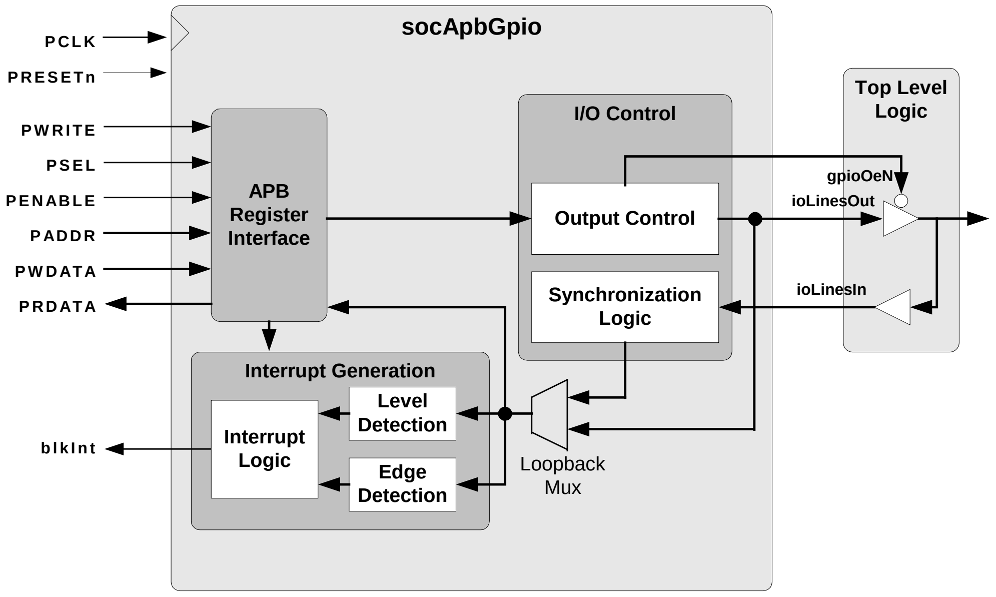


Figure 2-1: GPIO Block Diagram

2.3 Signal Descriptions

In the following table, NUM_LINES is a parameter that can be set to modify the width of the GPIO interface. NUM_LINES is valid in the range [1-32].

Table 2-1: GPIO Interface Signals

Signal Name	I/O	Description
APB Slave Interface		
PRESETn	I	Active low APB reset
PCLK	I	APB clock
PWRITE	I	APB Write (0=read 1=write)
PWDATA[31:0]	I	Write Data from the APB (32 bits)
PADDR[3:0]	I	Lower bits of APB Address bus (4 bits)
PSEL	I	APB Block select
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	Read data to the APB (32 bits)
GPIO Interface		

Signal Name	I/O	Description
blockInt	O	Block Interrupt
ioLinesOut[NUM_LINES-1:0]	O	Lines used if IO port is outbound (32 bits)
ioLinesIn[NUM_LINES-1:0]	I	Lines used if IO port is inbound (32 bits)
gpioOeN[NUM_LINES-1:0]	O	Direction of all I/O lines (1 = input, 0 = output) (32 bits)

2.4 Register Descriptions

Table 2-2: GPIO Registers Summary

Register Name	Offset	Description
State	0x00	Read to see current state of I/O signals
Set	0x00	Write 1’s to set output signals
Clear	0x04	Write 1’s to clear output signals
Interrupts	0x04	Read to see current state of interrupts
Out	0x08	write 1's to set direction to output
In	0x0c	write 1's to set direction to input
Enable Set	0x10	write 1's to set enable interrupt
Enable Clear	0x14	write 1's to set disable interrupt
Edge	0x18	write 1's to set interrupt to edge-sensitive
Level	0x1c	write 1's to set interrupt to level-sensitive
Set Level	0x20	write 1's to set interrupts to active high or rising edge
Clear Level	0x24	write 1's to set interrupts to active low or falling edge
Any Edge Set	0x28	write 1's to override interrupt edge selection & interrupt on any edge
Any Edge Clear	0x2c	write 1's to clear edge selection override
Interrupt Clear	0x30	write 1's to clear edge-sensitive interrupts
Control	0x34	Controls loopback/normal mode selection

Table 2-3: State Register (offset 0x00

Bits	Access	Default	Description
[31:0]	Read	0x00000000	Read to see current state of I/O signals. For each I/O, the synchronized input signal is read, regardless of whether I/O is configured as input or output.

Table 2-4: Set Register (offset 0x00)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 sets the corresponding output signal.

Table 2-5: Clear Register (offset 0x04)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 clears the corresponding output signal.

Table 2-6: Interrupts Register (offset 0x04

Bits	Access	Default	Description
[31:0]	Read	0x00000000	This register is read to see the current state of interrupts.

Table 2-7: Out Register (offset 0x08)

Bits	Access	Default	Description
[31:0]	R/W	0xFFFFFFFF	Each bit that is set to 1 configuresz the corresponding signal as an output. Read for current state of all active low output enable signals. All signals default to inputs.

Table 2-8: In Register (offset 0x0C)

Bits	Access	Default	Description
[31:0]	R/W	0xFFFFFFFF	Each bit that is set to 1 configures the corresponding signal as an input. Read for current state of all active low output enable signals. All signals default to inputs.

Table 2-9: Enable Set Register (offset 0x10)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 enables an interrupt for the corresponding signal.

Table 2-10: Enable Clear Register (offset 0x14)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 disables an interrupt for the corresponding signal.

Table 2-11: Edge Register (offset 0x18)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 indicates that the interrupt has been set to edge-sensitive.

Table 2-12: Level Register (offset 0x1C)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 indicates that the interrupt has been set to level-sensitive.

Table 2-13: Set Level Register (offset 0x20)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to set interrupts to active high or rising edge

Table 2-14: Clear Level Register (offset 0x24)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to set interrupts to active low or falling edge

Table 2-15: Any Edge Set Register (offset 0x28)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to override interrupt edge selection & instead interrupt on ANY edge

Table 2-16: Any Edge Clear Register (offset 0x2C)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to clear edge sensitive override

Table 2-17: Interrupt Clear Register (offset 0x30)

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to clear edge sensitive interrupts

Table 2-18: Control Register (offset 0x34)

Bits	Access	Default	Description
[31:1]	R/W	0x00000000	Reserved
[0]	R/W	0x0	Loopback bit - write 1 to place GPIO in loopback mode. This internally assigns the ioLinesOut to the synchronized version of ioLinesIn. Read for the state of loopback. 1 = loopback mode, 0 = normal operation.

2.5 Functional Description

Each GPIO line can be independently programmed as an input or an output. Separate Set and Clear registers are provided since it is likely that different software tasks may be servicing different I/O signals. The separate Set and Clear registers make read-modify-write operations with interrupts disabled unnecessary.

It is also possible to use a single I/O as a dedicated output AND a separate dedicated input. For instance, ioLinesOut[3] can drive an LED while ioLinesIn[3] reads a switch. In this case, gpioOenN[3] would be unused.

Inputs are synchronized to the system clock through a pair of flip-flops. Each input can be programmed to cause an interrupt to be generated. The interrupt can be programmed to be level-sensitive or edge-sensitive and the level (high or low) or edge (rising, falling or either) that causes the interrupt can be selected. Interrupts can be individually enabled or disabled. Level-sensitive interrupts stay asserted until the interrupting condition is cleared. Edge-triggered interrupts are cleared by writing to the GPIO interrupt clear register.

A loopback function optionally routes the output signals to the input of the interrupt level and edge detection logic. In loopback mode, the state of the output signals is also reflected in the state of the input registers.

2.6 I/O Control

The I/O Control sub-module drives the ioLinesOut outputs from the GPIO module, and handles the synchronization of the ioLinesIn by a standard double-registering of these signals to the system clock. The output of the synchronization block drives the interrupt generation logic in the case where the loopback mode is not selected. No other conditioning (e.g. debouncing) on the input signals is done here.

Neither the I/O Control sub-module nor the GPIO module instantiates the bi-directional buffer logic that is necessary for a full-function GPIO. It is expected that the following Verilog code (or similar) be placed at a system top-level in order to accomplish this.

```
inout [NUM_LINES-1:0] gpio;

assign gpio = gpioOeN ? {(NUM_LINES){1’bZ}} : ioLinesOut;

assign ioLinesIn = gpio;
```

2.7 Timing

The timing of the GPIO APB interface relies on the APB Master that is initiating the transaction. Usually, the transaction is initiated by an AHB to APB Bridge module, and passed through to the GPIO module via an APB Channel module.

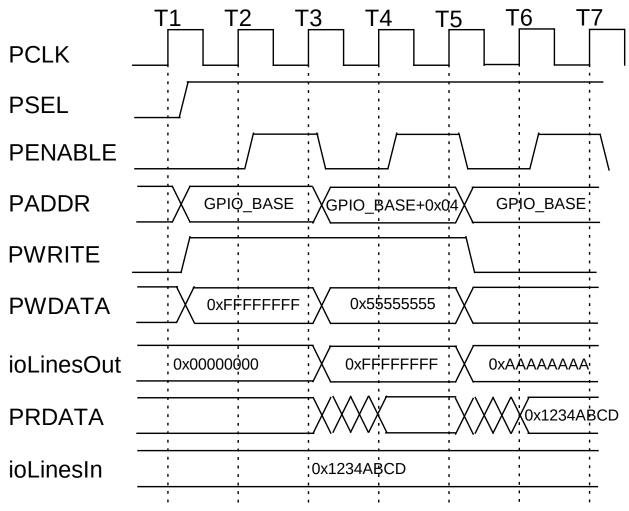


Figure 2-2: APB Interface Timing

Time T1:

The APB Master drives a new transfer on the APB bus at address GPIO_BASE + 0x00. This transfer is a write to the GPIO Set/State Register.

Time T2:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T3.

The GPIO module samples address GPIO_BASE + 0x00 along with PWRITE = 1’b1 and PSEL = 1’b1. PWDATA will be sampled at time T3.

Time T3:

The APB Master drives PENABLE = 1’b0. Also, the APB Master drives a new transfer on the APB bus at address GPIO_BASE + 0x04. This transfer is a write to the GPIO Clear/Interrupts Register.

The GPIO module samples PENABLE = 1’b1, and latches 0xFFFFFFFF from PWDATA. This write has the effect of setting all outputs to 1’b1 in the signal ioLinesOut immediately after the clock edge.

Time T4:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T5.

The GPIO module samples GPIO_BASE + 0x04 along with PWRITE = 1’b1 and PSEL = 1’b1. PWDATA will be sampled at time T5.

Time T5:

The APB Master drives PENABLE = 1’b0. Also, the APB Master drives a new transfer on the APB bus at address GPIO_BASE + 0x00. This transfer is a read of the Set/State Register.

The GPIO Module samples PENABLE = 1’b1, and latches 0x55555555 from PWDATA. This write has the effect of setting every other output to 1’b0 in the signal ioLinesOut immediately after the clock edge. Note that this write only effects the bit positions of ioLinesOut which were written with 1’b1; all bit positions that were written with 1’b0 remain unchanged.

Time T6:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T7.

The GPIO module samples GPIO_BASE + 0x00 along with PWRITE = 1’b0 and PSEL = 1’b1. As a result, the GPIO module drives PRDATA = 0x1234ABCD (reflecting the state of the ioLinesIn signal) back toward the APB Master.

Time T7:

The APB Master drives PENABLE = 1’b0. Since the current transaction is a read, the APB Master samples PRDATA = 0x1234ABCD at this time.

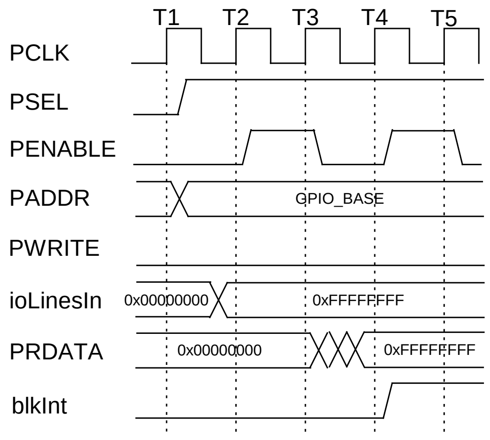


Figure 2-3: Input and Interrupt Timing

Time T1:

The APB Master drives a new transfer on the APB bus at address GPIO_BASE + 0x00. This transfer is a read from the GPIO Set/State Register.

Sometime after T1 and before T2, ioLinesIn changes from 0x00000000 to 0xFFFFFFFF.

Time T2:

The APB Master drives PENABLE = 1'b1 to indicate to the slave that the APB data phase will occur at time T3.

The GPIO module samples address GPIO_BASE + 0x00 along with PWRITE = 1'b0 and PSEL = 1'b1. As a result, the GPIO module drives PRDATA = 0x00000000 (reflecting the state of the double-registered ioLinesIn signal) back toward the APB Master. The GPIO synchronization logic begins to process the ioLinesIn signal that has changed from its previous value. Note that there are two cycles of delay (T2 and T3) between the time that ioLinesIn changes and the time where this change can be read via the GPIO Set/State Register. This is the effect of double-registering of the ioLinesIn signal to synchronize it to the PCLK domain.

Time T3:

The APB Master samples PRDATA = 0x00000000 and drives PENABLE = 1'b0. Also, the APB Master drives a new transfer on the APB bus at address GPIO_BASE. This transfer is another read from the GPIO Set/State Register.

The GPIO module samples PENABLE = 1'b1, indicating the end of the read transaction. Just after the T3 clock edge, the output from the synchronizing module changes to 0xFFFFFFFF.

Time T4:

The APB Master drives PENABLE = 1'b1 to indicate to the slave that the APB data phase will occur at time T5.

The GPIO module samples GPIO_BASE + 0x00 along with PWRITE = 1'b0 and PSEL = 1'b1. As a result, the GPIO module drives PRDATA = 0xFFFFFFFF (reflecting the state of the double-registered ioLinesIn signal) back toward the APB Master. The GPIO module detects a change in the input signal at this point, and can begin to drive blkInt = 1'b1 if interrupts are enabled, and configured to be sensitive to a rising edge or high level.

Time T5:

The APB Master samples PRDATA = 0xFFFFFFFF and drives PENABLE = 1'b0.

3. Universal Asynchronous Receiver / Transmitter

3.1 Overview

This is a complete implementation of a 16550 UART. The UART contains the following main sections:

Configuration Registers

Baud Rate Generator

Transmitter

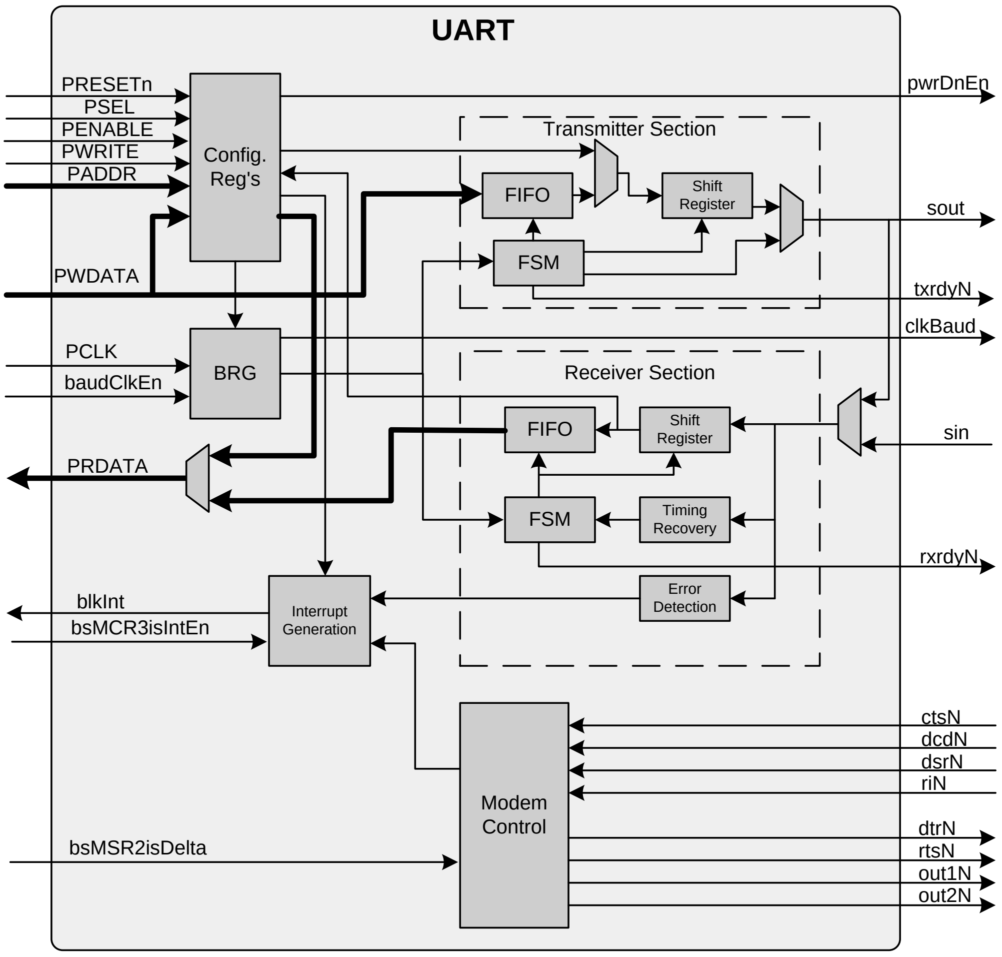
Receiver

Interrupt Generation Logic

Modem Control Logic

Configuration registers are written and read by the processor. The baud rate generator produces timing strobes at the baud rate (for the transmitter) and at 16 times the selected baud rate (for the receiver). The receiver examines the incoming data and uses the first edge of the start bit to determine the bit timing. The transmit and receive paths can be configured to use a single register for data or to use FIFOs. Finite State Machines (FSMs) control the transmit and receive sections. Various error conditions can cause an interrupt to be generated.

3.2 Block Diagram



3.3 Signal Descriptions

Table 3-1: UART Interface Signals

Signal Name	I/O	Description
APB Interface		
PRESETn	I	APB reset (active low)
PCLK	I	APB clock
PWRITE	I	APB write, 0=Read and 1=Write.
PWDATA[31:0]	I	APB write data bus
PADDR[2:0]	I	APB address bus, if the registers are expected to be on word boundaries, then PADDR[4:2] are connected to this input, otherwise the registers are located on byte boundaries and PADDR[2:0] is connected to this input.
PSEL	I	APB Block select; this signal must be set to access any of the UART's registers.
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	APB read data bus
Interrupt Interface		
blkInt	O	Block interrupt, this signal is set when any of the following interrupt sources become active: receiver line status, received data available, character timeout, transmitter holding register empty, or modem status.
Modem Interface		
ctsN	I	Clear to send
dcdN	I	Data carrier detect
dsrN	I	Data set ready
riN	I	Ring indicator
dtrN	O	Data terminal ready
rtsN	O	Request to send
out1N	O	General purpose output controlled by MCR[2].
out2N	O	General purpose output controlled by MCR[3].
Serial Interface		
sin	I	Serial data in
sout	O	Serial data out
rxrdyN	O	Receiver ready
txrdyN	O	Transmitter ready
Miscellaneous Signals		
clkBaud	O	Baud clock, can be used by external receivers as a sampling clock, the frequency is (baud rate x 16).
baudClkEn	I	Baud clock enable: series of 1-clock wide pulses for dividing down PCLK to get 16X baud clock
regSCR[7:0]	O	The contents of the Scratchpad Register.
pwrDnEn	O	pwrDnEn is a general-purpose output signal.
bsMCR3isIntEn	I	Bootstrap input, when this input is tied high, MCR[3] enables interrupts.
bsMSR2isDelta	I	Bootstrap signal, when this input is tied low, MSR[2] is set on the trailing edge (low to high transition) of the active low ring indicator input (aka "TERI" or Trailing Edge Ring Indicator mode). When this input is tied high, MSR[2] is set on any edge of the ring indicator input (aka "DRI" or Delta Ring Indicator mode).

3.4 Register Descriptions

Table 3-2: UART Registers Summary

Register Name	Abbreviation	Offset	Description
Receiver Buffer	RBR	0x0	Received Data (Read Only)

Register Name	Abbreviation	Offset	Description
Transmitter Holding	THR	0x0	Data to be transmitted (Write Only)
Interrupt Enable	IER	0x4	Enables the five types of UART interrupts.
Interrupt Identification	IIR	0x8	Four interrupts levels identified here in order of priority – Receiver Line Status, Received Data Ready, Transmitter Holding Register Empty, and MODEM Status (Read Only)
FIFO Control	FCR	0x8	Enable FIFOs, clear FIFOs, set RCVR FIFO trigger level (Write Only)
Line Control	LCR	0xC	Specifies asynchronous data communications exchange format and sets the Divisor Latch Access Bit
MODEM Control	MCR	0x10	Controls interface with MODEM (or peripheral device emulating a MODEM)
Line Status	LSR	0x14	Data Transfer Status information
MODEM Status	MSR	0x18	Current state of control lines from MODEM (or peripheral device) to CPU
Scratch	SCR	0x1C	Register can be used to hold temporary data
Divisor Latch (LS)	DLL	0x0	DLAB = 1. Least significant byte for input to baud rate generator
Divisor Latch (MS)	DLM	0x4	DLAB = 1. Most significant byte for input to baud rate generator

Table 3-3: Receiver Buffer Register (offset 0x00)

Bits	Access	Default	Description
[7:0]	Read	0x00	Received Data

Table 3-4: Transmitter Holding Register (offset 0x00)

Bits	Access	Default	Description
[7:0]	Write	0x00	Data To be transmitted

Table 3-5: Interrupt Enable Register (offset 0x04)

Bits	Access	Default	Description
[7:6]	Read/Write	0x0	Unused
[5]	Read/Write	0x0	Along with MCR[7], set this bit to cause pwrDnEn = 1
[4]	Read/Write	0x0	Unused
[3]	Read/Write	0x0	Enable MODEM Status Interrupt
[2]	Read/Write	0x0	Enable Receiver Line Status Interrupt
[1]	Read/Write	0x0	Enable Transmitter Holding Register Interrupt
[0]	Read/Write	0x0	Enable Received Data Available Interrupt

Table 3-6: Interrupt Identification Register (offset 0x08)

Bits	Access	Default	Description
[7:6]	Read	0x0	FIFOs Enabled
[5:4]	Read	0x0	Unused – value is set to 0
[3:1]	Read	0x0	Interrupt Identification Bits – see “Interrupt Identification” section
[0]	Read	0x1	Interrupt Pending – ‘0’ if interrupt pending

Table 3-7: FIFO Control Register (offset 0x08)

Bits	Access	Default	Description
[7:6]	Write	0x0	Receiver Trigger Level – see “FIFO Control” section
[5:4]	Write	Undefined	Reserved

[3]	Write	0x0	DMA Mode Select
[2]	Write	0x0	Transmit FIFO reset (self-clearing)
[1]	Write	0x0	Receive FIFO reset (self-clearing)
[0]	Write	0x0	FIFO Enable

Table 3-8: Line Control Register (offset 0x0C)

Bits	Access	Default	Description
[7]	Read/Write	0x0	Divisor Latch Access (DLAB)
[6]	Read/Write	0x0	Set Break
[5]	Read/Write	0x0	Stick Parity
[4]	Read/Write	0x0	Even Parity Select
[3]	Read/Write	0x0	Parity Enable
[2]	Read/Write	0x0	Number of Stop Bits
[1:0]	Read/Write	0x0	Word Length Select – see “Line Control” section

Table 3-9: Modem Control Register (offset 0x10)

Bits	Access	Default	Description
[7]	Read/Write	0x0	Along with IER[5], set this bit to cause pwrDnEn = 1
[6:5]	Read/Write	0x0	Reserved
[4]	Read/Write	0x0	Loopback mode select
[3:2]	Read/Write	0x0	Auxiliary user designated output
[1]	Read/Write	0x0	Request to Send (RTS)
[0]	Read/Write	0x0	Data Terminal Ready (DTR)

Table 3-10: Line Status Register (offset 0x14)

Bits	Access	Default	Description
[7]	Read	0x0	Error in Receiver FIFO
[6]	Read	0x0	Transmitter Empty
[5]	Read	0x0	Transmitter Holding Register Empty
[4]	Read	0x0	Break Interrupt
[3]	Read	0x0	Framing Error
[2]	Read	0x0	Parity Error
[1]	Read	0x0	Overrun Error
[0]	Read	0x0	Data Ready (DR)

Table 3-11: Modem Status Register (offset 0x18)

Bits	Access	Default	Description
[7]	Read	0x0	Data Carrier Detect (DCD)
[6]	Read	0x0	Ring Indicator (RI)
[5]	Read	0x0	Data Set Ready (DSR)
[4]	Read	0x0	Clear To Send (CTS)
[3]	Read	0x0	Delta Data Carrier Detect (DDCD)
[2]	Read	0x0	Trailing Edge Ring Indicator (TERI)
[1]	Read	0x0	Delta Data Set Ready (DDSR)
[0]	Read	0x0	Delta Clear To Send (DCTS)

Table 3-12: Scratch Register (offset 0x1C)

Bits	Access	Default	Description
[7:0]	Read/Write	0x00	Scratchpad register

Table 3-13: Divisor Latch (LS) (offset 0x00)

Bits	Access	Default	Description
[7:0]	Read/Write	0x01	Divisor Latch [7:0]

Table 3-14: Divisor Latch (MS) (offset 0x04)

Bits	Access	Default	Description
[7:0]	Read/Write	0x00	Divisor Latch [15:8]

3.5 Functional Description

3.5.1 Transmitter Operation

The UART Transmitter consists of the transmit holding register (THR), the transmit FIFO, a shift register, a state machine, and a multiplexor. When in FIFO mode, writing to the THR causes the THR value to be written into the FIFO. If there is no transmission in progress, transmission is started. The state machine controls the shift register and mux to insure the bits get transmitted in the proper sequence. The baud rate generator controls the pacing of the transmission. Parity is calculated if required and inserted into the bit stream.

3.5.2 Receiver Operation

Since another independent UART is transmitting the signal, timing has to be recovered from the incoming serial bit stream. The falling edge of the start bit starts a counter that counts into the middle of the bit time programmed into the baud rate generator. Thereafter, midBit strobes are generated which occur in the middle of the bit time and control the timing for the duration of the received word. The timing of the far end transmitter can be substantially different from that of the receiver and the word will be successfully received as long as the cumulative error across the word (including start and stop bits) does not exceed one half of the bit time.

A state machine sequences the receive operation. When all bits are received, parity is calculated and compared to the expected parity. Parity errors, framing errors, and break conditions are associated with a received data word and are written into the FIFO along with the data. A framing error is defined as the absence of an expected stop bit. A break condition is declared when too many space symbols occur in a row, that is when the serial input stays in the logic 0 state for a number of bit times greater than the sum of the number of start, stop, parity and data bits.

3.5.3 Baud Rate Generator

The baud rate generator can be configured to generate a wide range of baud rates, depending on the system clock frequency and the divisor latch. The divisor latch and the system clock frequency are related to the baud rate by the following expression:

$$BaudRate = \frac{SystemClockFrequency}{16 \times DivisorLatch}$$

For example, to configure the UART for a baud rate of 2400 with a system clock frequency of 18.432 MHz, the Divisor Latch would need to be decimal 480 (0x01E0), so the most significant byte of the divisor latch would be programmed 0x01 and the least significant byte of the divisor latch would be programmed 0xE0. To access the divisor latch, the Divisor Latch Access Bit (DLAB) must be set in the LCR.

The baud rate generator also generates the clkBaud output signal that can be used to provide a timing reference to external receivers. The clkBaud output frequency is 16 times the baud rate.

$$clkBaudFrequency = \frac{SystemClockFrequency}{DivisorLatch}$$

See the Timing Diagrams section for a detailed description of the clkBaud output.

The divisor latch can be programmed values ranging 0x0001 to 0xFFFF.

3.5.3.1. Optional Divisor for Baud Rate Generator

There is an optional divisor that can be implemented within the system to approximate a 1.8432MHz “UART” oscillator to the UART when the system clock frequency is 16.000MHz. This divisor is implemented when the USE_UART_BAUD_CLK_DIV has been defined in socSysDefs.hv. This divisor yields an extra factor of 8.5 (since 16.000/1.8432 ~= 8.5) in the BaudRate and clkBaudFrequency equations. Please use the following equations to determine the appropriate value to program the divisor latches when using this optional divisor.

$$BaudRate = \frac{SystemClockFrequency}{8.5 \times 16 \times DivisorLatch}$$

$$clkBaudFrequency = \frac{SystemClockFrequency}{8.5 \times DivisorLatch}$$

For example, to achieve 2400 baud with a 16.368MHz oscillator, a divisor latch of 0x0032 (decimal 50) would be necessary since (16368000/(8.5 * 16 * 50) = 2407, which approximates 2400).

3.5.4 Interrupts

The UART can generate five types of interrupts: receiver line status, received data available, character timeout, transmitter holding register empty, and modem status. Any of these interrupts can activate the UART block interrupt (blkInt) if they are enabled.

3.5.4.1. Enabling Interrupts

The enabling of these interrupts is controlled by the setting and clearing of bits in the IER. A set bit in the IER corresponds to an enabled interrupt while a clear bit in the IER corresponds to a disabled interrupt.

Bit 3 of the MCR can be optionally used as a global interrupt enable for the UART when the bootstrap input bsMCR3isIntEn is tied to a logic ‘1’. When bsMCR3isIntEn is tied to a logic ‘1’, MCR[3] acts as a global interrupt enable for the UART. If MCR[3] is clear while configured as an interrupt enable, the UART will not activate its interrupt signal (blkInt) under any condition. If MCR[3] is set while configured as an interrupt enable, the UART will activate its interrupt signal only if one of the five interrupt sources is active and enabled by the appropriate bit in the IER. When bsMCR3isIntEn is tied to a logic ‘0’, MCR[3] has no effect on the UART interrupt logic and the UART will activate its interrupt signal any time one of the five interrupt sources is active and enabled by the appropriate bit in the IER.

Table 3-15: UART Interrupts

IER Bit Number	Corresponding Interrupt Enabled
0	Received Data Available
0	Character Timeout (only when FIFOs are enabled)
1	Transmitter Holding Register Empty
2	Receiver Line Status
3	Modem Status

3.5.4.2. Interrupt Types

3.5.4.2.1. Receiver Line Status

This interrupt occurs when any one of the following conditions exist: framing error, parity error, overrun error, or break interrupt. This interrupt is cleared by reading the line status register.

3.5.4.2.2. Received Data Available

When FIFOs are disabled, this interrupt occurs when the RBR contains received data. Reading the RBR clears this interrupt.

When FIFOs are enabled, this interrupt occurs when the amount of data in the receiver FIFO exceeds the trigger level (controlled by FCR[7:6]). This interrupt is cleared when the amount of data in the receiver FIFO drops below the trigger level. This may require multiple reads from the RBR.

3.5.4.2.3. Character Timeout

This interrupt only occurs when FIFOs are enabled and no characters have been written to or read from the FIFO in the last 4 character times. Each character time consists of the start bit, data bits, parity bit (if enabled), and stop bit(s). This interrupt is cleared by reading the RBR.

3.5.4.2.4. Transmitter Holding Register Empty

When FIFOs are disabled, this interrupt occurs when a character from the THR is loaded into the transmitter shift register, indicating that the THR is ready to accept a new character.

When FIFOs are enabled, this interrupt occurs when the transmitter FIFO becomes empty.

In either case, this interrupt is cleared when the THR is read, or when the IIR was read when THRE was the source of the interrupt.

Note that reading the IIR clears the THRE interrupt, however the THRE bit in the LSR will remain set until a character has been loaded into the THR or transmitter FIFO.

3.5.4.2.5. Modem Status

The modem status interrupt occurs when any of the following bits in the LSR is set: delta clear to send (DCTS), delta data set ready (DDSR), trailing edge ring indicator (TERI), or delta data carrier detect (DDCD). This interrupt is cleared by reading the MSR.

Be aware that in loopback mode these “delta” bits are not controlled by the modem inputs but rather by the lower four bits of the MCR. See the section on Loopback Operation for a more detailed description of generating modem status interrupts in loopback mode.

3.5.4.3. Interrupt Identification

Interrupts generated by the UART are distinguished by the value in the least significant nibble of the interrupt identification register. Bit 0 of the IIR can be tested to determine if there is an interrupt pending.

Table 3-16: UART Interrupt Types

Priority	IIR [3:0]	Interrupt Type
n/a	0001	No Interrupt Pending
first	0110	Receiver Line Status
second	0100	Received Data Available
second	1100	Character Timeout
third	0010	Transmitter Holding Register Empty
fourth	0000	Modem Status

3.5.5 FIFO Control

FIFOs are enabled by setting bit 0 in the FCR. When FIFOs are enabled, bits 7 and 6 of the IIR are set, otherwise these bits are clear.

3.5.6 Receiver FIFO

The receiver FIFO is used to store received data until it is read by software. Up to 16 bytes can be stored in the receiver FIFO. Once the programmed trigger level has been reached within the FIFO, the data ready bit (and received data available interrupt if enabled) will be set. Bits 7 and 6 of the FCR register control the receiver trigger level.

Table 3-17: UART FIFO Trigger Levels

FCR[7:6]	Receiver FIFO Trigger Level (Bytes)
00	1
01	4
10	8
11	14

The receiver FIFO can be reset by setting bit 1 in the FCR. This bit is self-clearing.

3.5.7 Transmitter FIFO

The transmitter FIFO is used to store data that is intended to be transmitted. The transmitter FIFO can store up to 16 bytes. Each byte written to the THR location is loaded into the transmitter FIFO. The transmitter FIFO can be reset by setting bit 2 in the FCR. This bit is self-clearing.

3.5.8 Asynchronous Serial Data Format

The line control register (LCR) specifies the format of the asynchronous data that is transmitted and received by the UART. The number of data bits in each serial character is specified by bits 1 and 0 of the LCR.

Table 3-18: UART Character Length

LCR[1:0]	Character Length
00	5 Bits
01	6 Bits
10	7 Bits
11	8 Bits

The number of stop bits in each serial character is specified by bit 2 of the LCR along with the current character length selected by bits 1 and 0 of the LCR.

Table 3-19: UART Stop Bits

LCR[2]	Number of Stop Bits
0	1
1	2 *

Note that this is a difference from a standard 16550 implementation in that when the character length is 5 bits, the number of stop bits is supposed to be 1.5. For simplicity, this implementation generates 2 stop bits.

Parity bit generation is controlled by bits 4 and 3 of the LCR. Bit 3 enables parity and bit 4 selects even or odd parity.

Table 3-20: UART Parity Type

LCR[4:3]	Type of Parity
X0	No Parity
01	Odd Parity
11	Even Parity

Even parity sets the parity bit to a logic ‘1’ when there are an even number of ‘1’s in the serial character and sets the parity bit to a logic ‘0’ when there are an odd number of ‘1’s in the serial character. Odd parity sets the parity bit to a logic ‘1’ when there are an odd number of ‘1’s in the serial character and sets the parity bit to a logic ‘0’ when there are an even number of ‘1’s in the serial character.

3.5.9 Diagnostic Mechanisms

The line control register also provides mechanisms to force break conditions, as well as parity and framing error conditions. Utilizing these mechanisms while in loopback mode allow for diagnostic checking of receiver line logic.

Bit 5 of the LCR is the Stick Parity bit. Stick Parity causes the transmitter to force the parity bit of a transmitted serial character to a logic ‘1’ or logic ‘0’ regardless of the calculated parity of the transmitted serial character. Stick Parity also causes the receiver to check the parity bit of a received serial character against either a logic ‘1’ or logic ‘0’ regardless of the calculated parity of the received serial character. It is the Stick Parity bit in conjunction with bits 4 and 3 of the LCR which determine how the parity bit is affected.

The following table summarizes all possible combinations of parity functionality:

Table 3-21: UART Parity Functionality

LCR[5]	LCR[4:3]	Type of Parity
0	00	parity disabled, no parity bit generated/checked
0	01	odd parity generated/checked
0	10	parity disabled, no parity bit generated/checked
0	11	even parity generated/checked
1	00	parity disabled, no parity bit generated/checked
1	01	stick parity, generated/checked as logic ‘1’
1	10	parity disabled, no parity bit generated/checked
1	11	stick parity, generated/checked as logic ‘0’

Bit 6 of the LCR is the Set Break bit. Setting this bit forces sout low (spacing) condition. The sout line will remain in the spacing condition until this bit is cleared.

Bit 7 of the LSR is the Divisor Latch Access Bit (DLAB). Please see the section on the Baud Rate Generator for a description of the use of this bit.

3.5.10 Line Status

Bit 0 of the LSR is the Data Ready (DR) bit. This bit is set when an incoming character is completely received into the RBR or receiver FIFO and cleared once the character from the RBR is read or once the receiver FIFO has been emptied.

Bit 1 of the LSR is the overrun error indicator. Overrun errors occur in 16450 mode when both the RBR and receiver shift register contain data, and another incoming character is received, destroying the character in the receiver shift register. The character in the RBR is not corrupted.

In 16550 mode, overrun errors occur when the receiver FIFO is completely full, the receiver shift register contains a character, and another incoming character is received, destroying the character in the receiver shift register. The data in the receiver FIFO is not corrupted.

In both modes, the overrun error indicator bit is set as soon as the start bit of the character causing the overrun is detected.

Bit 2 of the LSR is the parity error indicator. This bit is set when a received character’s parity (as calculated by the receiver) does not match the value of its received parity bit. In 16550 mode, this bit is set only when the offending character is at the top of the FIFO. Reading the LSR clears this bit.

Bit 3 of the LSR is the framing error indicator. This bit is set when the receiver does not detect a valid stop bit for an incoming character, i.e. the serial input signal was low during the baud time in which a stop bit (high) was expected. In 16550 mode, this bit is set only when the offending character is at the top of the FIFO. Reading the LSR clears this bit.

Bit 4 of the LSR is the break interrupt indicator. This bit is set whenever the serial input is held in the space (logic 0) state for more time than a complete character transfer (i.e. the time it takes to transmit a start bit, the data bits, the parity bit if enabled, and any stop bits). If a break is detected in 16550 mode, only one break character (0x00) is loaded into the receiver FIFO. Reading the LSR clears this bit.

Bit 5 of the LSR is the Transmitter Holding Register Empty bit. When FIFOs are disabled, this bit is set when a character from the THR is loaded into the transmitter shift register, indicating that the THR is ready to accept a new character. It is cleared when a new character is written to the THR.

When FIFOs are enabled, this bit is set when the transmitter FIFO becomes empty and is cleared after at least one character is loaded into the transmitter FIFO.

Bit 6 of the LSR is the Transmitter Empty Bit. This bit is set when both the THR and the transmitter shift register are empty. This bit is clear when either the THR contains a character or the transmitter shift register has not completed shifting out a character being transmitted.

Bit 7 of the LSR indicates that there is at least one error in the receiver FIFO. In 16450 mode this bit is always clear. In 16550 mode, this bit is set when there is at least one framing error, parity error, or break indication in the receiver FIFO. Reading the LSR clears this bit.

3.5.11 Modem Control / Status

The Modem Control and Modem Status Registers (MCR and MSR) are used to control and monitor the modem related I/O: Clear to Send, Data Set Ready, Ring Indicator, Data Carrier Detect, Data Terminal Ready, and Request to Send.

3.5.12 Modem Control

Bits 0 and 1 of the MCR directly affect the state of the Data Terminal Ready and Request to Send outputs (dtrN and rtsN). Setting bit 0 of the MCR causes the dtrN output to be asserted (low) while clearing bit 0 of the MCR causes dtrN to be deasserted (high). In a similar manner, setting bit 1 of the MCR causes the rtsN output to be asserted (low) while clearing bit 1 of the MCR causes rtsN to be deasserted (high).

3.5.13 Modem Status

Bits 4-7 of the MSR indicate the current state of the Clear to Send, Data Set Ready, Ring Indicator, and Data Carrier Detect input signals, respectively (ctsN, dsrN, riN, dcdN). On the condition that any of these input signals are active (low), then the corresponding bits in the MSR will be set. When any of these inputs are in their inactive (high) state, then the corresponding bits in the MSR will be clear.

Bits 0-3 of the MSR are known as the “delta” bits which indicate whether the state of the modem input signals have changed since a previous read of the MSR. The Delta Clear to Send (DCTS), Delta Data Set Ready (DDSR), and Delta Data Carrier Detect (DDCD) bits in the MSR are asserted any time one of the corresponding modem inputs (ctsN, dsrN, dcdN) changes state.

The “delta” bit for the Ring Indicator can be configured as a Delta Ring Indicator (DRI) or as a Trailing Edge Ring Indicator (TERI) based on the state of the bootstrap input bsMSR2isDelta. When this bootstrap is tied high (configured for DRI), bit 2 of the MSR will be asserted any time the state of the riN input changes. Alternatively, bsMSR2isDelta input can be tied low (configured for TERI), and bit 2 of the MSR will only be asserted under the condition that there is a low to high transition (active low trailing edge) of the riN input.

3.5.14 DMA modes

There are two modes of DMA operation. These modes affect the operation of the active low txrdyN and rxrdyN signals. In 16450 mode, only Mode 0 is supported. In 16550 mode, both Mode 0 and Mode 1 are supported and are selected by bit 3 in the FCR.

3.5.15 Mode 0 (16450 and 16550)

In this mode, rxrdyN will be asserted (low) when there is at least one character in either the RBR or the receiver FIFO and will be deasserted (high) when the RBR or the receiver FIFO are empty. The txrdyN signal will be asserted (low) when the THR or the transmitter FIFO are empty and will be deasserted (high) immediately after a character has been loaded into the THR or the transmitter FIFO.

3.5.16 Mode 1 (16550 only)

In this mode, rxrdyN will be asserted (low) when the amount of characters in the receiver FIFO reaches the receiver FIFO trigger level. A character timeout will also cause rxrdyN to be asserted (low). The rxrdyN signal will be deasserted (high) once the receiver FIFO has been emptied. The txrdyN signal will be asserted (low) when the transmitter FIFO is empty and will be deasserted (high) once the transmitter FIFO has been completely filled.

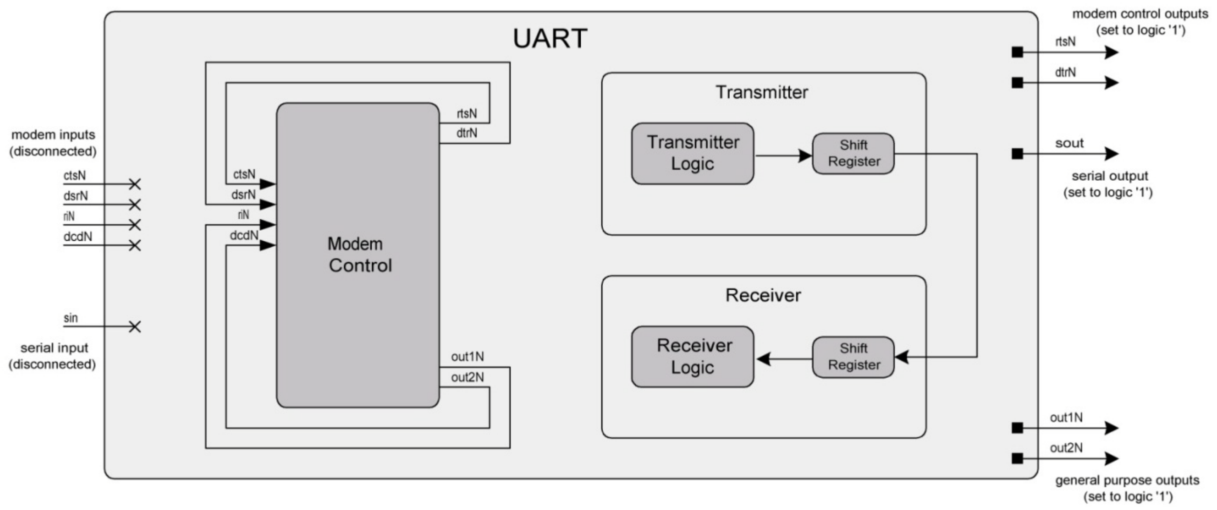
3.5.17 Loopback Operation

The UART provides loopback operation for transmitter, receiver, and modem status diagnostics. The UART operates in loopback mode when bit 4 of the MCR is set.

In loopback mode, the serial data input (sin) is disconnected, and the serial data output (sout) is driven high (marking state). The transmitter is internally connected to the receiver so that any data transmitted is also received within the UART.

The ctsN, dsrN, riN, and dcdN modem inputs are disconnected, and the dtrN, rtsN, out1N, and out2N outputs are driven high (inactive state). The modem control and general purpose outputs (dtrN, rtsN, out1N, out2N) are internally connected to the modem status inputs (dsrN, ctsN, riN, dcdN).

Table 3-22: UART Loopback Operation


UART
modem inputs (disconnected)
ctsN
dsrN
riN
dcdN
sin
serial input (disconnected)
rtsN
dtrN
ctsN
dsrN
riN
dcdN
Modem Control
out1N
out2N
Transmitter
Transmitter Logic
Shift Register
Receiver
Receiver Logic
Shift Register
modem control outputs (set to logic '1')
rtsN
dtrN
sout
serial output (set to logic '1')
out1N
out2N
general purpose outputs (set to logic '1')

The modem status interrupt can still be generated, although in loopback mode it is controlled through the least four significant bits in the MCR.

For example, writing 0x11 to the MCR (asserting DTR bit and maintaining loopback bit) will cause the DSR bit to go active in the MSR (since DTR is connected to DSR in loopback mode). The “delta” edge detection logic then detects the change from low to high of the DSR bit in the MSR, therefore causing the modem status interrupt (any asserted “delta” bit in the MSR causes the modem status interrupt). Reading the MSR will clear this interrupt.

Be aware that the option to have trailing edge or delta edge detection on the ring indicator also applies in loopback mode. However, in this mode, the “delta” or trailing edge will be detected on the out1 bit in the MCR since this is connected to the Ring Indicator bit in the MSR when operating in loopback mode. This configuration allows diagnostic testing of the Trailing Edge Ring Indicator (TERI) and Delta Ring Indicator (DRI) in loopback mode through the use of the out1 bit of the MCR.

3.5.18 General Purpose Outputs

The out1N and out2N outputs are driven from bits 2 and 3 of the MCR, respectively. These outputs are active low, therefore setting bit 2 in the MCR will cause the out1N output to be driven low, while clearing bit 2 in the MCR causes the out1N output to be driven high. The out2N output is controlled in a similar fashion by bit 3 of the MCR. A system reset will set these signals to their inactive state (logic ‘1’) since bits 2 and 3 of the MCR are cleared on a system reset.

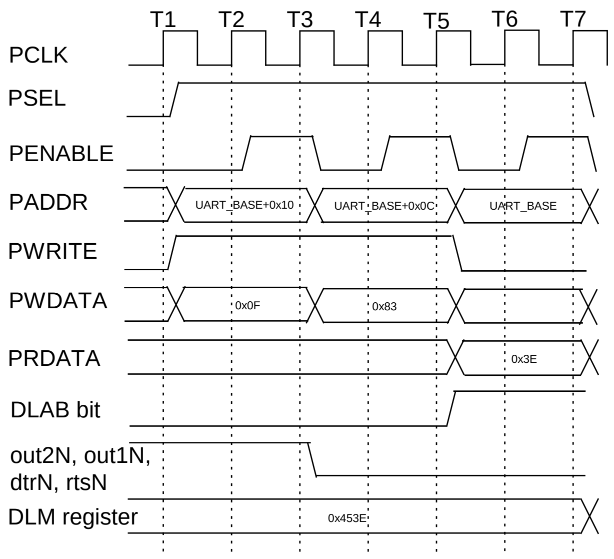
Similarly, the pwrDnEn output is normally a logic ‘0’. If MCR[7] and IER[5] are both set to a logic ‘1’, then pwrDnEn is a logic ‘1’.

3.6 Timing Diagrams

3.6.1 APB Interface Timing

The timing of the UART module’s APB interface relies on the APB Master that is initiating the transaction. Usually, the transaction is initiated by an AHB to APB Bridge module, and passed through to the UART module via an APB Channel module.

Table 3-23: APB Interface Timing



Time T1:

The APB Master drives a new transfer on the APB bus at address UART_BASE+0x10. This transfer is a write to the Modem Control Register to enable the various (active low) modem control outputs.

Time T2:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T3.

The UART module samples address UART_BASE+0x10 along with PWRITE = 1’b1 and PSEL = 1’b1. PWDATA will be sampled at time T3.

Time T3:

The APB Master drives PENABLE = 1’b0. Also, the APB Master drives a new transfer on the APB bus at address UART_BASE+0x0C. This transfer is a write to the Line Status Register to enable access to the Divisor Latch Register (via the DLAB bit) and to configure the UART line as 8 bits per character with 1 stop bit.

The UART module samples PENABLE = 1’b1, and latches 0x0F from PWDATA. This write has the effect of toggling the various modem control outputs after the clock edge.

Time T4:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T5.

The UART module samples UART_BASE+0x0C along with PWRITE = 1’b1 and PSEL = 1’b1. PWDATA will be sampled at time T5.

Time T5:

The APB Master drives PENABLE = 1’b0. Also, the APB Master drives a new transfer on the APB bus at address UART_BASE. This transfer is a read of the Divisor Latch (Least Significant Byte) Register.

The UART Module samples PENABLE = 1’b1, and latches 0x83 from PWDATA. This write has the effect of setting the DLAB bit and reconfiguring the UART line parameters after the clock edge.

Time T6:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T7.

The UART module samples UART_BASE along with PWRITE = 1’b0 and PSEL = 1’b1. As a result, the UART module drives PRDATA = 0x3E (reflecting the state of the LSB of the Divisor Latch Register) back toward the APB Master.

Time T7:

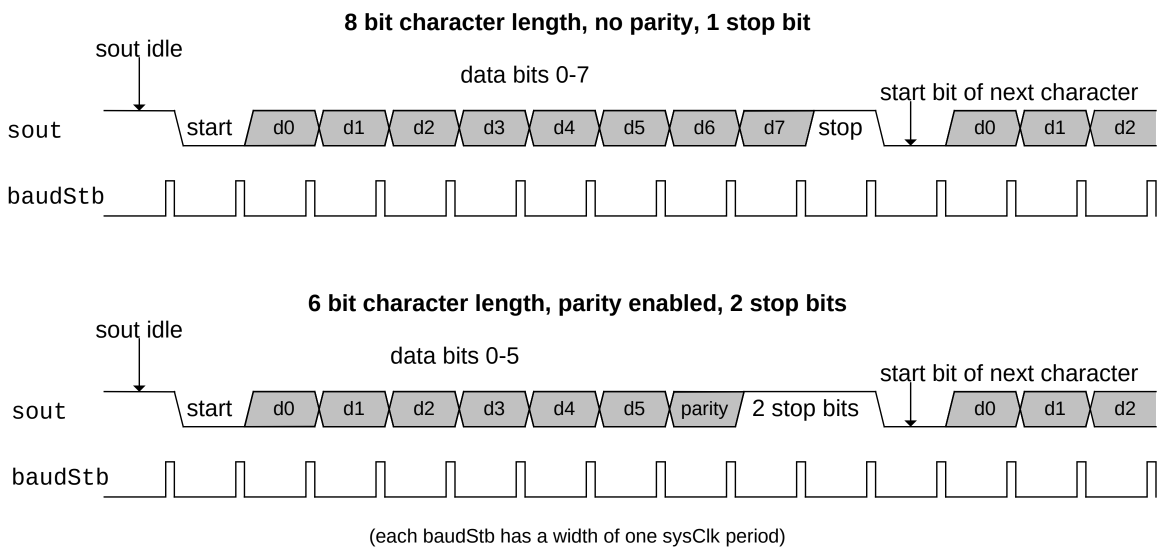
The APB Master drives PENABLE = 1’b0. Since the current transaction is a read, the APB Master samples PRDATA = 0x3E at this time.

The APB Slave does nothing of consequence on this cycle.

3.6.2 Example Transmitter Timing

baudStb is an internal signal produced by the Baud Rate Generator. It has the frequency of BaudRate. The following diagram shows two configurations of the transmitter: 8-bit character length, no parity, 1 stop bit; 6-bit character length, parity enabled, 2 stop bits.

Table 3-24: UART Example Transmitter Timing

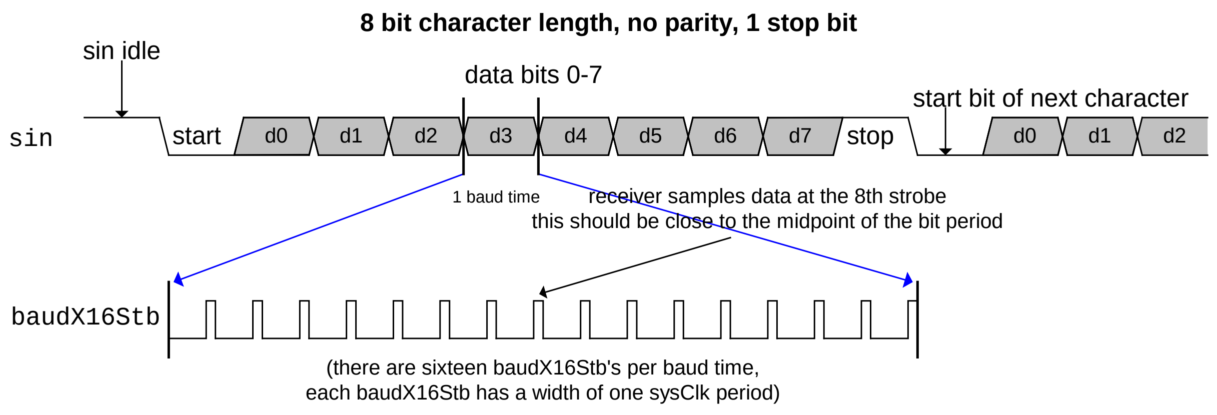

8 bit character length, no parity, 1 stop bit
sout idle
data bits 0-7
start bit of next character
sout
start
d0
d1
d2
d3
d4
d5
d6
d7
stop
baudStb
6 bit character length, parity enabled, 2 stop bits
sout idle
data bits 0-5
start bit of next character
sout
start
parity
2 stop bits
baudStb
(each baudStb has a width of one sysClk period)

3.6.3 Example Receiver Timing

baudX16Stb is an internal signal produced by the Baud Rate Generator. It has a frequency of (16 * BaudRate). The following diagram shows receiver timing for the following configuration: 8-bit character length, no parity, 1 stop bit.

The receiver attempts to sample the data near the midpoint of the baud time. However, over several baud times, there may be some drift since sin is being driven by another UART that probably is using a different clock source. As long as the drift is no more than ½ baud time over the course of the entire character transmission (start, data, parity, stop), then the transmission will be sampled correctly. This is because the UART uses each start bit to resynchronize the sampling of received data.

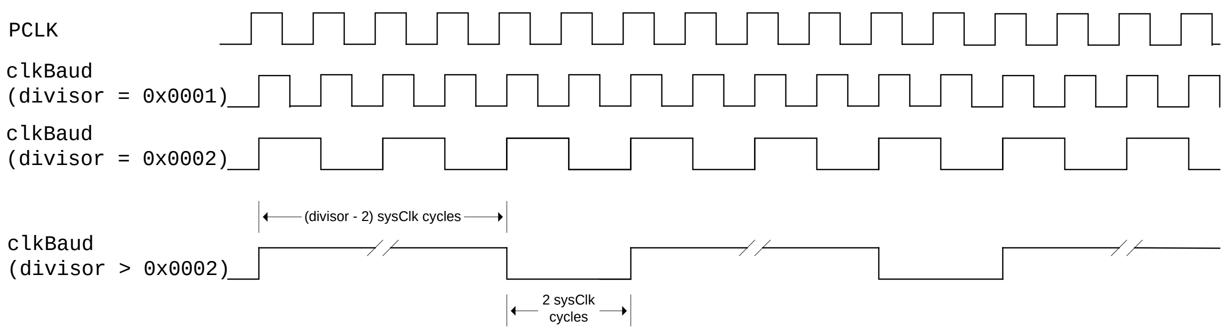
Table 3-25: UART Example Receiver Timing


8 bit character length, no parity, 1 stop bit
sin idle
data bits 0-7
start bit of next character
sin
start
d0
d1
d2
d3
d4
d5
d6
d7
stop
d0
d1
d2
1 baud time
receiver samples data at the 8th strobe
this should be close to the midpoint of the bit period
baudX16Stb
(there are sixteen baudX16Stb's per baud time,
each baudX16Stb has a width of one sysClk period)

3.6.4 PCLK & clkBaud

clkBaud is an output signal from the UART that is produced by the Baud Rate Generator. The diagram below shows the relationship of PCLK to clkBaud.

Table 3-26: UART PCLK to clkBaud Relationship

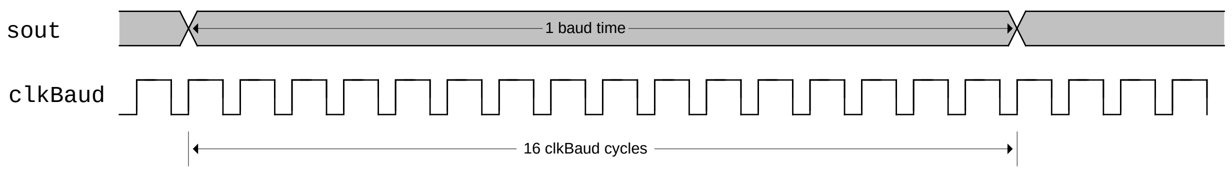


Note that the 2nd and 3rd waveforms only apply when the baudClkEn input is tied high. When the input clock is divided down using the baudClkEn signal, no special cases exist and the clkBaud signal looks like the last waveform above.

3.6.5 sout & clkBaud

clkBaud is an output signal from the UART that is produced by the Baud Rate Generator. The diagram below shows the relationship of clkBaud to the sout output signal.

Table 3-27: UART sout to clkBaud Relationship



3.7 Interrupt Generation

The interrupt generation block generates a single-bit block interrupt (blkInt) that can be connected to an interrupt input of a microprocessor. The blkInt signal is active if any enabled interrupt condition occurs; if multiple interrupt conditions caused this signal to become active, the signal will remain active until all the interrupting conditions have been cleared.

3.8 Minimal Implementation

There is a definition in the Verilog source code that is normally in place: USE_GPIO_INTERRUPTS. If a minimal implementation is desired and interrupt capability is unnecessary, it is possible to undefine USE_GPIO_INTERRUPTS by commenting out its definition. This has the effect of removing all of the internal Interrupt Generation logic and registers. In this scenario, writes to the following registers will have no effect, and reads from the following registers will appear as 0x0: Enable Set, Enable Clear, Edge, Level, Set Level, Clear Level, Any Edge Set, Any Edge Clear, Interrupt Clear. Lastly, the blkInt output will be assigned a constant 1’b0 if USE_GPIO_INTERRUPTS is not defined.

4. Timer (Event Counter)

4.1 Overview

The APB Timer module is a sixteen-bit down counter with a selectable prescaler. Prescale values of 1, 16 and 256 can be selected. The prescaler extends the timer's range at the expense of precision.

The Timer provides two modes of operation that provide a free running value and also periodic interrupts.

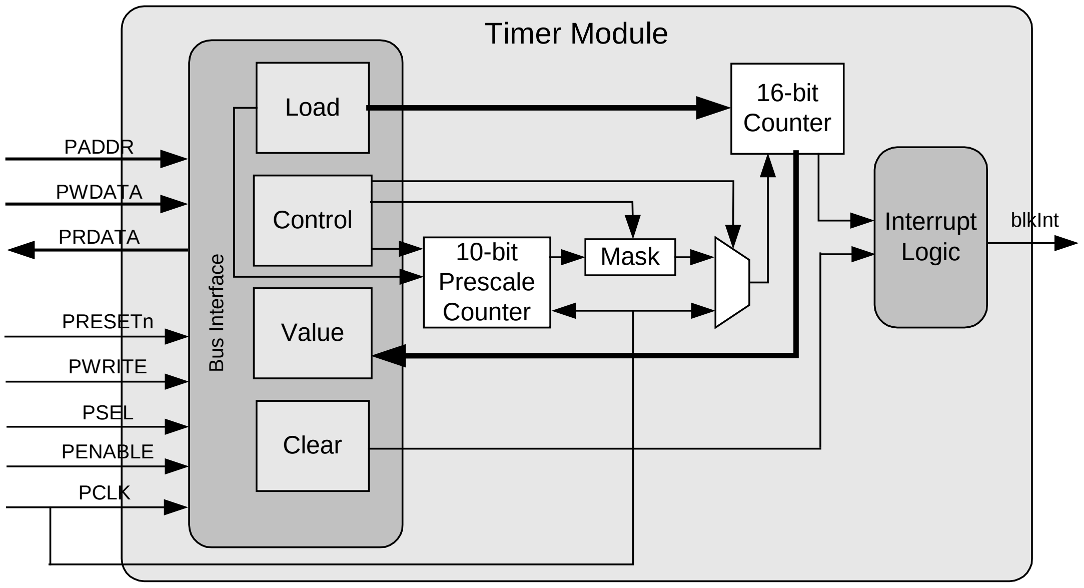
The Timer contains several configuration registers that can be written and read by the processor. Two 4-bit prescalers precede a 16-bit counter. The counter can be clocked at either the input clock rate, or a choice of 7 prescaled rates. The counter can be loaded with a value from a preload register. The counter can optionally generate an interrupt.

The Timer module is a standard APB Slave peripheral; the Timer registers are accessed through this interface.

The timer is a 16-bit peripheral but it is assumed that the addressing will be adjusted so that each register is on a 32-bit boundary.

4.2 Block Diagram

Table 4-1: Timer Block Diagram



4.3 Signal Descriptions

Table 4-2: Timer Interface Singals

Signal Name	I/O	Description
APB Slave Interface		
PRESETn	I	APB reset (active low)
PCLK	I	APB clock
PWRITE	I	APB write, 0=Read and 1=Write.
PWDATA[31:0]	I	APB write data bus
PADDR[1:0]	I	APB Address bus
PSEL	I	APB Block select
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	APB read data bus

Interrupt Interface		
blkInt	O	Block Interrupt (1=interrupt active)

4.4 Register Descriptions

Table 4-3: Timer Registers Summary

Register Name	Offset	Description
Load	0x0	Initial value of the timer
Value	0x4	The current value of the timer (Read Only)
Control	0x8	Provides enable/disable, mode and pre-scale configurations
Clear	0xC	Clears the interrupt (Write Only)

Table 4-4: Load (offset 0x00)

Bits	Access	Default	Description
[31:16]	Read/Write	undefined	Unused – undefined on a read, should be set to zero on a write
[15:0]	Read/Write	0x0	The initial value of the timer. Also, the reload value when the timer is in periodic mode.

Table 4-5: Value (offset 0x04)

Bits	Access	Default	Description
[31:16]	Read	undefined	Unused – undefined
[15:0]	Read	0x0	The current value of the timer

Table 4-6: Control (offset 0x08)

Bits	Access	Default	Description
[31:8]	Read/Write	undefined	Unused – undefined on a read, should be set to zero on a write
[7]	Read/Write	0x0	Enable – 0=Timer Disabled, 1=Timer Enabled
[6]	Read/Write	0x0	Mode – 0=Free running mode, 1=Periodic mode
[5]	Read/Write	0x0	Unused - 0 on a read, should be set to zero on a write
[4:2]	Read/Write	0x0	Prescale – 0=none – clock is divided by one 1=clock is divided by 16 2=clock is divided by 256 3=clock is divided by 2 4=clock is divided by 8 5=clock is divided by 32 6=clock is divided by 128 7=clock is divided by 1024
[1:0]	Read/Write	0x0	Unused - 0 on a read, should be set to zero on a write

Table 4-7: Clear (offset 0x0C)

Bits	Access	Default	Description
[31:0]	Write	undefined	Clear Interrupt – write any value to clear the timer interrupt

4.5 Functional Description

The timer is a basic down counter. When the timer is enabled by setting bit 7 in the Control Register or a new value is set in the Load Register, the Load Register is loaded into the Value Register. Bits 2 through 4 of the Control Register control the scaling of PCLK creating a timer clock. The Value Register is decremented on the leading edge of each timer clock until it reaches zero. The blkInt output is then activated until it is cleared by writing to the Clear Register.

The timer has two modes of operation determined by bit 6 in the Control Register: free running and periodic. When the Value Register reaches zero and the free running mode is selected, Value is decremented to 0xFFFF on the next timer clock edge. When the Value Register reaches zero and the periodic running mode is selected, the Load Register is reloaded into the Value Register on the next timer clock edge. In either case, the Value Register will continue to be decremented on each timer clock until the Value register reaches zero where the whole process starts over again.

Because zero is a separate count, periodic counts are actually one count longer than the contents of the Load Register. For example, if a value of 99 is written to the Load Register, no prescale is selected, and the periodic mode is selected, an interrupt will occur every 100 clock cycles.

The prescalers are free running counters. They are held in the clear state when the Timer is disabled (bit 6 of the Control Register is zero). They are also cleared when the Load Register is loaded. Bits 2 through 4 of the Control Register determine which prescale output is selected. This is used to produce a clock enable for the timer counter register. A bitwise OR of the prescale counter and a Mask value (based on three bits in the Control Register) is performed; if all bits of the OR output are high and the timer is enabled, then the timer counter is allowed to decrement.

So in periodic mode the number of clocks between interrupts will be:

# clocks = (load + 1) * pDiv,

where val is the contents of the Load register and pDiv is the prescale divisor value corresponding to the value programmed into the Control register.

Note that in periodic interrupt mode with small Load and prescale values, interrupts can be triggered faster than the software can service them. This should be avoided.

4.6 Timing

The timing of the Timer module's APB interface relies on the APB Master that is initiating the transaction. Usually, the transaction is initiated by an AHB to APB Bridge module, and passed through to the Timer module via an APB Channel module.

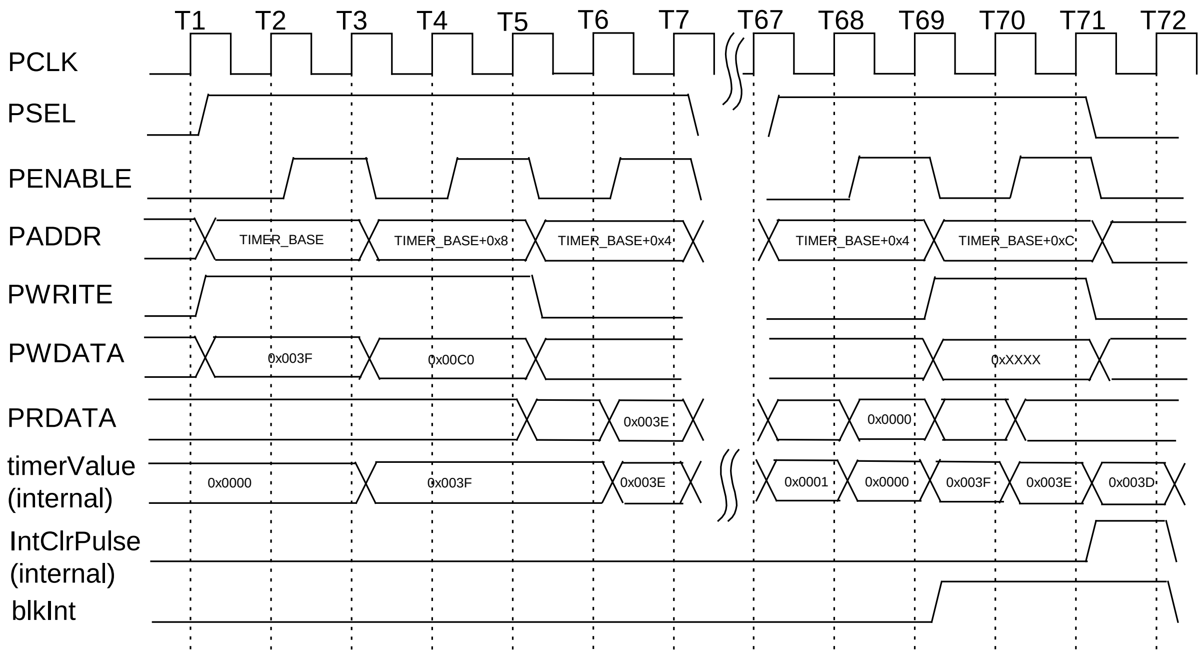


Figure 4-1: Interface and Interrupt Timing

Time T1:

The APB Master drives a new transfer on the APB bus at address TIMER_BASE. This transfer is a write to the Load Register to load the timer counter with a value of 63.

Time T2:

The APB Master drives PENABLE = 1'b1 to indicate to the slave that the APB data phase will occur at time T3.

The Timer module samples address TIMER_BASE along with PWRITE = 1’b1 and PSEL = 1’b1. PWDATA will be sampled at time T3.

Time T3:

The APB Master drives PENABLE = 1’b0. Also, the APB Master drives a new transfer on the APB bus at address TIMER_BASE. This transfer is a write to the Control Register to configure periodic mode and to enable the timer.

The Timer module samples PENABLE = 1’b1, and latches 0x3F from PWDATA. This write has the effect of changing the timer counter value immediately after the clock edge.

Time T4:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T5.

The Timer module samples TIMER_BASE+0x08 along with PWRITE = 1’b1 and PSEL = 1’b1. PWDATA will be sampled at time T5.

Time T5:

The APB Master drives PENABLE = 1’b0. Also, the APB Master drives a new transfer on the APB bus at address TIMER_BASE+0x04. This transfer is a read of the Value Register.

The Timer Module samples PENABLE = 1’b1, and latches 0xC0 from PWDATA. This write has the effect of enabling the timer and enabling periodic mode.

Time T6:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T7.

The Timer module samples TIMER_BASE+0x04 along with PWRITE = 1’b0 and PSEL = 1’b1. As a result, the Timer module drives PRDATA = 0x3E (reflecting the state of the Value Register) back toward the APB Master.

The Timer’s timerValue register begins to decrement.

Time T7:

The APB Master drives PENABLE = 1’b0. Since the current transaction is a read, the APB Master samples PRDATA = 0x3D at this time.

The APB Slave does nothing of consequence on this cycle.

The Timer’s timerValue register continues to decrement.

Many clock cycles go by with the timerValue register decrementing toward zero.

Time T67:

The APB Master drives a new transfer on the APB bus at address TIMER_BASE+0x04. This transfer is a read of the Value Register.

The APB Slave does nothing of consequence on this cycle.

The Timer’s timerValue register decrements to 0x01.

Time T68:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T69.

The Timer module samples TIMER_BASE+0x04 along with PWRITE = 1’b0 and PSEL = 1’b1. As a result, the Timer module drives PRDATA = 0x00 (reflecting the state of the Value Register) back toward the APB Master.

The Timer’s timerValue register decrements to 0x00.

Time T69:

The APB Master drives PENABLE = 1’b0. Also, the APB Master drives a new transfer on the APB bus at address TIMER_BASE+0x0C. This transfer is a write to the Clear Register.

The APB Slave does nothing of consequence on this cycle.

Since we are in periodic mode, the Timer’s timerValue register being zero causes the interrupt output, blkInt, to assert. The Timer’s timerValue register is reloaded with the Load value (0x3F).

Time T70:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T71.

The Timer module samples address TIMER_BASE+0xC along with PWRITE = 1’b1 and PSEL = 1’b1. PWDATA will be sampled at time T71.

The Timer continues decrementing its timerValue register.

Time T71:

The APB Master drives PENABLE = 1’b0.

The Timer module samples PENABLE = 1’b1, and latches 0xXXXX (any value) from PWDATA. This write has the effect of causing an internal one-cycle interrupt clear pulse to be 1’b1.

The Timer continues decrementing its timerValue register.

Timer T72:

The one-cycle interrupt clear pulse is sampled as 1’b1, and the Timer drives blkInt = 1’b0 as a result. The one-cycle interrupt clear pulse is driven to 1’b0.

5. Processor WatchDog Timer

5.1 Overview

The Watchdog Timer module is a programmable reset controller module. The Watchdog Timer module is a standard APB Slave peripheral; the Watchdog Timer registers are accessed through this interface. The Watchdog Timer implements a 16-bit down counter with a selectable prescaler in order to produce a delayed reset signal and a watchdog reset signal, as well as a warning interrupt.

The Watchdog Timer is a 16-bit peripheral but it is assumed that the addressing will be adjusted so that each register is on a 32-bit boundary.

5.2 Block Diagram

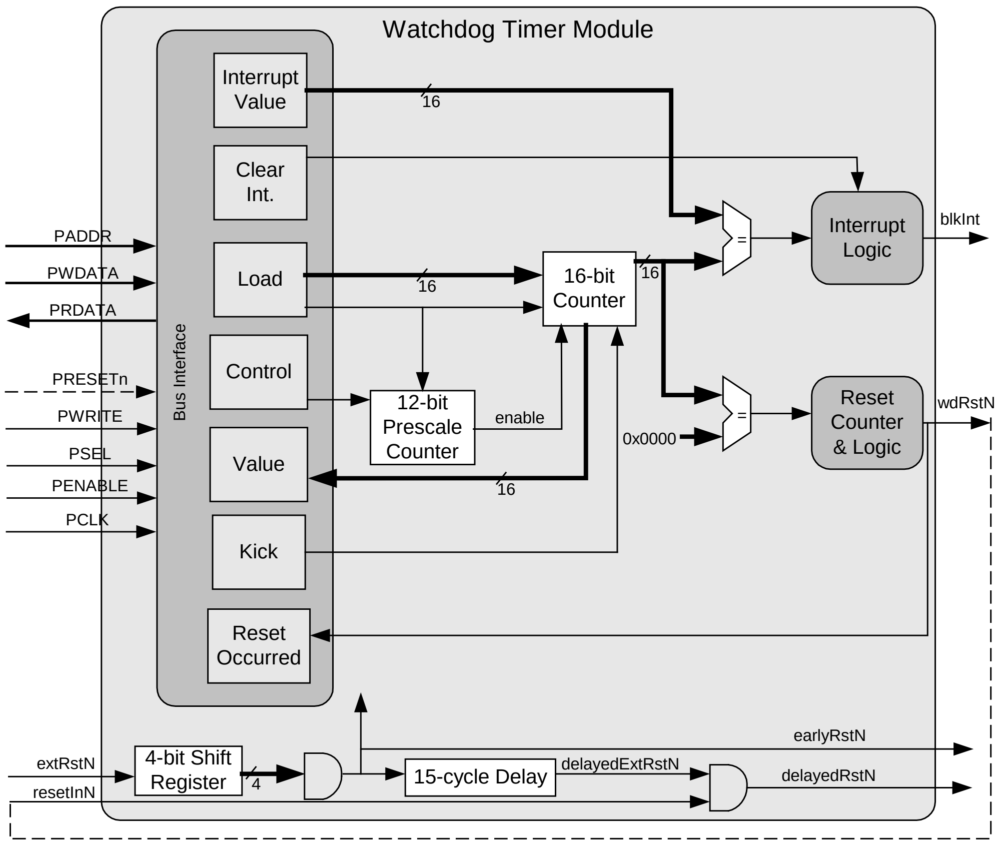


Figure 5-1: WatchDog Timer Block Diagram

5.3 Signal Descriptions

Table 5-1: WatchDog Timer Interface Singals

Signal	I/O	Description
APB Slave Interface		
PRESETn	I	Active low APB reset (unused in this module)
PCLK	I	APB clock
PWRITE	I	APB Write (0=read 1=write)
PWDATA[31:0]	I	Write Data from the APB
PADDR[31:0]	I	APB Address bus
PSEL	I	APB Block select
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	Read data to the APB
Interrupt Interface		

blkInt	O	Active high interrupt output
Reset Interface		
extRstN	I	The external reset signal, typically from a bush-button
resetInN	I	Usually this is driven by the wdRstN output
earlyRstN	O	A synchronized, de-glitched version of extRstN; note that this is NOT asserted by wdRstN, only by extRstN
delayedRstN	O	A delayed version of earlyRstN ANDed with resetInN; this is the system reset of a typical system
wdRstN	O	The watchdog reset signal; typically wired to resetInN external to the watchdog module

Table 5-2: WatchDog Timer Configurable Parameters

Configurable Parameter	Default	Description
`WATCHDOG_SECURITY	Not Defined	This can be defined in socPlatformDefs.vh in order to have the registers work in a more secure fashion. If defined, then additional security measures become available that make spurious resets less likely.

5.4 Programming Interface

5.4.1 Register Descriptions

The Watchdog Timer module contains several registers. These registers are outlined below.

A Verilog definition may be instantiated in order to have the registers work in a more secure fashion. If the following code is instantiated in code or in a compile script, then additional security measures become available that make spurious reconfigurations less likely:

```
`define WATCHDOG_SECURITY
```

The effects of this define are explained in the various register description sections.

Table 5-3: Register Summary

Register Name	Offset	Description
Load	0x00	Initial value of the timer
Value	0x04	The current value of the timer (Read Only)
Control	0x08	Provides enable/disable, mode and pre-scale configurations
Kick	0x0C	Provides a mechanism to easily reload the 16-bit counter
Reset Occurred	0x10	Provides status of whether a watchdog reset event occurred
Clear	0x14	Clears the interrupt (Write Only)
Interrupt Value	0x18	Value of counter to cause interrupt

The Load Register is a R/W register unless WATCHDOG_SECURITY is a define in the Verilog code and the lockOut bit in the Control Register has been set to 1’b1. If lockOut = 1’b1, then the register is read-only.

Table 5-4: Load (offset: 0x0)

Bits	Access	Default	Description
[31:16]	Read/Write	0x0	Reserved. Read as 0x0.
[15:0]	Read/Write	0x0	This 16-bit value gets loaded into the timer’s counter register and reloaded upon a specific write to the watchdog Kick Register.

Table 5-5: Value (offset: 0x4)

Bits	Access	Default	Description
[31:16]	Read	0x0	Reserved. Read as 0x0.
[15:0]	Read	0xFFFF	The current value of the watchdog timer.

The Control Register is a R/W register unless WATCHDOG_SECURITY is a define in the Verilog code and the lockOut bit has been set to 1'b1 via a previous write to the Control Register. If lockOut = 1'b1, then the Control Register is read-only.

Table 5-6: Control (offset: 0x8)

Bits	Access	Default	Description
[31:8]	Read/Write	0x0	Reserved. Read as 0x0.
[7]	Read/Write	0x0	Enable – 0=Timer Disabled, 1=Timer Enabled
[6:5]	Read/Write	0x0	Reserved. Read as 0x0.
[4:2]	Read/Write	0x0	Prescale – 0=none – clock is divided by one 1=clock is divided by 16 2=clock is divided by 256 3=clock is divided by 32 4=clock is divided by 128 5=clock is divided by 1024 6=clock is divided by 4096 7=clock is divided by 4096
[1]	Read/Write	0x0	Reserved. Read as 0x0.
[0]	Read/Write	0x0	lockOut - If WATCHDOG_SECURITY is a `define in the Verilog code, this bit is the lockOut bit. Write 1'b1 to prevent further configuration changes to the Load, Control, or Reset Occurred Registers. Otherwise, this bit is reserved and writes will have no effect.

The Kick Register is a write-only register. If WATCHDOG_SECURITY is defined in the Verilog code, then a specific value must be written to the Kick Register in order to reload the watchdog timer. If WATCHDOG_SECURITY is not defined in the Verilog code, then a write of any value to this register will reload the watchdog timer.

Table 5-7: Kick (offset: 0xC)

Bits	Access	Default	Description
[31:0]	Write	undefined	Kick the watchdog - reset the timer value to its default load and clear the block interrupt. WATCHDOG_SECURITY is not defined: Write any value to reload the watchdog timer. WATCHDOG_SECURITY is defined: Write 0xA5A5 to reload the watchdog timer.

The Reset Occurred Register is a R/W register unless WATCHDOG_SECURITY is a define in the Verilog code and the lockOut bit in the Control Register has been set to 1'b1. If lockOut = 1'b1, then the Reset Occurred Register is read-only and writes have no effect.

Table 5-8: Reset Occurred (offset: 0x10)

Bits	Access	Default	Description
[31:1]	Read/Write	0x0	Reserved. Read as 0x0.
[0]	Read/Write	0x0	resetOccurred status bit **Writes:** If the lockOut bit in the Control Register is 1'b1, writes have no effect. If the lockOut bit is 1'b0, then write any value to the Reset Occurred register to clear the resetOccurred status bit. **Reads:** Read the resetOccurred status to see if a watchdog reset event has occurred. 1'b1 = watchdog reset event has occurred. 1'b0 = watchdog reset event has not occurred.

The Clear Register is a write-only register that clears the active interrupt signal (blkInt). This register clears the interrupt regardless of the state of the lockOut bit.

Table 5-9: Clear (offset 0x14)

Bits	Access	Default	Description
[31:0]	Write	undefined	Write any value to clear blkInt interrupt.

The Interrupt Value Register is a R/W register unless WATCHDOG_SECURITY is defined in the Verilog code and the lockOut bit in the Control Register has been set to 1’b1. If lockOut = 1’b1, then the register is read-only.

Table 5-10: Interrupt Value (0x18)

Bits	Access	Default	Description
[31:16]	Read/Write	0x0	Reserved. Read as 0x0.
[15:0]	Read/Write	0x0	This 16-bit value is compared with the timer’s counter register. If they are equal an interrupt is generated.

5.5 Functional Description

The watchdog timer is a basic down counter which generates a reset when it decrements to zero. The watchdog timer is only active when enabled, and can be enabled by setting bit 7 in the Control Register.

Bits 2 thru 4 of the Control Register control the prescaling of PCLK creating a timer clock. For example, when all 3 bits are zero, PCLK is used directly to generate the timer clock, or when bit 2 is asserted and bit 3 is unasserted, PCLK is divided by 16 to generate the timer clock.

The prescaler is a free running counter. Its output is a 1-clock-wide pulse that feeds the watchdog timer’s main 16-bit counter as a clock enable. The prescaler is held in the clear state when the Timer is disabled (bit 6 of the Control Register is zero). It is also cleared when the Load Register is loaded. Bits 2 thru 4 of the Control Register determine which prescale divide ratio is selected. A bitwise OR of the prescale counter and a Mask value (based on three bits in the Control Register) is performed; if all bits of the OR output are high and the timer is enabled, then the 16-bit watchdog timer counter is allowed to decrement.

When the watchdog timer is active, the Value Register counts down from the value of the Load Register. The Value Register is decremented on each enabled clock until it reaches zero. The module then asserts wdRstN, which can be used to restart a system that appears to have timed out. The system should intermittently “Kick” the Watchdog, to prevent this system reset. When kicked the watchdog reloads the Value register with the contents of the Load Register and resumes decrementing.

resetClks = (loadVal+1)*prescaleDiv + 1

The watchdog can be kicked by a single write to the Kick Register. A timer kick, a block enable, or a write to the Load Register all cause the Value Register to be loaded with the contents of the Load Register.

The watchdog timer will issue a warning interrupt (assert blkInt = 1’b1) intValue number of timer counts before it asserts wdRstN. The register intValue can be programmed with a 16-bit value. The purpose of the interrupt is to warn the system that it hasn’t kicked the watchdog recently and should do so or the system will be reset. blkInt will remain active until the watchdog has been kicked or until the Clear Register has been written to. This feature also allows the watchdog timer to be used as an additional timer to generate and interrupt if the watchdog reset is not needed.

intClks = (loadVal-intVal+1)*prescaleDiv + 1

The watchdog timer also provides reset controller functions. The module feeds extRstN into a 4-bit linear shift register and asserts earlyRstN when all 4 bits of the shift register are zero. This makes earlyRstN a glitch free version of extRstN. The internal signal delayedExtRstN is the earlyRstN signal with its assertion delayed by 15 PCLK cycles.

The delayedRstN output is simply a logical AND of resetInN and delayedExtRstN. It can be used to reset the processor after other devices, such as flash, have been reset. The watchdog reset will stay asserted for 256 clocks and will then de-assert.

Note that the PRESETn signal is present but unused. This is because in most systems, PRESETn will be tied to wdRstN via the delayedRstN signal. In order to work as specified, the watchdog timer cannot use the generated reset to reset its own registers.

5.6 Timing

5.6.1 Interface and Interrupt Timing

The timing of the Timer module's APB interface relies on the APB Master that is initiating the transaction. Usually, the transaction is initiated by an AHB to APB Bridge module, and passed through to the Timer module via an APB Channel module.

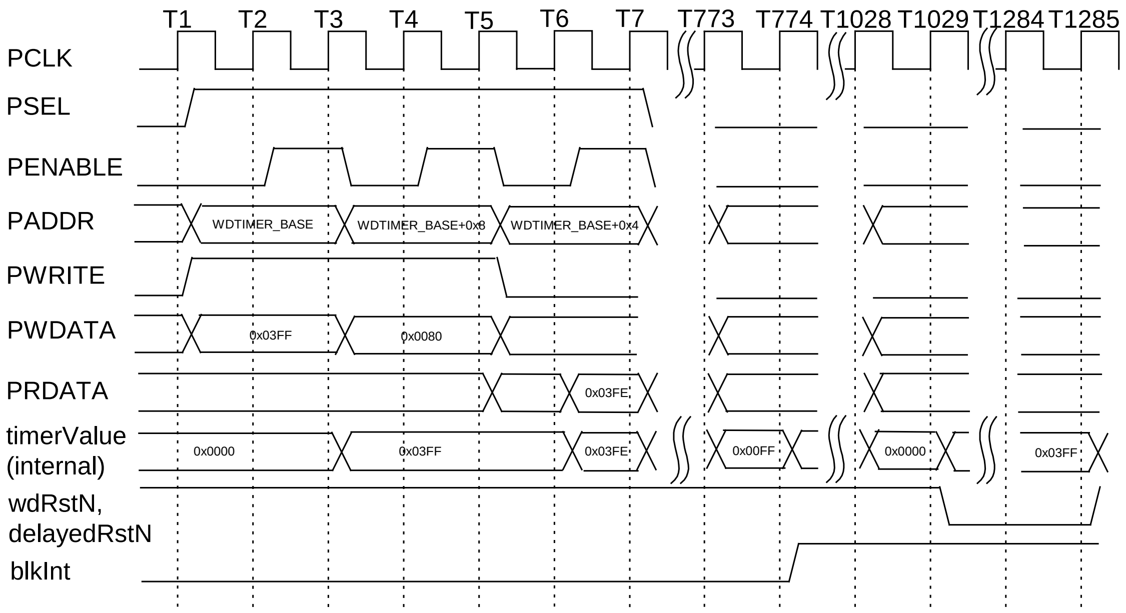


Figure 5-2: Interface and Interrupt Timing

Time T1:

The APB Master drives a new transfer on the APB bus at address WDTIMER_BASE. This transfer is a write to the Load Register to load the counter with a value of 1023.

Time T2:

The APB Master drives PENABLE = 1'b1 to indicate to the slave that the APB data phase will occur at time T3.

The Watchdog Timer module samples address WDTIMER_BASE along with PWRITE = 1'b1 and PSEL = 1'b1. PWDATA will be sampled at time T3.

Time T3:

The APB Master drives PENABLE = 1'b0. Also, the APB Master drives a new transfer on the APB bus at address WDTIMER_BASE. This transfer is a write to the Control Register to enable the timer.

The Watchdog Timer module samples PENABLE = 1'b1, and registers 0x03FF from PWDATA. This write has the effect of changing the timer counter value immediately after the clock edge.

Time T4:

The APB Master drives PENABLE = 1'b1 to indicate to the slave that the APB data phase will occur at time T5.

The Watchdog Timer module samples WDTIMER_BASE+0x08 along with PWRITE = 1'b1 and PSEL = 1'b1. PWDATA will be sampled at time T5.

Time T5:

The APB Master drives PENABLE = 1'b0. Also, the APB Master drives a new transfer on the APB bus at address WDTIMER_BASE+0x04. This transfer is a read of the Value Register.

The Watchdog Timer Module samples PENABLE = 1'b1, and registers 0x80 from PWDATA. This write has the effect of enabling the watchdog timer counter.

Time T6:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T7.

The Watchdog Timer module samples WDTIMER_BASE+0x04 along with PWRITE = 1’b0 and PSEL = 1’b1. As a result, the Timer module drives PRDATA, reflecting the state of the Value Register, back toward the APB Master.

The Timer’s timerValue register begins to decrement.

Time T7:

The APB Master drives PENABLE = 1’b0. Since the current transaction is a read, the APB Master samples PRDATA = 0x03FE at this time.

The APB Slave does nothing of consequence on this cycle.

The Timer’s timerValue register continues to decrement.

Many clock cycles go by with the timerValue register decrementing toward zero.

Time T773:

The Watchdog Timer’s timerValue register decrements to 0x00FF.

Time T774:

The Watchdog Timer’s timerValue register is sampled to be 0x00FF. This causes blkInt to transition to 1’b1.

More clock cycles go by with the timerValue register decrementing toward zero.

Time T1028:

The Watchdog Timer’s timerValue register decrements to 0x0000.

Time T1029:

The Watchdog Timer’s timerValue register is sampled to be 0x0000. This causes wdRstN (and delayedRstN) to transition to 1’b0. The watchdog reset will be active for 256 cycles of PCLK. (Not shown: the assertion of wdRstN causes the 16-bit counter to be reloaded to 0x03FFF at time T1030).

Many clock cycles go by with wdRstN = 1’b0.

Time T1284:

The wdRstN signal has been low for 255 cycles.

Time T1285:

The wdRstN signal has been low for 256 cycles. This causes wdRstN to transition back to 1’b1, thus releasing the system reset. Note that blkInt will remain high until software explicitly clears it by either writing to the Kick Register or by writing to the Clear Register.

5.6.2 Reset Timing

This section documents the timing of the reset controller portion of the Watchdog Timer module.

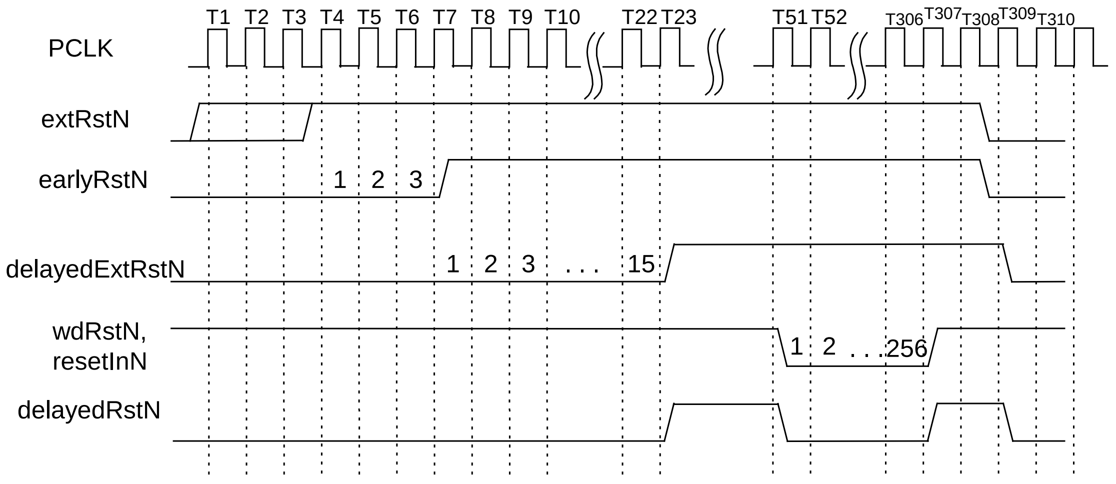


Figure 5-3: Reset Timing

Time T1-T6:

The asynchronous external reset input extRstN is asserted then unstable until midway through clock cycle T3, when it deasserts. Each clock, extRstN is being clocked into a 4-bit shift register.

Time T7:

When all shift register bits are high, the earlyRstN output signal is deasserted. This takes at least 3 full clock cycles. The delayCnt counter starts counting.

Time T8-T22:

The delayCnt counter continues counting.

Time T23:

On the 15th count, the internal signal delayedExtRstN is deasserted. This causes the delayedRstN output to deassert. This output is commonly used for a system reset (for HRESETn & PRESETn).

Time T51:

The watchdog timer has timed out and caused the wdRstN output to be asserted for 256 clock cycles. This output is externally routed back to the resetInN input.

Time T307:

The wdRstN output deasserts after 256 clock cycles.

Time T308:

Asynchronous to the clock, the extRstN has been reasserted. This causes earlyRstN to assert immediately.

Time T309:

On the next rising clock edge, the delayedExtRstN and delayedRstN signals will assert. Note that the wdRstN output does not change. This completes the cycle, which can start over again at T1.

6. Real Time Clock (RTC)

6.1 Overview

The SoC-RTC is a clock-calendar IP core that keeps track of the "Time of Day". The core is organized as a series of BCD counters that counts Seconds, Minutes, Hours, Days, Months and Years (Time Units).

The RTC seconds counter time base is generated from a programmable divisor of the system clock. The divisor should be set to generate a 1Hz clock enable signal to the seconds counter. This divisor can also be used to adjust the clock accuracy by periodically adding or subtracting values from the divisor. Calibration accuracy is controlled by software.

Each of the Time Units can be loaded by writing BCD information to the proper Time Unit register. The RTC will immediately load the new time and will continue to count.

The Core can also be loaded with an Alarm Compare value for each Time Unit that will generate an interrupt when that Time Unit reaches the compare value. A Repeat Alarm function is also available. When enabled, this function will cause an interrupt at the terminal count of each Time Unit counter. For example, if the Repeat Alarm bit (bit 7 or Seconds Alarm Register) for the Seconds counter is set, the RTC engine will generate an interrupt every second when the counter rolls over from 59 to 00.

Testmode bits for each Time Unit counter enable the counter to be clocked by the secondsClk in order to speed testability. This can also be used to incrementally update the RTC.

The testRstN bit enables the RTC counters to be reset under software control. Otherwise the RTC counters are free running and do not reset upon system reset.

The SoC-RTC operates as a slave device on the AMBA APB bus. Each register is accessed by simple APB bus transactions.

The SoC-RTC package includes Verilog source and simulation test-benches.

6.2 Block Diagram

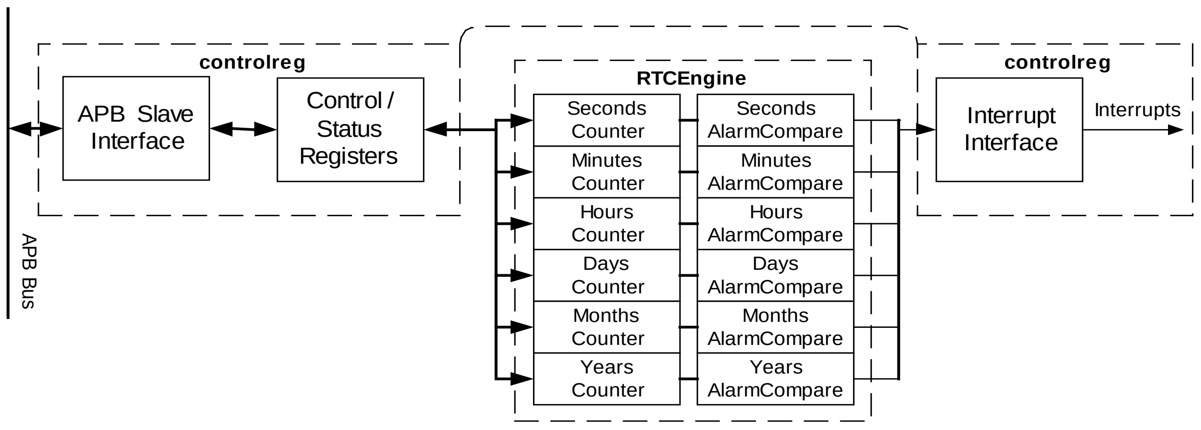


Figure 6-1: RTC Block Diagram

6.3 Register Descriptions

Note: seconds, minutes, etc. are BCD.

Bits [7:4] represent 10s (10-99)

Bits [3:0] represent 1s (0-9)

Table 6-1: RTC Registers

Register	Bits	Offset Address	R/W	default	Bit Description
Seconds Ctr		**0x00**	**R/W**	**0x00**	
Seconds 00-50	6:4				Read actual seconds BCD value, write to update new seconds BCD value.
Seconds 0-9	3:0				
					Counts from 00 to 59
Minutes Ctr		**0x04**	**R/W**	**0x00**	
Minutes 00-50	6:4				Read actual minutes BCD value, write to update new minutes BCD value.
Minutes 0-9	3:0				
					Counts from 00 to 59
Hours Ctr		**0x08**	**R/W**	**0x00**	
Hours 00-20	5:4				Read actual hours BCD value, write to update new hours BCD value.
Hours 0-9	3:0				
					Counts from 00 to 23
Days Ctr		**0x0C**	**R/W**	**0x01**	
Days 00-30	5:4				Read actual days BCD value, write to update new days BCD value.
Days 0-9	3:0				
					Counts from 01 to 28,29,30 or 31
Months Ctr		**0x10**	**R/W**	**0x01**	
Months 00-20	4				Read actual months BCD value, write to update new months BCD value.
Months 0-9	3:0				
					Counts from 01 to 12
Years Ctr		**0x14**	**R/W**	**0x00**	
Years 00-20	7:4				Read actual years BCD value, write to update new years BCD value.
Years 0-9	3:0				
					Counts from 00 to 99
RTC ctrl		0x18	R/W	00	
	5:0				testmode – Used to speed up Time Unit counters for test.
	7				testRstN – Used to reset the Time Unit counters under software control.
Clk Divisor		**0x1C**	**R/W**		
	31:0			0x01	Clock Divisor for generating a 1Hz enable to the Seconds Counter
Seconds Alarm		**0x20**	**R/W**	**0x00**	
Repeat	7				When set, generates an interrupt every second.
Seconds 00-50	6:4				BCD Alarm Compare value used to generate an interrupt on a seconds counter match.
Seconds 0-9	3:0				
Minutes Alarm		**0x24**	**R/W**	**0x00**	
Repeat	7				When set, generates an interrupt every minute.
Minutes 00-50	6:4				BCD Alarm Compare value used to generate an interrupt on a

Register	Bits	Offset Address	R/W	default	Bit Description
Seconds Ctr		**0x00**	**R/W**	**0x00**	
Seconds 00-50	6:4				Read actual seconds BCD value, write to update new seconds BCD value.
Seconds 0-9	3:0				
					Counts from 00 to 59
Minutes Ctr		**0x04**	**R/W**	**0x00**	
Minutes 00-50	6:4				Read actual minutes BCD value, write to update new minutes BCD value.
Minutes 0-9	3:0				
					Counts from 00 to 59
Hours Ctr		**0x08**	**R/W**	**0x00**	
Hours 00-20	5:4				Read actual hours BCD value, write to update new hours BCD value.
Hours 0-9	3:0				
					Counts from 00 to 23
Days Ctr		**0x0C**	**R/W**	**0x01**	
Days 00-30	5:4				Read actual days BCD value, write to update new days BCD value.
Days 0-9	3:0				
					Counts from 01 to 28,29,30 or 31
Months Ctr		**0x10**	**R/W**	**0x01**	
Months 00-20	4				Read actual months BCD value, write to update new months BCD value.
Months 0-9	3:0				
					Counts from 01 to 12
Years Ctr		**0x14**	**R/W**	**0x00**	
Years 00-20	7:4				Read actual years BCD value, write to update new years BCD value.
Years 0-9	3:0				
					Counts from 00 to 99
RTC ctrl		0x18	R/W	00	
	5:0				testmode – Used to speed up Time Unit counters for test.
	7				testRstN – Used to reset the Time Unit counters under software control.
Clk Divisor		**0x1C**	**R/W**		
	31:0			0x01	Clock Divisor for generating a 1Hz enable to the Seconds Counter
					minutes counter match.
Minutes 0-9	3:0				
Hours Alarm		**0x28**	**R/W**	**0x00**	
Repeat	7				When set, generates an interrupt every hour.
Hours 00-20	5:4				BCD Alarm Compare value used to generate an interrupt on a hours counter match.
Hours 0-9	3:0				
Days Alarm		**0x2C**	**R/W**	**0x00**	
Repeat	7				When set, generates an interrupt every day.
Days 00-30	5:4				BCD Alarm Compare value used to generate an interrupt on a

Register	Bits	Offset Address	R/W	default	Bit Description
Seconds Ctr		**0x00**	**R/W**	**0x00**	
Seconds 00-50	6:4				Read actual seconds BCD value, write to update new seconds BCD value.
Seconds 0-9	3:0				
					Counts from 00 to 59
Minutes Ctr		**0x04**	**R/W**	**0x00**	
Minutes 00-50	6:4				Read actual minutes BCD value, write to update new minutes BCD value.
Minutes 0-9	3:0				
					Counts from 00 to 59
Hours Ctr		**0x08**	**R/W**	**0x00**	
Hours 00-20	5:4				Read actual hours BCD value, write to update new hours BCD value.
Hours 0-9	3:0				
					Counts from 00 to 23
Days Ctr		**0x0C**	**R/W**	**0x01**	
Days 00-30	5:4				Read actual days BCD value, write to update new days BCD value.
Days 0-9	3:0				
					Counts from 01 to 28,29,30 or 31
Months Ctr		**0x10**	**R/W**	**0x01**	
Months 00-20	4				Read actual months BCD value, write to update new months BCD value.
Months 0-9	3:0				
					Counts from 01 to 12
Years Ctr		**0x14**	**R/W**	**0x00**	
Years 00-20	7:4				Read actual years BCD value, write to update new years BCD value.
Years 0-9	3:0				
					Counts from 00 to 99
RTC ctrl		0x18	R/W	00	
	5:0				testmode – Used to speed up Time Unit counters for test.
	7				testRstN – Used to reset the Time Unit counters under software control.
Clk Divisor		**0x1C**	**R/W**		
	31:0			0x01	Clock Divisor for generating a 1Hz enable to the Seconds Counter

					days counter match.
Days 0-9	3:0				
Months Alarm		**0x30**	**R/W**	**0x00**	
Repeat	7				When set, generates an interrupt every month.
Months 00-20	4				BCD Alarm Compare value used to generate an interrupt on a months counter match.
Months 0-9	3:0				
Years Alarm		**0x34**	**R/W**	**0x00**	
Years 00-20	7:4				BCD Alarm Compare value used to generate an interrupt on a years counter match.

Register	Bits	Offset Address	R/W	default	Bit Description
Seconds Ctr		**0x00**	**R/W**	**0x00**	
Seconds 00-50	6:4				Read actual seconds BCD value, write to update new seconds BCD value.
Seconds 0-9	3:0				
					Counts from 00 to 59
Minutes Ctr		**0x04**	**R/W**	**0x00**	
Minutes 00-50	6:4				Read actual minutes BCD value, write to update new minutes BCD value.
Minutes 0-9	3:0				
					Counts from 00 to 59
Hours Ctr		**0x08**	**R/W**	**0x00**	
Hours 00-20	5:4				Read actual hours BCD value, write to update new hours BCD value.
Hours 0-9	3:0				
					Counts from 00 to 23
Days Ctr		**0x0C**	**R/W**	**0x01**	
Days 00-30	5:4				Read actual days BCD value, write to update new days BCD value.
Days 0-9	3:0				
					Counts from 01 to 28,29,30 or 31
Months Ctr		**0x10**	**R/W**	**0x01**	
Months 00-20	4				Read actual months BCD value, write to update new months BCD value.
Months 0-9	3:0				
					Counts from 01 to 12
Years Ctr		**0x14**	**R/W**	**0x00**	
Years 00-20	7:4				Read actual years BCD value, write to update new years BCD value.
Years 0-9	3:0				
					Counts from 00 to 99
RTC ctrl		0x18	R/W	00	
	5:0				testmode – Used to speed up Time Unit counters for test.
	7				testRstN – Used to reset the Time Unit counters under software control.
Clk Divisor		**0x1C**	**R/W**		
	31:0			0x01	Clock Divisor for generating a 1Hz enable to the Seconds Counter
Years 0-9	3:0				
INT control		**0x38**	**R/W**	**0x00**	
	5				Enable Year Alarm interrupt.
	4				Enable Month Alarm interrupt.
	3				Enable Day Alarm interrupt.
	2				Enable Hour Alarm interrupt.
	1				Enable Minute Alarm interrupt.
	0				Enable Second Alarm interrupt.
INT status		**0x3C**	**RO**	**0x00**	
	5				Year Alarm interrupt status

Register	Bits	Offset Address	R/W	default	Bit Description
Seconds Ctr		**0x00**	**R/W**	**0x00**	
Seconds 00-50	6:4				Read actual seconds BCD value, write to update new seconds BCD value.
Seconds 0-9	3:0				
					Counts from 00 to 59
Minutes Ctr		**0x04**	**R/W**	**0x00**	
Minutes 00-50	6:4				Read actual minutes BCD value, write to update new minutes BCD value.
Minutes 0-9	3:0				
					Counts from 00 to 59
Hours Ctr		**0x08**	**R/W**	**0x00**	
Hours 00-20	5:4				Read actual hours BCD value, write to update new hours BCD value.
Hours 0-9	3:0				
					Counts from 00 to 23
Days Ctr		**0x0C**	**R/W**	**0x01**	
Days 00-30	5:4				Read actual days BCD value, write to update new days BCD value.
Days 0-9	3:0				
					Counts from 01 to 28,29,30 or 31
Months Ctr		**0x10**	**R/W**	**0x01**	
Months 00-20	4				Read actual months BCD value, write to update new months BCD value.
Months 0-9	3:0				
					Counts from 01 to 12
Years Ctr		**0x14**	**R/W**	**0x00**	
Years 00-20	7:4				Read actual years BCD value, write to update new years BCD value.
Years 0-9	3:0				
					Counts from 00 to 99
RTC ctrl		0x18	R/W	00	
	5:0				testmode – Used to speed up Time Unit counters for test.
	7				testRstN – Used to reset the Time Unit counters under software control.
Clk Divisor		**0x1C**	**R/W**		
	31:0			0x01	Clock Divisor for generating a 1Hz enable to the Seconds Counter
	4				Month Alarm interrupt status
	3				Day Alarm interrupt status
	2				Hour Alarm interrupt status
	1				Minute Alarm interrupt status
	0				Second Alarm interrupt status
INT clear		0x40	WO	0x00	LCD Control Register
	5				Clear Year Alarm interrupt.
	4				Clear Month Alarm interrupt.
	3				Clear Day Alarm interrupt.
	2				Clear Hour Alarm interrupt.

Register	Bits	Offset Address	R/W	default	Bit Description
Seconds Ctr		**0x00**	**R/W**	**0x00**	
Seconds 00-50	6:4				Read actual seconds BCD value, write to update new seconds BCD value.
Seconds 0-9	3:0				
					Counts from 00 to 59
Minutes Ctr		**0x04**	**R/W**	**0x00**	
Minutes 00-50	6:4				Read actual minutes BCD value, write to update new minutes BCD value.
Minutes 0-9	3:0				
					Counts from 00 to 59
Hours Ctr		**0x08**	**R/W**	**0x00**	
Hours 00-20	5:4				Read actual hours BCD value, write to update new hours BCD value.
Hours 0-9	3:0				
					Counts from 00 to 23
Days Ctr		**0x0C**	**R/W**	**0x01**	
Days 00-30	5:4				Read actual days BCD value, write to update new days BCD value.
Days 0-9	3:0				
					Counts from 01 to 28,29,30 or 31
Months Ctr		**0x10**	**R/W**	**0x01**	
Months 00-20	4				Read actual months BCD value, write to update new months BCD value.
Months 0-9	3:0				
					Counts from 01 to 12
Years Ctr		**0x14**	**R/W**	**0x00**	
Years 00-20	7:4				Read actual years BCD value, write to update new years BCD value.
Years 0-9	3:0				
					Counts from 00 to 99
RTC ctrl		0x18	R/W	00	
	5:0				testmode – Used to speed up Time Unit counters for test.
	7				testRstN – Used to reset the Time Unit counters under software control.
Clk Divisor		**0x1C**	**R/W**		
	31:0			0x01	Clock Divisor for generating a 1Hz enable to the Seconds Counter
	1				Clear Minute Alarm interrupt.
	0				Clear Second Alarm interrupt.

7. I2C Master

7.1 Overview

This is an implementation of a standard I2C Master. The I2C peripheral contains the following main sections:

Configuration Registers

Clock Divider/Clock Select

Command FIFO and Read Data FIFO

I2C Transmit and Receive Engine

Interrupt Generation Logic

Configuration registers are written and read by the processor via an APB Interface. The clock divider/clock select module is used to customize the frequency of the I2C portion of the module. Two separate FIFOs are used – one for storing up to 32 commands from the APB Interface, the other for storing up to 16 bytes of read data from the I2C Bus. The transmit engine reads commands from the command FIFO and implements these as I2C instructions. The receive engine monitors the I2C bus for slave responses, and stores data in a Read Data FIFO, the contents of which are available to the processor on the APB Interface. Various conditions can cause an interrupt to be generated.

7.2 Block Diagram

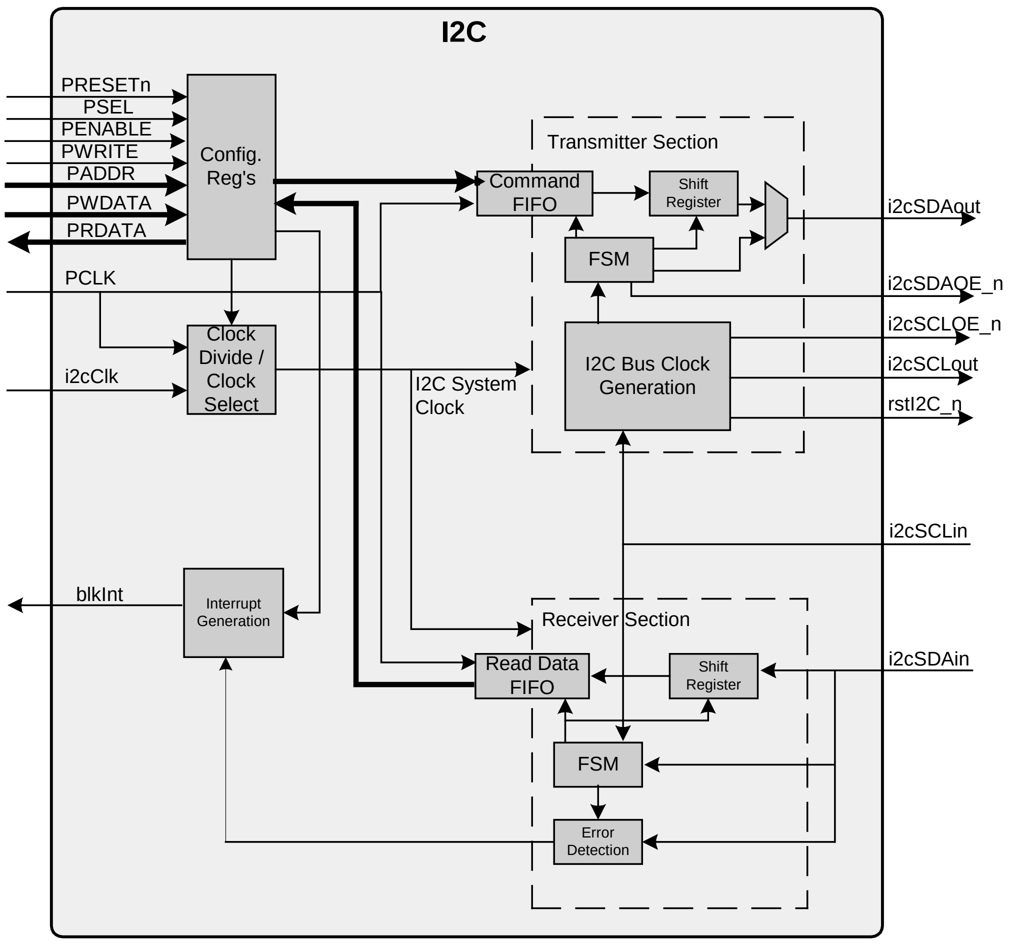


Figure 7-1: I2C Block Diagram

7.3 Signal Descriptions

Table 7-1: I2C Interface Signals

Signal Name	I/O	Description
PRESETn	I	APB reset (active low)
PCLK	I	APB clock
PWRITE	I	APB write, 0=Read and 1=Write.
PWDATA[31:0]	I	APB write data bus
PADDR[2:0]	I	APB address bus, if the registers are expected to be on word boundaries, then PADDR[4:2] are connected to this input, otherwise the registers are located on byte boundaries and PADDR[2:0] is connected to this input.
PSEL	I	APB Block select; this signal must be set to access any of the UART's registers.
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	APB read data bus
blkInt	O	Block interrupt, this signal is set when any of the following interrupt sources become active: Command FIFO Empty, Read Data FIFO Not Empty, I2C Error.
clkI2C	I	External clock to drive I2C logic
rstI2C_n	O	Reset synchronized to I2C system clock
i2cSCLin	I	I2C Bus clock input
i2cSCLout	O	I2C Bus clock output
i2cSCLOE_n	O	Active-low output enable for bi-directional i2cSCL signal
i2cSDAin	I	I2C Bus Data input
i2cSDAout	O	I2C Bus Data output
i2cSDAOE_n	O	Active-low output enable for bi-directional i2cSDA signal

7.4 Programming Interface

7.4.1 Register Descriptions

Table 7-2: I2C Register Summary

Register Name	Offset	Access	Description
Status	0x00	Read	Operational Status of the I2C Module and its sub-modules. Reading the Status register clears the blkInt Interrupt signal.
Read Data	0x04	Read	This register provides access to the Read Data FIFO.
Command	0x08	Write	Programming interface to the I2C engine. See a later section for the list of commands and their effects.
Interrupt Enable	0x0C	R/W	This register enables (with 1's) or disables (with 0's) any of three individually configurable interrupt conditions. An interrupt to the processor (blkInt) can occur if any of these interrupt conditions are enabled AND if the I2C enable bit in the Control register is set.
Control	0x10	R/W	Operational Control of the I2C Module, including Enable, I2C Clock Select and I2C Clock Divider.
Prescale	0x14	R/W	Extension of the I2C Clock Divider value.

Table 7-3: Status Register (0ffset 0x00)

Bits	Access	Default	Description
[7:6]	Read	0x0	Unused
[5]	Read	0x0	Command FIFO Full
[4]	Read	0x0	Error - Command FIFO Overflow
[3]	Read	0x0	Error - Read Data FIFO Underflow
[2]	Read	0x0	Error - I2C Protocol (this signifies that an I2C "NACK" was detected when an "ACK" was expected.)

[1]	Read	0x0	Read Data FIFO Not Empty
[0]	Read	0x1	Command FIFO Empty

Table 7-4: Read Data Register (offset 0x04)

Bits	Access	Default	Description
[7:0]	Read	0x00	Data from I2C Slave to be read by processor

Table 7-5: Command Register (offset 0x08)

Bits	Access	Default	Description
[7:0]	Write	0x00	Commands from processor to be implemented as I2C events

Table 7-6: Interrupt Enable Register (offset 0x0C)

Bits	Access	Default	Description
[7:3]	Read/Write	0x00	Unused
[2]	Read/Write	0x0	Enable Error Interrupt
[1]	Read/Write	0x0	Enable Read Data FIFO Not Empty Interrupt
[0]	Read/Write	0x0	Enable Command FIFO Empty Interrupt

Table 7-7: Control Register (offset 0x10)

Bits	Access	Default	Description
[7]	Read/Write	0x0	I2C Enable
[6]	Read/Write	0x1	I2C Clock Source (1 = PCLK, 0 = clkI2C)
[5]	Read/Write	0x0	I2C APB Clock Divide Enable (1 = Enable, 0 = Disable)
[4:0]	Read/Write	0x00	I2C APB Clock Divider (Lower 5 bits)

Table 7-8: Prescale Register (offset 0x14)

Bits	Access	Default	Description
[7:0]	Read/Write	0x00	I2C APB Clock Divider (Upper 7 bits)

7.5 Functional Description

7.5.1 Clock Divider

The I2C Clock Select/Clock Divider module can be configured to generate a wide range of frequencies. The choices fall into three main categories:

Use the APB clock as the I2C System Clock. To do this, set the I2C Clock Source bit in the Control Register to 1.

Use the external clock (clkI2C) as the I2C System Clock. To do this, set the I2C Clock Source bit in the Control Register to 0.

Use a divided APB clock as the I2C System Clock. To do this, set the I2C Clock Source bit in the Control Register to 1, and set the divide value in the I2C Control and Prescale Registers to the desired amount.

The formula below gives the resulting I2C System Clock Frequency as a function of APB Clock Frequency and I2C Clock Divide Value (from the Control and Prescale Registers):

$$I2CSystemClockFrequency = \frac{APBClockFrequency}{(ClockDivideValue + 2)}$$

Note that the I2C System Clock Frequency is a factor of 4 faster than the nominal I2C Bus Clock Frequency.

$$I2CBusClockFrequency = \frac{I2CSystemClockFrequency}{4} = \frac{APBClockFrequency}{(ClockDivideValue + 2) \times 4}$$

The additional factor of 4 is necessary for the I2C Engine to generate all phases of the I2C Bus Clock, and to generate start and stop conditions on the I2C Bus that are synchronous to the I2C System Clock.

7.6 Command Interface

The Command register is the programming interface to the I2C Engine. The commands arrive at the I2C Engine via the Command FIFO, so the first valid command that is written to the Command register is the first I2C instruction implemented on the I2C bus. Because the command interface provides the basic building blocks for any I2C transaction, access to a wide range of I2C slave devices is supported.

If the Command FIFO is empty, up to 32 commands can be written to the Command Interface. It is the responsibility of the programmer to keep track of how many commands have been written to the FIFO in order to prevent a Command FIFO Overflow Error. Alternatively, the programmer may choose to poll the Status Register and check for Command FIFO Full before writing to the Command Register.

The following table shows the commands that can be used to program the I2C Engine to generate I2C events:

Table 7-9: I2C Commands

Command Name	Value	Tx/Rx	I2C Bus Function
I2C_CMD_NULL	0x00	N/A	This command is used at the end of an I2C command sequence. The I2C_CMD_NULL command has no effect on the I2C Bus.
I2C_CMD_WDAT0	0x10	Tx	This command is used to transmit a one-bit logic “low” condition on the I2C bus. Typically, this command is used to generate a Master Acknowledge (MACK) condition on the I2C bus.
I2C_CMD_WDAT1	0x11	Tx	This command is used to transmit a one-bit logic “high” condition on the I2C bus. Typically, this command is used to generate a Master Not Acknowledge (NACK) condition on the I2C bus.
I2C_CMD_WDAT8 <DATA>	0x12	Tx	The I2C_CMD_WDAT8 command sequence is used to transmit an 8-bit data value on the I2C bus. The I2C_CMD_WDAT8 command is the first command in a two-command sequence. The command directly following the I2C_CMD_WDAT8 is the 8-bit data value to be transmitted.
I2C_CMD_RDAT8	0x13	Rx	The I2C_CMD_RDAT8 command is used to alert the receiver to capture an 8-bit data value on the I2C Bus. It is expected that an I2C Slave will drive this value. Once the 8-bit data value has been captured, the I2C Engine places the value in the Read Data register, and sets the Read Data FIFO Not Empty flag in the Status register.
I2C_CMD_STOP	0x14	Tx	This command is used to generate an I2C “Stop” condition on the I2C bus.
I2C_CMD_STRT	0x15	Tx	This command is used to generate an I2C “Start” condition on the I2C bus.
I2C_CMD_VACK	0x16	Rx	This command is used to alert the receiver to verify a one-bit logic “low” condition, typically a Slave Acknowledge (ACK). If a logic “high” is detected on the I2C bus, the I2C Engine sets an error flag in the status register.

The I2C Protocol requires the receiving party (usually the Slave) to issue an Acknowledge after every 8-bit data transfer. The receiving party accomplishes this by driving the data line low for one I2C Clock cycle after receiving the 8-bit data value. The transmitting party will check for the Acknowledge by checking for a logic “low” on the I2C Bus for the clock cycle directly following the 8-bit data transfer. The I2C Engine can check for this Acknowledge with the I2C_CMD_VACK command, or produce the Acknowledge with the I2C_CMD_WDAT0 command.

7.6.1 Command Interface Examples

In the following examples, i2cDeviceAddress is the device address of the I2C Slave, i2cRegisterAddress is the internal register address of the I2C Slave, and i2cWriteData is the data to be written to the I2C Slave.

In these examples, note that the two-command sequence I2C_CMD_WDAT8 + <DATA> can be replaced by an eight-command sequence consisting of I2C_CMD_WDAT0 and I2C_CMD_WDAT1. For example, if

```
DATA = 8’b0010_0110,
```

the replacement sequence would be:

```
I2C_CMD_WDAT0, I2C_CMD_WDAT0, I2C_CMD_WDAT1, I2C_CMD_WDAT0,

I2C_CMD_WDAT0, I2C_CMD_WDAT1, I2C_CMD_WDAT1, I2C_CMD_WDAT0.
```

7.6.2 I2C Write Example

Here is an example command sequence that implements a one-byte write on the I2C interface:

```
I2C_CMD_STRT

I2C_CMD_WDAT8

{i2cDeviceAddress[7:1], 1’b0}  //Write to Device Address

I2C_CMD_VACK

I2C_CMD_WDAT8

i2cSlaveRegisterAddress[7:0]

I2C_CMD_VACK

I2C_CMD_WDAT8

i2cWriteData[7:0]

I2C_CMD_VACK

I2C_CMD_STOP

I2C_CMD_NULL
```

The write commences with the Master issuing a START in step 1, followed by the I2C Slave Device Address in steps 2 and 3. The Master checks for the Slave Acknowledge in step 4, then transmits the I2C Slave Register Address where the write is to occur in steps 5 and 6. The Master checks for the Slave Acknowledge in step 7, then transmits the data to be written in steps 8 and 9. The Master again checks for the Slave Acknowledge in step 10, then issues the STOP command in step 11, signifying the end of the transaction. The NULL command has no effect on the bus, but can be used to separate successive I2C transactions.

7.6.3 I2C Read Example

Here is an example command sequence that implements a one-byte read on the I2C interface:

```
I2C_CMD_STRT

I2C_CMD_WDAT8

{i2cDeviceAddress[7:1], 1’b0}  //Write to Device Address

I2C_CMD_VACK
```

```
I2C_CMD_WDAT8

i2cSlaveRegisterAddress[7:0]

I2C_CMD_VACK

I2C_CMD_START

I2C_CMD_WDAT8

{i2cDeviceAddress[7:1], 1’b1}  //Read from Device Address

I2C_CMD_VACK

I2C_CMD_RDAT8

I2C_CMD_WDAT1

I2C_CMD_STOP

I2C_CMD_NULL
```

The first seven commands for a read are the same as for a write, but here the I2C Register Address will be interpreted by the slave as the address from which the read occurs. The Read commences in step 8 with the START condition, and is confirmed in step 10 since the low bit in the I2C Device Address is a logic “high”. The Master checks for the Slave Acknowledge in step 11, then samples the read data in step 12. In step 13, the Master issues a Not Acknowledge, indicating that no further reads are requested, followed by a STOP in step 14 signifying the end of the transaction. The NULL command has no effect on the bus, but can be used to separate successive I2C transactions.

Some I2C Slaves will drive the I2C clock (SCL) low at the beginning of a read cycle. This happens when the I2C Slave needs more time to present the read data back to the I2C Master – the I2C Slave will hold the clock low until its data is ready. The I2C Engine contains clock-detect logic that determines if an I2C Slave is controlling the clock. If the I2C Engine determines that a slave is controlling the clock, then the I2C Engine will stay in its current state until it determines that the I2C Slave has released the clock. After the I2C Slave’s data is ready and control of the clock has been released to the I2C Master, the I2C Engine resumes normal operation, and can read the data presented by the I2C Slave.

7.7 Interrupts

The I2C can generate three types of interrupts: Error, Read Data FIFO Not Empty, and Command FIFO Empty.

7.7.1 Enabling Interrupts

The enabling and disabling of interrupts is controlled by the setting and clearing of bits in the Interrupt Enable Register. A logic “high” in the Interrupt Enable Register corresponds to an enabled interrupt while a logic “low” in the Interrupt Enable Register corresponds to a disabled interrupt. Note that the I2C Enable bit in the Control Register must be set also in order to enable the interrupt signal, blkInt.

Table 7-10: I2C Interrupts

IER Bit Number	Corresponding Interrupt Enabled
0	Command FIFO Empty
1	Read Data FIFO Not Empty
2	Error

The Command FIFO Empty interrupt can be used by the processor to commence writing up to 32 commands into the command FIFO.

The Read Data FIFO Not Empty interrupt can be used to alert the processor that an I2C read has occurred, and that the data is available.

The Error Interrupt signifies that an error condition has occurred. The status register contains more detailed information about the cause of the error.

Setting the Interrupt

The individual interrupt conditions are set by the Interrupt Logic in the I2C Module when it is determined that a change to the interrupt condition has been detected.

The Command FIFO Empty interrupt is set only upon the change from not empty to empty. By design, this interrupt condition is also set when the I2C module emerges from a reset condition (a low to high transition on PRESETn).

The Read Data FIFO Not Empty Interrupt is set only upon the change from empty to not empty.

The Error interrupt is set upon any of the following changes:

Command FIFO Normal Operation → Command FIFO Overflow

Read Data FIFO Normal Operation → Read Data FIFO Underflow

I2C Protocol Normal Operation → I2C Protocol Error

7.7.2 Clearing the Interrupt

The interrupt signal to the processor, blkInt, is cleared by reading the Status Register. Note that reading the Status Register may only clear the interrupt condition – further action from the processor may be necessary in order to address the cause of the interrupt.

If the Command FIFO Empty condition is the source of the interrupt, the processor may place up to 32 commands in the Command register. Or it may simply choose to set a software flag indicating that the Command FIFO is empty, for example if there is no data ready to be written to the I2C Module at the time of the interrupt. Until the Command FIFO is written to, the status register will indicate that the Command FIFO is empty, and no further interrupts due to a Command FIFO Empty condition will occur.

If the Read Data FIFO Not Empty condition is the source of the interrupt, then the processor should read the Read Data register until the Status register indicates that the Read Data FIFO is empty. Until the Read Data FIFO has been read enough times so that it is again empty, the status register will indicate that there is Read Data present in the Read Data FIFO, and no further interrupts due to a Read Data FIFO Not Empty condition will occur.

If an Error condition is the source of the interrupt, no further action is required by the processor. However, the programmer may want to address the problems in the program, specifically in the event of a FIFO error. A FIFO error should not occur during normal operation; its existence would indicate an indexing problem in the software. The Command FIFO Overflow and Read Data FIFO Underflow interrupt conditions are self-clearing, and the I2C Protocol Error condition is cleared by reading the Status register. In case of an I2C Protocol Error, this could indicate any number of problems, including an incorrect command sequence, a malfunctioning or incompatible Slave, an I2C Bus connection problem, or even an I2C Bus Clock frequency that is too fast for the I2C Slave.

7.8 FIFOs

There are two separate clock domains in the I2C module, the PCLK domain and the I2C System Clock domain. The two FIFOs effectively serve as the clock boundary between the two domains, as they require separate read and write clocks. (The several other “handshaking” signals that cross the clock boundary make no assumptions about the relative speed of the two clock domains, and are double-buffered to guard against metastability.)

The FIFOs implemented in the core can be customized. Specifically, the FIFO depth can be changed if more (or less) storage is desirable. The system has been verified in simulation using Xilinx FIFO models; any other implementation will require the replacement FIFOs to be compatible with the Xilinx models. The FIFO output is registered, so the first data input does not automatically appear at the output. At a minimum, a replacement FIFO will need the following signals:

Table 7-11: I2C Clock Domains

Signal	Direction	Clock Domain
Reset	Input	Asynchronous
Write Clock	Input	Write
Read Clock	Input	Read
Empty	Output	Write
Full	Output	Read
Overflow	Output	Write
Underflow	Output	Read

7.9 Timing Diagrams

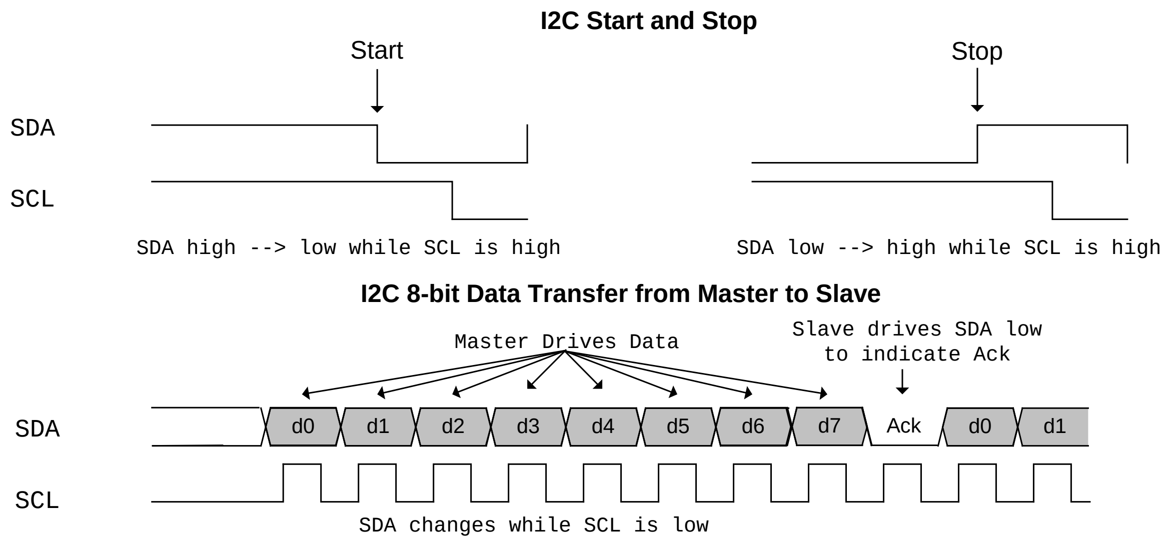

I2C Start and Stop
Start
Stop
SDA
SCL
SDA high --> low while SCL is high
SDA low --> high while SCL is high
I2C 8-bit Data Transfer from Master to Slave
Master Drives Data
Slave drives SDA low to indicate Ack
d0 d1 d2 d3 d4 d5 d6 d7 Ack d0 d1
SDA changes while SCL is low

Figure 7-2: Example I2C Bus Events

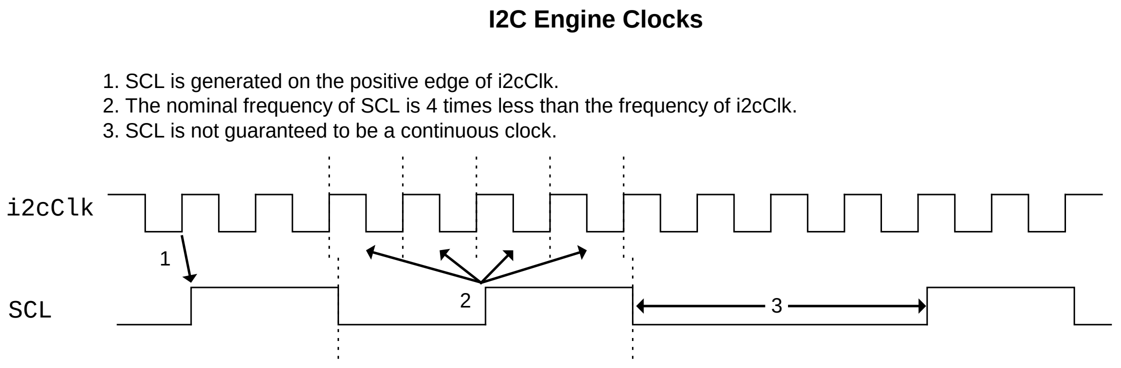

I2C Engine Clocks
1. SCL is generated on the positive edge of i2cClk.
2. The nominal frequency of SCL is 4 times less than the frequency of i2cClk.
3. SCL is not guaranteed to be a continuous clock.
i2cClk
SCL
1
2
3

Figure 7-3: Example I2C Clock Relationships

8. Serial Peripheral Interface (SPI) Controller

8.1 Overview

The Serial Peripheral Interface Bus Controller is synchronous serial data link controller.

The SPI bus controller can be configured under software control to be a master or slave device. Reading and writing the core is done on the AMBA APB bus interface. The core operates in 8 bit, 16 bit, 24 bit, and 32 bit data modes. The data is then serialized and then transmitted from master to slave device using the standard 4-wire SPI bus interface.

The data is transmitted synchronously with the MOSI (Master Out, Slave In) relative to the SCLK generated by the master device. The master also receives data on the MISO (Master In, Slave Out) signal in a full duplex fashion.

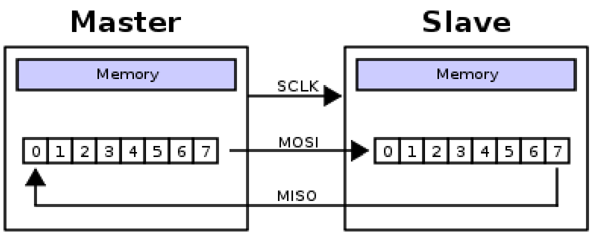


Figure 8-1: SPI Master - Slave Data Transmission

The SPI controller can be used with up to three slave devices. When the core is configured as a slave, the MISO signal is tristated to allow for multiple slaves to transmit data to the master when the slave's SSn control is active (or selected).

The following figure shows devices communicating in a master/slave mode where the master initiates the data frame transmission.

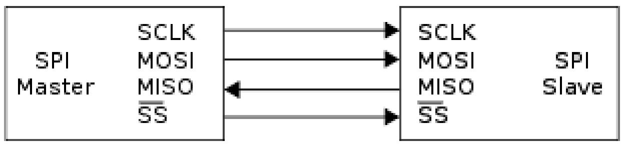


Figure 8-2: SPI Master - Slave interconnect

The following figure shows three slave devices connected to a master in a MISO tristate bus fashion.

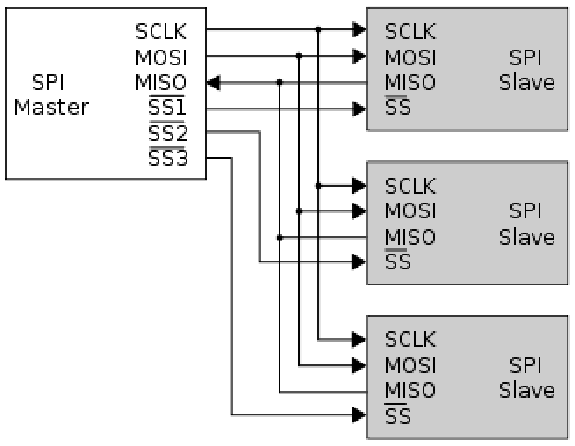


Figure 8-3: SPI Master to 3 Slaves interconnect

8.2 Block Diagram

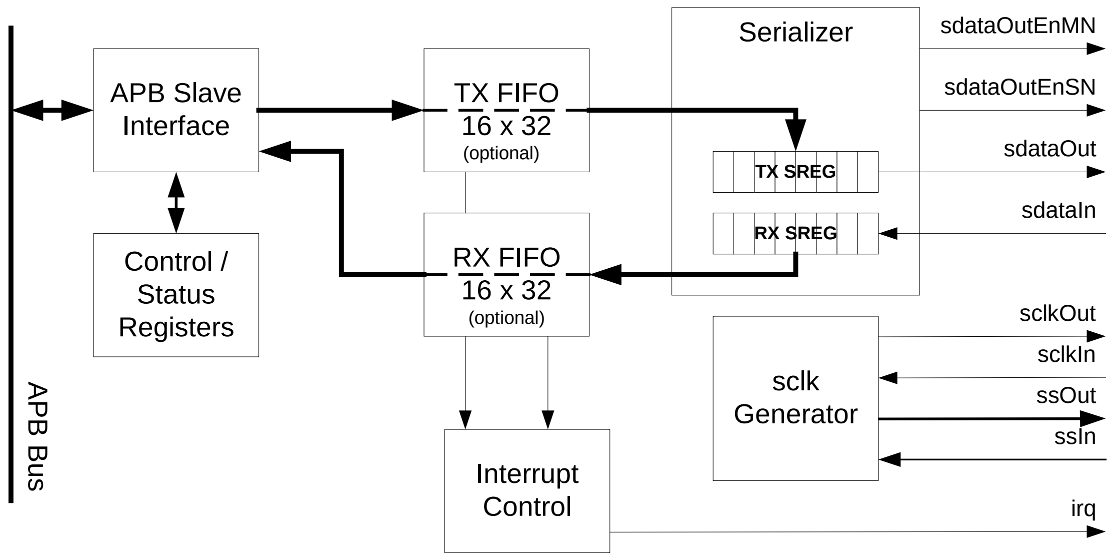


Figure 8-4: SPI Block Diagram

The figure below shows typical top-level connections that are external to the IPC-SPI-APB module. Note that inverters may be necessary for the output enable signals if the output buffers are not active low enabled.

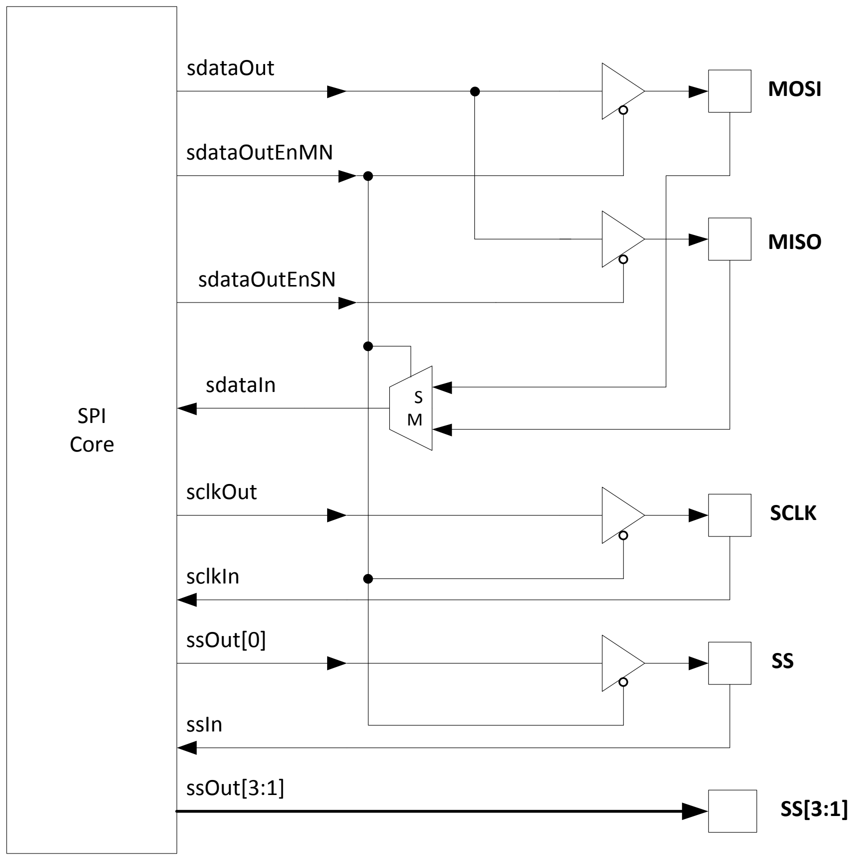


Figure 8-5: Typical IPC-SPI-APB I/O Connections

8.3 Signal Descriptions

Signal Name	Type	Description
APB Slave Interface		
PRESETn	I	Active low APB reset
PCLK	I	APB clock
PWRITE	I	APB Write (0=read 1=write)
PWDATA[31:0]	I	Write Data from the APB
PADDR[31:0]	I	APB Address bus (only 4 bits are used out of 32)
PSEL	I	APB Block select, enables the SPI module
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	Read data to the APB
SPI Interface		
sdataIn	I	SPI Serial Data Input
sdataOut	O	SPI Serial Data Output
sdataOutEnMN	O	sdataOut Master Active Low Output Enable – use with MOSI
sdataOutEnSN	O	sdataOut Slave Active Low Output Enable – use with MISO
sClkIn	I	SPI External Serial Clock Input
ssIn	I	SPI Serial Select Input – Slave
sClkOut	O	SPI External Serial Clock Output
ssOut[3:0]	O	SPI Slave Select Output from Master
Miscellaneous Signals		
intrReq	O	Interrupt Request (1 = interrupt, 0 = no interrupt)

8.4 Build Options

Definition	Default	Description
USE_SPI_FIFOS	defined	If this is defined, FIFOs are used for transmit and receive data. If this is not defined, a single register is used for these purposes

8.5 Register Description

Table 8-1: Offset 0x00: Serial Data Transmit (to Tx FIFO)

Bits	Signal	WO	Default	Description
[31:0]	data reg		0	Serial Transmit Data Register

Table 8-2: Offset 0x04: Serial Data Receive (from Rx FIFO)

Bits	Signal	RO	Default	Description
[31:0]	data reg		0	Serial Receive Data Register

Table 8-3: Offset 0x08: Serial Clock Divisor

Bits	Signal	R/W	Default	Description
[31:0]	divisor		0x00	Divisor for serial clock. SCLK = PCLK/2(divisor+1)

Table 8-4: Offset 0x0C: Control Register

Bits	Signal	R/W	Default	Description
[7:6]	bitsize		0	00 = 8 bit serial mode, 01 = 16 bit serial mode, 10 = 32 bit serial mode 11 = 24 bit serial mode
[5]	master		0	SPI port bus master mode of operation; 0=slave mode, 1=master mode
[4]	sclkPol		0	0 = sclkOut active high (default), 1= sclkOut active low
[3]	leadingEdgeTx		0	0=transmit sdataOut with slave select signal, and with each falling edge of internal SCLK while the transfer is active (except for the last falling edge). 1=transmit sdataOut with each rising edge of internal SCLK while the transfer is active.
[2]	msb1st		0	MSB/LSB 0=MSB first, 1=LSB first
[1]	sampleData		0	0=sample sdataIn on different edge of internal SCLK as used to transmit sdataOut. 1=sample sdataIn on same edge of internal SCLK as used to transmit sdataOut.
[0]	enable		0	SPI port enable 0=Disable SPI TX and RX; RX and TX shift registers and FIFOs (if using FIFOs) are reset. 1=Enable SPI TX and RX;
[7:6]	bitsize		0	00 = 8 bit serial mode, 01 = 16 bit serial mode, 10 = 32 bit serial mode 11 = 24 bit serial mode

Table 8-5: Offset 0x10: Status Register (USE_SPI_FIFOS is defined)

Bits	Signal	RO	Default	Description
[7]	rxFull		0	Receive FIFO full flag
[6]	rxHalf		0	Receive FIFO half empty flag based on watermark (control reg)
[5]	rxEmpty		1	Receive FIFO empty flag
[4]	txFull		0	Transmit FIFO full flag
[3]	txHalf		0	Transmit FIFO half empty flag based on watermark (control reg)
[2]	txEmpty		1	Transmit FIFO empty flag
[1]	xferError		0	Transmit or received terminated before all bits transferred.
[0]	xferIP		0	transfer is in progress

Table 8-6: Offset 0x10: Status Register (USE_SPI_FIFOS is not defined)

Bits	Signal	RO	Default	Description
[7]	rxFull		0	Receive register full flag
[6]			0	Unused
[5]	rxEmpty		1	Receive register empty flag
[4]	txFull		0	Transmit register full flag
[3]			0	Unused
[2]	txEmpty		1	Transmit register empty flag
[1]	xferError		0	Transmit or received terminated before all bits transferred.
[0]	xferIP		0	transfer is in progress

Table 8-7: Offset 0x14: Slave Select Register

Bits	Signal	R/W	Default	Description
[3]	ssout[3]		0	0=Do not select slave 3for TX/RX. 1=Select Slave 3 for TX/RX.
[2]	ssout[2]		0	0=Do not select slave 2for TX/RX. 1=Select Slave 2 for TX/RX.
[1]	ssout[1]		0	0=Do not select slave 1for TX/RX. 1=Select Slave 1 for TX/RX.
[0]	ssout[0]		0	0=Do not select slave 0for TX/RX. 1=Select Slave 0 for TX/RX.

Table 8-8: Offset 0x18: Slave Select Polarity Register

Bits	Signal	R/W	Default	Description
[3]	sspol3		0	0 =ssOut[3] (Master) is an active low signal 1= ssOut[3] (Master) is an active high signal
[2]	sspol2		0	0 =ssOut[2] (Master) is an active low signal 1= ssOut[2] (Master) is an active high signal
[1]	sspol1		0	0 =ssOut[1] (Master) is an active low signal 1= ssOut[1] (Master) is an active high signal
[0]	sspol0		0	0 =ssOut[0] (Master) or ssIn (Slave) is an active low signal 1= ssOut[0] (Master) or ssIn (Slave) is an active high signal

Table 8-9: Offset 0x1C: Interrupt Enable Register

Bits	Signal	R/W	Default	Description
[7:0]	intrEnable		0	Enable interrupt

Table 8-10: Offset 0x20: Interrupt Status Register

Bits	Signal	RO	Default	Description
[7:0]	intrStatus		0	Interrupt Status 1 = interrupt source is active, 0 = no interrupt

Table 8-11: Offset 0x24: Interrupt Clear Register

Bits	Signal	WClr	Default	Description
[7:0]	intrClear		0	Clear Interrupt Writing a 1 clears the interrupt bit. Writing a 0 = nop.

Table 8-12: Interrupt Bit Mapping

Bit	Mapping using FIFOs	Mapping without FIFOs
[7]	rx_full	--
[6]	rx_half	--
[5]	rx_empty	--
[4]	tx_full	--
[3]	tx_half	--
[2]	tx_empty	xferDone
[1]	xferError	xferError
[0]	ssInSync	ssInSync

Note: FIFO interrupts are edge triggered. See Status Register for actual FIFO flag status.

Table 8-13: Offset 0x28: TX FIFO Watermark Level Register (only valid when USE_SPI_FIFOS is defined)

Bits	Signal	R/W	Default	Description
[3:0]	rx_watermark		0	Transmit FIFO watermark level for triggering half full interrupt.

Table 8-14: Offset 0x2C: RX FIFO Watermark Level Register (only valid when USE_SPI_FIFOS is defined)

Bits	Signal	R/W	Default	Description
[3:0]	rx_watermark		0	receive FIFO watermark level for triggering half full interrupt.

Table 8-15: Offset 0x30: FIFO Level Register

Bits	Signal	RO	Default	Description
[3:0]	tx_FIFO_level		0	Read transmit FIFO current level.

Table 8-16: Offset 0x34: FIFO Level Register

Bits	Signal	RO	Default	Description
[3.0]	rx_FIFO_level		0	Read receive FIFO current level.

8.6 APB Interface Timing

The timing of the SPI APB interface relies on the APB Master that is initiating the transaction. Usually, the transaction is initiated by an AHB to APB Bridge module, and passed through to the SPI module via an APB Channel module.

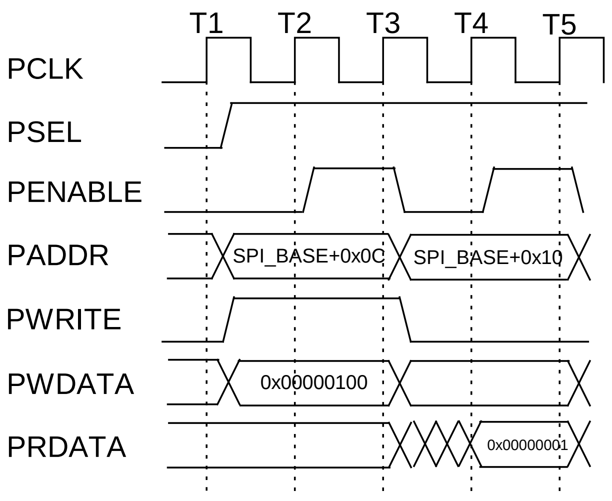


Figure 8-6: APB Interface Timing

Time T1:

The APB Master drives a new transfer on the APB bus at address SPI_BASE + 0x0C. This transfer is a write of 0x00000100 to the SPI Control Register. The transaction will configure the SPI Module to be enabled, and to use default modes.

Time T2:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T3.

The SPI module samples address SPI_BASE + 0x0C along with PWRITE = 1’b1 and PSEL = 1’b1. PWDATA will be sampled at time T3.

Time T3:

The APB Master drives PENABLE = 1’b0. Also, the APB Master drives a new transfer on the APB bus at address SPI_BASE + 0x10. This transfer is a read of the SPI Status Register.

The SPI module samples PENABLE = 1’b1, and registers 0x00000100 from PWDATA, thus altering the SPI Configuration register.

Time T4:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T5.

The SPI module samples SPI_BASE + 0x10 along with PWRITE = 1’b0 and PSEL = 1’b1. As a result, the SPI module drives PRDATA = 0x00000001 (reflecting the state of the Status register) back toward the APB Master.

Time T5:

The APB Master drives PENABLE = 1’b0. Since the current transaction is a read, the APB Master samples PRDATA = 0x00000001 at this time.

8.7 SPI Master Data Transfer

In general, different SPI devices may have very different timing considerations. For this reason, the APB SPI supports a highly configurable timing arrangement that is governed by two bits in the control register: Leading Edge Transmit and Sample Edge. These two bits determine which edge(s) of the SPI clock on which data is transmitted and sampled. The following diagrams show the effects of these bits for an 8-bit data transfer.

Internally, the SPI engine assumes that SCLK is low by default and that the first transition of SCLK during a SPI transaction is from low to high as shown in the following diagrams. If the external clock is (or needs to be) high by default, the SCLK Polarity bit in the control register should be set. Setting the SCLK Polarity bit has the following effects: for a master, sclkOut is inverted; for a slave, sclkIn is inverted.

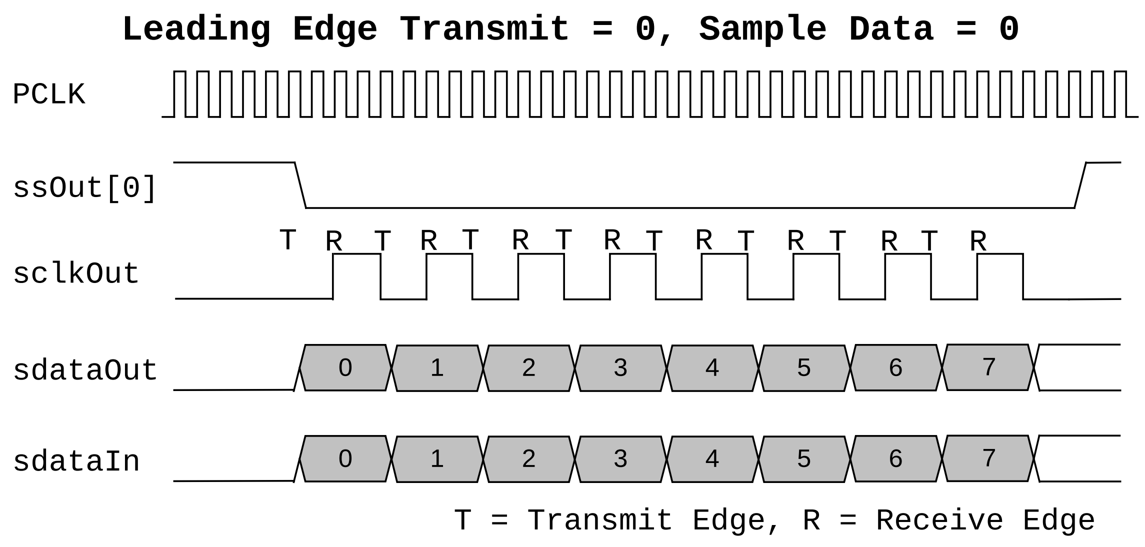

Leading Edge Transmit = 0, Sample Data = 0
PCLK
ssOut[0]
sclkOut
T R T R T R T R T R T R T R T R
sdataOut
0 1 2 3 4 5 6 7
sdataIn
0 1 2 3 4 5 6 7
T = Transmit Edge, R = Receive Edge

Figure 8-7: SPI Master Data Transfer for LET = 0, SD = 0.

In the above figure the SPI SCLK from the master is divided down (by 4) from the APB clock (PCLK). The control register has been written to select master mode and to enable the SPI. A data word has been written to the transmit register. It is assumed that the selected slave is selected with an active low signal (ssOut). Data is transmitted on the falling edge of SCLK and incoming data is expected to be stable on the rising edge of SCLK. Note that the first data bit has been transmitted before any SCLK edge; this is so the selected slave module can sample the data on the first rising edge of SCLK.

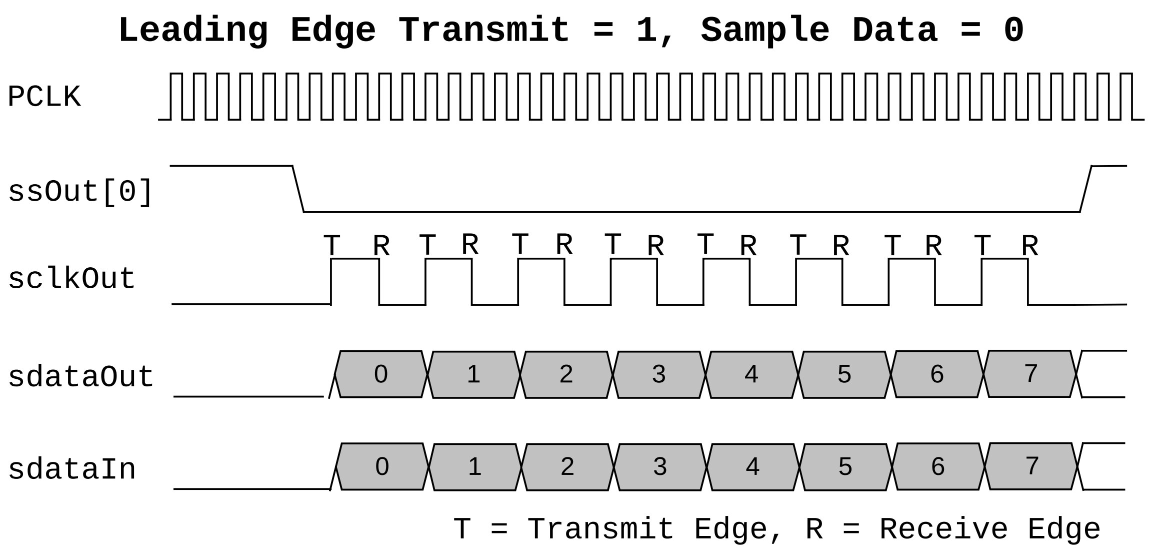

Leading Edge Transmit = 1, Sample Data = 0
PCLK
ssOut[0]
sclkOut
sdataOut
sdataIn
T R T R T R T R T R T R T R T R
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
T = Transmit Edge, R = Receive Edge

Figure 8-8: SPI Master Data Transfer for LET = 1, SD = 0.

In the above figure the SPI SCLK from the master is divided down (by 4) from the APB clock (PCLK). The control register has been written to select master mode and to enable the SPI. A data word has been written to the transmit register. It is assumed that the selected slave is selected with an active low signal (ssOut). Data is transmitted on the rising edge of SCLK and incoming data is expected to be stable on the falling edge of SCLK. Note that the first data is transmitted on the leading edge of SCLK; this is so the selected slave module can sample the data on the first falling edge of SCLK.

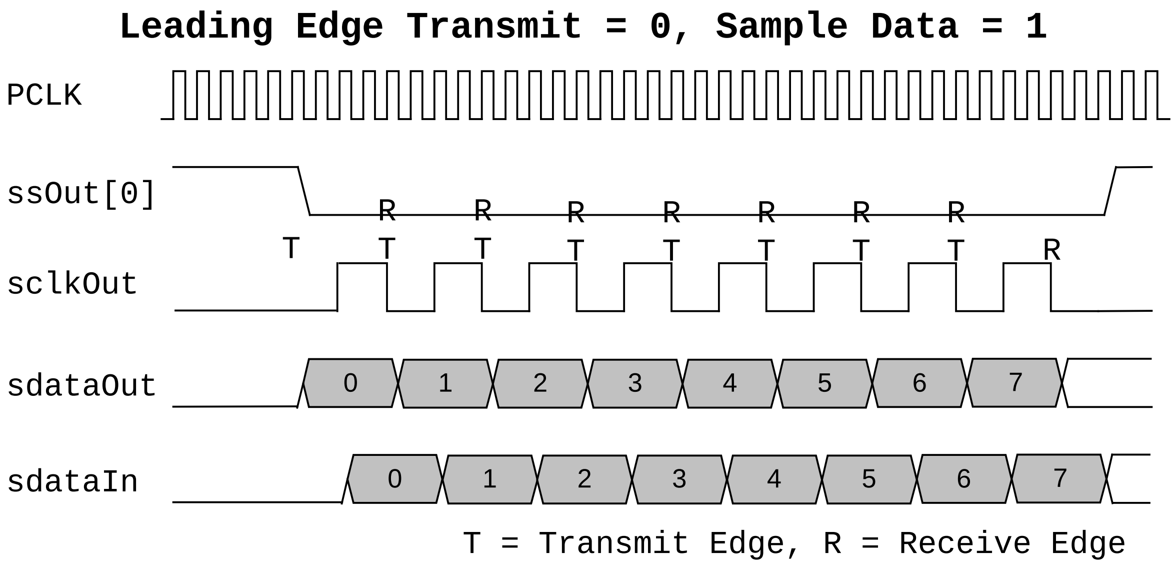


Figure 8-9: SPI Master Data Transfer for LET = 0, SD = 1.

In the above figure the SPI SCLK from the master is divided down (by 4) from the APB clock (PCLK). The control register has been written to select master mode and to enable the SPI. A data word has been written to the transmit register. It is assumed that the selected slave is selected with an active low signal (ssOut). Data is transmitted on the falling edge of SCLK and incoming data is expected to be stable on the falling edge of SCLK. Note that the first data is transmitted before any SCLK edge; this is so the selected slave module can sample the data on the first rising edge of SCLK.

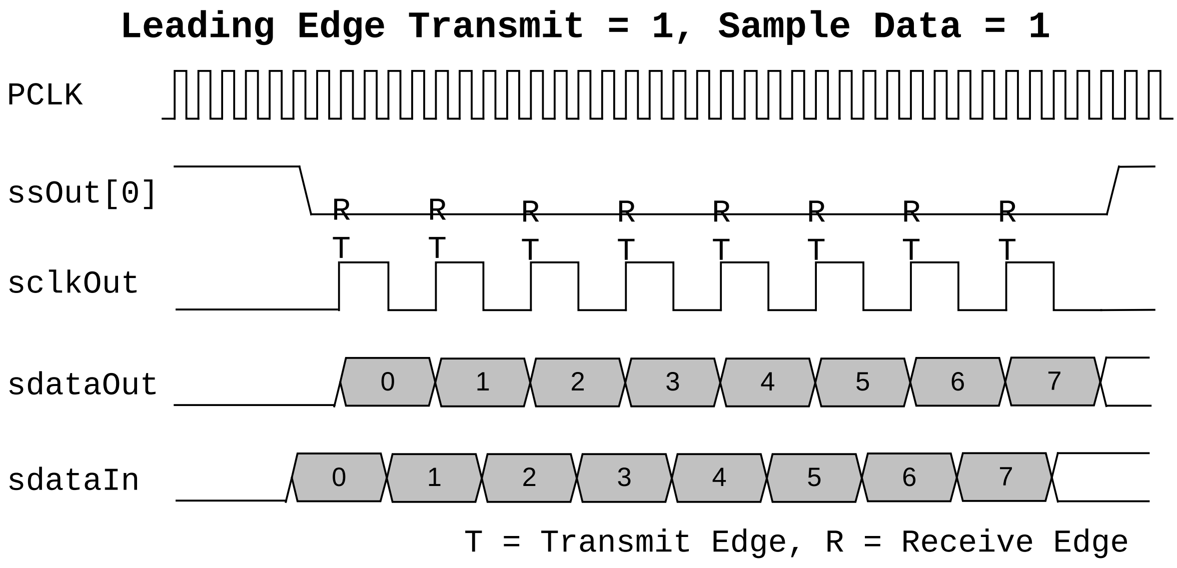

Leading Edge Transmit = 1, Sample Data = 1
PCLK
ssOut[0]
sclkOut
sdataOut
sdataIn
R
T
0
1
2
3
4
5
6
7
T = Transmit Edge, R = Receive Edge

Figure 8-10: SPI Master Data Transfer for LET = 1, SD = 1.

In the above figure the SPI SCLK from the master is divided down (by 4) from the APB clock (PCLK). The control register has been written to select master mode and to enable the SPI. A data word has been written to the transmit register. It is assumed that the selected slave is selected with an active low signal (ssOut). Data is transmitted on the rising edge of SCLK and incoming data is expected to be stable on the rising edge of SCLK. Note that the first data is transmitted on the leading edge of SCLK; this is so the selected slave module can sample the data on the first falling edge of SCLK.

8.8 SPI Slave Data Transfer

All of the discussions and diagrams that concern the Master operation can also be applied to Slave operation. There is one difference, however, that limits the frequency ratio of PCLK to SCLK when in Slave mode. In the APB SPI, SCLK is sampled by PCLK; a double-synchronized version of SCLK is used internally by the logic. The result of the double-synchronization is that there are 2-3 cycles of PCLK delay between the actual SCLK edge and when the SCLK edge is detected internally.

The discussions about the timing of a Master transfer all apply to Slave operation as well, with the exception that instead of sclkOut (SCLK), the Slave uses a registered version of SCLK for its operation: sclkInSync. The timing relationship between SCLK and sclkInSync is shown below.

Usually, the frequency of a typical SPI bus SCLK will be much less than the frequency of PCLK. In these cases, the delay between SCLK and sclkInSync is negligible. However, as the frequency of SCLK approaches the frequency of PCLK, these effects become more prevalent, and proper operation may cease if the frequency ratio is less than 8.

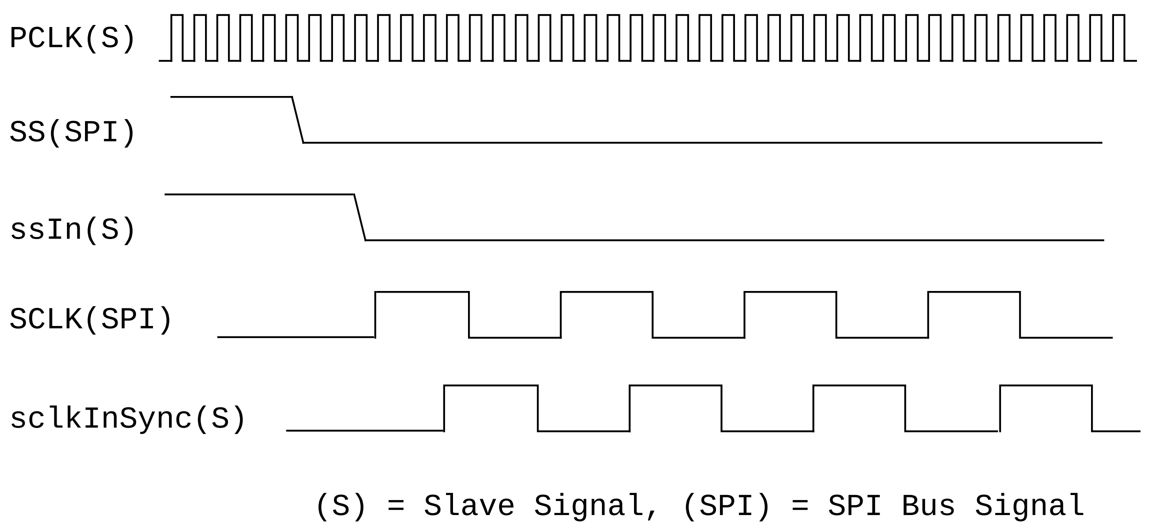


Figure 8-11: SPI Slave Clock Relationships.

The diagram above shows what happens as the frequency of SCLK is a factor of 8 slower than the frequency of PCLK. Since the first edge of sclkInSync on the slave occurs before the first falling edge of sclkOut, this scenario is still valid. However, if SCLK were any faster, the first rising edge of sclkInSync would be after the first falling edge of SCLK, and the SPI communication would not work in most cases.

8.9 Programming

8.9.1 Reset

Setting Control Register enable (bit[8]) to “0” resets the SPI port. Resetting the port clears the tx and rx FIFOs as well as clears the tx and tx serial shift registers.

8.9.2 Setup Sequence

i) Write Control Register
   (1) Disable
ii) Program the Divisor, the SS polarity, and set FIFO watermarks (if FIFOs are used)
iii) Clear any interrupts by writing ‘hFF to Interrupt Clear Register.
iv) If using interrupts, set interrupt masks by writing the Interrupt Mask Register.
v) If using interrupts, enable interrupts by writing the Interrupt Enable Register.
vi) Set the polarity of the select lines by writing the Slave Select Polarity Register
vii) If a master, select which slave or slaves are to be selected by writing to the Slave Select Register
viii) Write Control Register
   (1) bitsize
   (2) master or slave mode
   (3) sclk polarity
   (4) leadingEdgeTx
   (5) msb1st
   (6) sampleData
   (7) enable

8.9.3 Transmit Sequence Non-FIFO mode

i) Write byte/word to Serial Data Transmit Register
ii) Read Status register to check ssIn (transfer in progress) and xferError.

iii) If there is no xferError and ssIn is not active then the next byte/word can be written to the Serial Data Transmit Register.

8.9.4 Transmit Sequence FIFO mode

i) Read FIFO Level Register to determine the number of bytes/words you can write.

ii) Write byte/word(s) to Serial Data Transmit Register.

iii) Read Interrupt Status register to check ssIn (transfer in progress) and tx FIFO Full flags before writing additional bytes/words.

iv) Alternatively, use an interrupt service routine to to interrupt when the TX FIFO is empty or below half full; write the appropriate number of bytes/words to the Serial Data Transmit register in the interrupt service routine.

8.9.5 Receive Sequence Non-FIFO mode

Upon interrupt or read of Interrupt Status Register:

i) If xferDone flag is set and xferError flag is false then read byte/word from the Serial Data Receive Register.

ii) If using interrupts, clear interrupts by writing “1”s to the Interrupt Clear Register.

8.9.6 Receive Sequence FIFO mode

Upon Rx Data Full/Half interrupt or read of Status Register:

i) If rx_not_empty flag is set and xferError flag is false then read byte(s)/word(s) from the Serial Data Receive Register.

ii) If using interrupts, clear interrupts by writing “1”s to the Interrupt Clear Register.

8.9.7 Transmit or Receive Error

Upon interrupt or read of Interrupt Status Register, if xferError flag is true, then reset the SPI by following sequence:

i) Read Control Register

ii) Logical AND “0x0FF” to the register value.

iii) Write the Control Register to disable the SPI.

iv) Logical AND “0x1FF” to the register value

v) Write the Control Register again to enable the SPI.

9. External Bus Interface

9.1 Overview

The AHB External Bus Interface (EBI) allows the processor to transmit and receive data to an external device, usually a memory (SRAM, Flash, etc.). The number of read wait states, the number of write wait states, and the memory width are all configurable through the APB register interface of the EBI. The EBI allows word, half-word, and byte width addressing to 32-bit, 16-bit, and 8-bit external devices.

9.2 Block Diagram

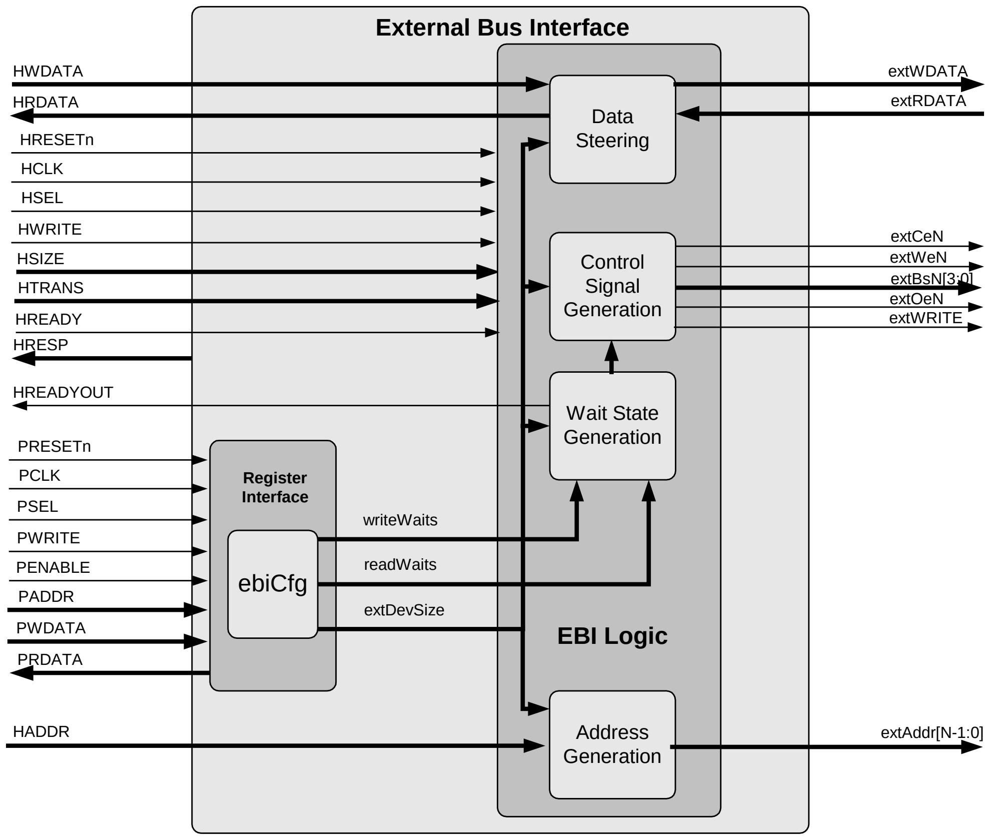


Figure 9-1: EBI Bock Diagram

9.3 Signal Descriptions

Table 9-1: EBI Interface Signals

Signal Name	I/O	Description
AHB Interface		
HCLK	I	AHB clock
HRESETn	I	AHB reset (active low)
HADDR[31:0]	I	AHB address bus
HWDATA[31:0]	I	32-bit write data bus from AHB
HWRITE	I	write select from AHB (0 = read, 1 = write)
HSIZE[2:0]	I	memory access size select signal from AHB
HTRANS[1:0]	I	AHB transfer type
HSEL	I	AHB block select signal for memory
HREADY	I	Combined AHB ready signal from all AHB slaves
HREADYOUT	O	ready signal produced by EBI, used to hold AHB while memory access completes

HRESP[1:0]	O	AHB Slave response
HRDATA[31:0]	O	32-bit read data bus to processor
APB Interface		
PCLK	I	APB clock
PRESETn	I	APB reset (active low)
PADDR[31:0]	I	APB address bus
PWDATA[31:0]	I	32-bit write data bus from APB
PWRITE	I	write select from APB (0 = read, 1 = write)
PSEL	I	APB block select signal for register accesses
PENABLE	I	APB Enable signal
PRDATA[31:0]	O	32-bit read data bus to processor
External Memory		
extRDATA[31:0]	I	32-bit read data bus from external devices
extCeN	O	Active low chip enable for external device
extOeN	O	Active low output enable for external device
extWeN	O	Active low write enable for external device
extBsN[3:0]	O	Active low byte selects for external device
extWRITE	O	write select to external device (0 = read, 1 = write)
extWDATA[31:0]	O	32-bit write data bus to external devices
extADDR[N_EXT_ADDR-1:0]	O	address bus to external devices

9.4 Programming Interface

9.4.1 Register Descriptions

Table 9-2: Register Summary

Register Name	Offset	Description
Configuration	0x00	Configuration register for: bus width read wait states write wait states

Table 9-3: Configuration Register (offset 0x00)

Bits	Signal	Access	Default	Description
[15:12]	reserved	R/W	4’b0000	reserved
[11:8]	writeWaits	R/W	4’b1100	Number of clock cycles that HREADYOUT is assigned 1’b0 in order to execute a 32-bit write to a 32-bit external device. For 16-bit and 8-bit devices, HREADYOUT is assigned 1’b0 for (writeWaits*2) and (writeWaits*4) clock cycles respectively. writeWaits must be in the range [0x1, 0xF]; 0 is not supported.
[7:4]	readWaits	R/W	4’b1100	Number of clock cycles that HREADYOUT is assigned 1’b0 in order to execute a 32-bit read from a 32-bit external device. For 16-bit and 8-bit devices, HREADYOUT is assigned 1’b0 for (readWaits*2) and (readWaits*4) clock cycles respectively. readWaits must be in the range [0x1, 0xF]; 0 is not supported.
[3:2]	reserved	R/W	2’b00	reserved
[1:0]	busWidth	R/W	2’b10	Data bus width of device used: 0 = 8 bits (byte) 1 = 16 bits (half word) 2 = 32 bits (word)

				3 = reserved

9.5 Functional Description

The EBI translates AHB writes and reads into writes and reads for an external device. In order to accomplish this task, the EBI is comprised of five major functional blocks: APB Register Interface, Wait State Generation, Address Generation, Control Signal Generation, Data Steering.

9.5.1 APB Register Interface

The APB Register Interface is used to configure the EBI to interface to a particular memory width (8, 16, or 32 bits wide), and to configure the number of wait states for reads and writes to the device. Three signals are produced by the configuration register which are used internally by other functional blocks in the design: writeWaits, readWaits, extDevSize. A synchronization signal is also provided from the register interface to the main logic since signals are being transferred from the PCLK domain to the HCLK domain. In practice these are usually the same clock, but that assumption is not made.

9.5.2 Wait State Generation

Two types of wait states are generated by this module: AHB wait states and external device wait states. For a given AHB transaction, the EBI must assert HREADYOUT = 1'b0 until the memory transaction has completed. External device wait states are necessary because different memories have different requirements for minimum hold times of the various signals that cause a transaction to occur. These may not technically be wait states from the perspective of an asynchronous memory, but they are wait states in terms of the HCLK logic in the EBI.

The two types of wait states are the same duration if HSIZE and extDevSize refer to the same width (e.g. both 8-bit, 16-bit, or 32-bit), or if HSIZE refers to a smaller width than extDevSize. If extDevSize refers to a smaller width than HSIZE, then the number of AHB wait states is either 4X or 2X the number of device wait states. For example, if HSIZE refers to a 32-bit transfer and extDevSize specifies an 8-bit memory, 4 different transactions must occur with the memory during the time that HREADYOUT is held at 1'b0. An internal transfer counter keeps track of the transfers to the external device.

9.5.3 Address Generation

The EBI generates the extADDR signal based off a registered version of HADDR and an internal transfer counter. Similar to what occurs in the Wait State Generation, if the width referred to by HSIZE <= the width referred to by extDevSize, then the internal transfer counter is not needed in the generation of extADDR. Otherwise, the transfer counter is used to modify the lower two bits of extADDR in order to have the transactions to the device complete correctly.

9.5.4 Control Signal Generation

Control Signal Generation is at the heart of the EBI. This block generates the control signals (extCeN, extBsN, extWeN, extOeN) that directly control the external device, and the extWRITE signal which can be used to control a bi-directional data bus to the external device.

extCeN is generated from a registered version of HSEL. This signal remains active for the duration of the AHB transfer, regardless of the actual number of memory transfers.

extOeN is generated from registered versions of HSEL and HWRITE. This signal remains active for the duration of the AHB transfer, regardless of the actual number of memory transfers.

extWRITE is generated from registered versions of HSEL and HWRITE. This signal remains active for the duration of the AHB transfer, regardless of the actual number of memory transfers.

extBsN is generated using registered versions of HSEL, HADDR, and HSIZE, as well the internal signal extDevSize. This signal remains active for the duration of the AHB transfer, regardless of the actual number of memory transfers.

extWeN is generated using registered versions of HSEL, HWRITE, and an internal wait state counter that indicates that an external transfer is nearing completion. This signal will toggle for each external transfer that occurs for a given AHB transaction.

9.5.5 Data Steering

The external bus interface (EBI) allows memory to be written to in word, half-word, and byte width; 32-bit, 16-bit, and 8-bit memories are acceptable. To do this, all bits are sent by an AHB Master (usually a processor) via the 32-bit HWDATA signal; the HSIZE signal determines the width of the transaction. If the transaction width (HSIZE) is not a 32-bit transfer, then the relevant data is steered to the least significant part of extWDATA. For example, when executing a 32-bit write with data 0x44332211 to a 16-bit memory, extWDATA would be 0x00002211 for the first write and 0x00004433 for the second write.

For reading, a similar process is used, except that the Data Steering assembles HRDATA from 4, 2, or 1 reads of the external device. In the case where HSIZE <= extDevSize, extRDATA is replicated in 1, 2, or 4 areas of HRDATA. In the case where extDevSize < HSIZE, HRDATA is assembled from 4 or 2 reads of the external device. The placing of the various bytes in HRDATA conforms to AHB specifications.

9.6 Timing Diagrams

This section will be divided into separate discussions concerning the configuration register interface and the external device interface.

9.6.1 Configuration Register Timing

The timing of the EBI's APB interface relies on the APB Master that is initiating the transaction. Usually, the transaction is initiated by a processor, and passed through to an AHB Channel to an AHB-to-APB Bridge module which acts as the APB Master.

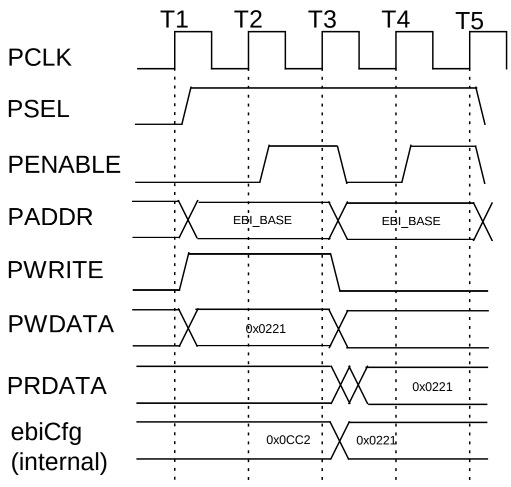


Figure 9-2: APB Interface Timing

Time T1:

The APB Master drives a new transfer on the APB bus at address EBI_BASE. This transfer is a write to the Configuration Register to configure writeWaits=2, readWaits=2, memory width=16-bit.

Time T2:

The APB Master drives PENABLE = 1'b1 to indicate to the slave that the APB data phase will occur at time T3.

The EBI module samples PADDR = EBI_BASE along with PWRITE = 1'b1 and PSEL = 1'b1. PWDATA will be sampled at time T3.

Time T3:

The APB Master drives PENABLE = 1'b0. Also, the APB Master drives a new transfer on the APB bus at address EBI_BASE. This transfer is a read of the Configuration Register.

The EBI Module samples PENABLE = 1'b1, and latches 0x0221 from PWDATA. This write has the effect of reconfiguring write wait states, read wait states, and memory width.

Time T4:

The APB Master drives PENABLE = 1’b1 to indicate to the slave that the APB data phase will occur at time T5.

The EBI module samples PADDR = EBI_BASE along with PWRITE = 1’b0 and PSEL = 1’b1. As a result, the Timer module drives PRDATA = 0x3E (reflecting the state of the Value Register) back toward the APB Master.

Time T5:

The APB Master drives PENABLE = 1’b0. Since the current transaction is a read, the APB Master samples PRDATA = 0x0221 at this time.

9.6.2 AHB and EBI Timing

The following timing diagram shows timing for writes and reads to external memories. 32-bit, 16-bit, and 8-bit wide external memory accesses are shown. Writes are discussed in detail; the diagram for reads closely parallels the diagram for writes. In the diagrams, grayed out areas indicate that the processor is accessing other peripherals or doing internal accesses during these times.

Here are some general notes that apply to the timing:

The wait state signal HREADYOUT is asserted low at the beginning of the transaction and returns one clock before the end of the completion of the transaction.

Timing for the output chip enables, write enables, byte lane selects, and output enables are short combinational delays from the registered address and control signals, notably regHSEL and regHWRITE.

The total number of clock cycles for each external memory device transaction is the number of wait states plus one.

It is assumed that the external memory device will register the write data on the rising edge of the write enable signal.

On a read, the processor will sample the data on the rising edge of the next clock where HREADYOUT is high.

On a read, extCeN and extOeN have the same timing. For a write, extOeN remains in the inactive state (1’b1).

On a read in which HSIZE > extDevSize, each read from the external device is stored; HRDATA is assembled from the stored values and presented to the AHB Master at the end of the transaction.

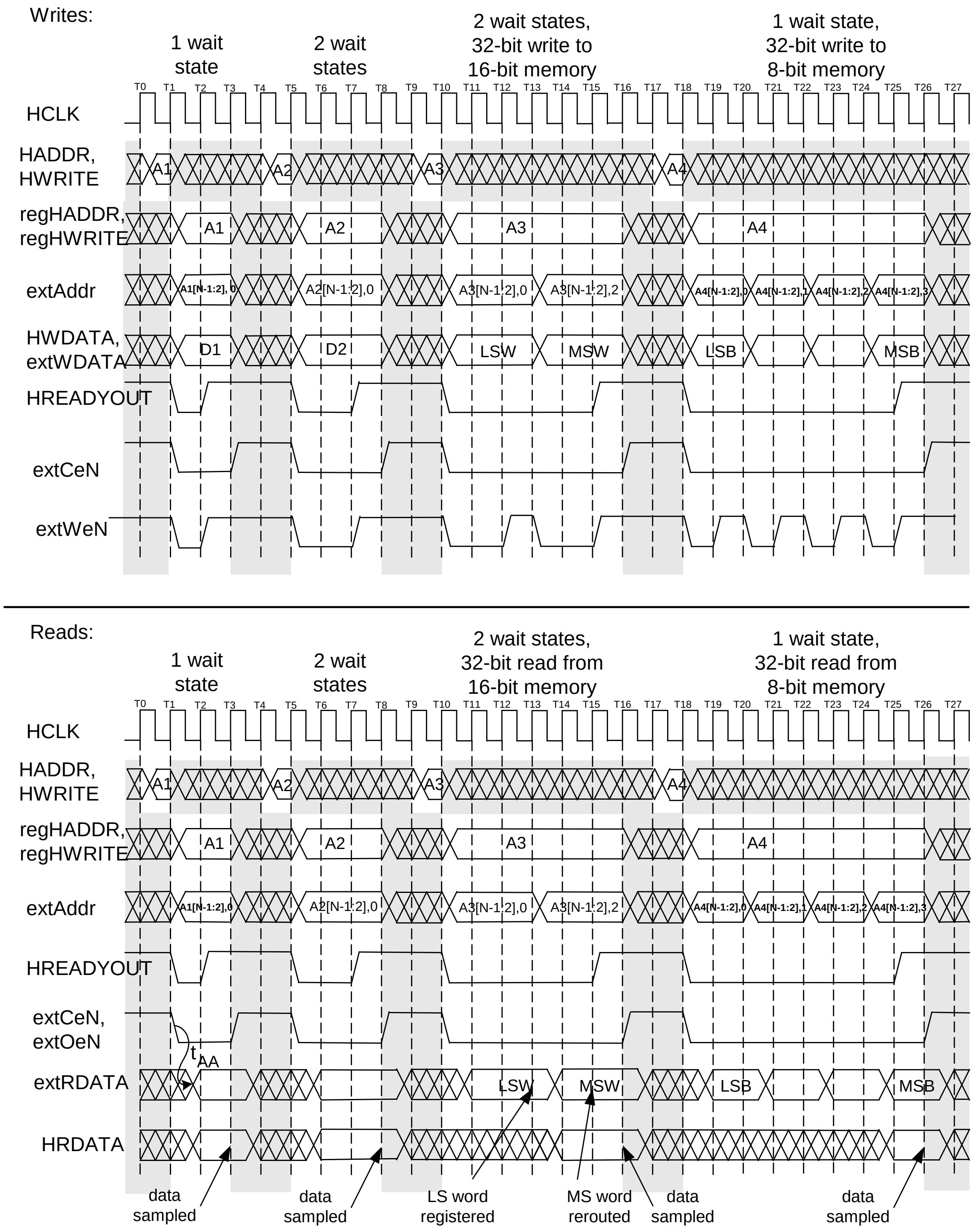


Figure 9-3: AHB, Memory Interface Timing

Time T0:

The AHB Master drives a new transfer on the AHB bus at address HADDR = A1. This transfer is a write to an external memory through the EBI. (The EBI has previously been configured for a single wait state on this memory.)

Time T1:

The AHB Master detects the combined HREADY signal high, and begins another transaction. This transfer is not to the EBI module.

The EBI module samples the transaction with address HADDR = A1, and registers the AHB address & control signals; the registered AHB address & control signals are used to generate the signaling that is presented to the external memory. The EBI also asserts HREADYOUT = 1’b0 for a single AHB wait state. The EBI also drives extCeN, extBsN, and extWeN to enable the chip, enable a write, and enable the various byte lanes.

Time T2:

The EBI drives HREADYOUT = 1’b1, indicating the end of the wait-stated transaction, and drives extWeN = 1’b1, indicating to the memory that the write should be executed.

Time T3:

The AHB Master samples HREADY = 1’b1, but does nothing of consequence here.

The EBI module drives extCeN = 1’b1, indicating the end of the transfer. (In the case of back-to-back transfers, the next transfer would be registered here, and extCeN would remain 1’b0.)

The (asynchronous) external device has detected a low-to-high transition on extWeN sometime between T2 and T3 while extCeN = 1’b0. The write is executed to the memory upon the transition of extWeN.

Time T4:

The AHB Master drives a new transfer on the AHB bus at address HADDR = A2. This transfer is a write to an external memory through the EBI. (The EBI has previously been configured for two wait states on this memory.)

Time T5:

The AHB Master detects the combined HREADY signal high, and begins another transaction. This transfer is not to the EBI module.

The EBI module samples the transaction with address HADDR = A2, and registers the AHB address & control signals; the registered AHB address & control signals are used to generate the signaling that is presented to the external memory. The EBI also asserts HREADYOUT = 1’b0 to begin the two AHB wait states. The EBI also drives extCeN, extBsN, and extWeN to enable the chip, enable a write, and enable the various byte lanes.

Time T6:

Nothing of consequence occurs during this wait state.

Time T7:

The EBI drives HREADYOUT = 1’b1, indicating the end of the wait-stated transaction, and drives extWeN = 1’b1, indicating to the memory that the write should be executed.

Time T8:

The AHB Master samples HREADY = 1’b1, but does nothing of consequence here.

The EBI module drives extCeN = 1’b1, indicating the end of the transfer. (In the case of back-to-back transfers, the next transfer would be registered here, and extCeN would remain 1’b0.)

The (asynchronous) external device has detected a low-to-high transition on extWeN sometime between T2 and T3 while extCeN = 1’b0. The write is executed to the memory upon the transition of extWeN.

Time T9:

The AHB Master drives a new transfer on the AHB bus at address HADDR = A3. This transfer is a write to an external memory through the EBI. (The EBI has previously been configured for two wait states on this memory.)

Time T10:

The AHB Master detects the combined HREADY signal high, and begins another transaction. This transfer is not to the EBI module.

The EBI module samples the transaction with address HADDR = A3, and registers the AHB address & control signals; the registered AHB address & control signals are used to generate the signaling that is presented to the external memory. The EBI also asserts HREADYOUT = 1’b0 to begin the necessary AHB wait states. Note that because the memory width (16-bits) is less than the transfer width (32-bits), the number of AHB wait states will be more than two. The EBI also drives extCeN, extBsN, and extWeN to enable the chip, enable a write, and enable the various byte lanes. extAddr is driven with {A3[N-1:2], 2'b00}, and the lower bits of HWDATA are routed to extWDATA.

Time T11:

Nothing of consequence occurs during this wait state.

Time T12:

The EBI drives extWeN = 1'b1, indicating to the memory that the write should be executed; HREADYOUT remains at 1'b0 because only 16 bits out of the 32-bit transfer will have completed by time T13.

Time T13:

The AHB Master does nothing of consequence here.

The EBI module drives extWeN = 1'b0 to start the second 16-bit transfer. extAddr is driven with {A3[N-1:2], 2'b10}, and the upper bits of HWDATA are routed to extWDATA.

The (asynchronous) external device has detected a low-to-high transition on extWeN sometime between T12 and T13 while extCeN = 1'b0. The write is executed to the memory upon the transition of extWeN.

Time T14:

Nothing of consequence occurs during this wait state.

Time T15:

The EBI drives HREADYOUT = 1'b1, indicating the end of the wait-stated transaction, and drives extWeN = 1'b1, indicating to the memory that the write should be executed.

Time T16:

The AHB Master samples HREADY = 1'b1, but does nothing of consequence here.

The EBI module drives extCeN = 1'b1, indicating the end of the transfer. (In the case of back-to-back transfers, the next transfer would be registered here, and extCeN would remain 1'b0.)

The (asynchronous) external device has detected a low-to-high transition on extWeN sometime between T15 and T16 while extCeN = 1'b0. The write is executed to the memory upon the transition of extWeN.

Time T17:

The AHB Master drives a new transfer on the AHB bus at address HADDR = A4. This transfer is a write to an external memory through the EBI. (The EBI has previously been configured for a single wait state on this memory.)

Time T18:

The AHB Master detects the combined HREADY signal high, and begins another transaction. This transfer is not to the EBI module.

The EBI module samples the transaction with address HADDR = A4, and registers the AHB address & control signals; the registered AHB address & control signals are used to generate the signaling that is presented to the external memory. The EBI also asserts HREADYOUT = 1'b0 to begin the necessary AHB wait states. Note that because the memory width (8-bits) is less than the transfer width (32-bits), the number of AHB wait states will be more than one. The EBI also drives extCeN, extBsN, and extWeN to enable the chip, enable a write, and enable the various byte lanes. extAddr is driven with {A3[N-1:2], 2'b00}, and the lower bits of HWDATA are routed to extWDATA.

Time T19:

The EBI drives extWeN = 1'b1, indicating to the memory that the write should be executed; HREADYOUT remains at 1'b0 because only 8 bits out of the 32-bit transfer will have completed by time T20.

Time T20:

The AHB Master does nothing of consequence here.

The EBI module drives extWeN = 1’b0 to start the second 8-bit transfer. extAddr is driven with {A3[N-1:2], 2’b01}, and the middle-lower bits of HWDATA are routed to extWDATA.

The (asynchronous) external device has detected a low-to-high transition on extWeN sometime between T19 and T20 while extCeN = 1’b0. The write is executed to the memory upon the transition of extWeN.

Time T21:

The EBI drives extWeN = 1’b1, indicating to the memory that the write should be executed; HREADYOUT remains at 1’b0 because only 16 bits out of the 32-bit transfer will have completed by time T22.

Time T22:

The AHB Master does nothing of consequence here.

The EBI module drives extWeN = 1’b0 to start the third 8-bit transfer. extAddr is driven with {A3[N-1:2], 2’b10}, and the middle-upper bits of HWDATA are routed to extWDATA.

The (asynchronous) external device has detected a low-to-high transition on extWeN sometime between T21 and T22 while extCeN = 1’b0. The write is executed to the memory upon the transition of extWeN.

Time T23:

The EBI drives extWeN = 1’b1, indicating to the memory that the write should be executed; HREADYOUT remains at 1’b0 because only 24 bits out of the 32-bit transfer will have completed by time T24.

Time T24:

The AHB Master does nothing of consequence here.

The EBI module drives extWeN = 1’b0 to start the fourth and final 8-bit transfer. extAddr is driven with {A3[N-1:2], 2’b11}, and the upper bits of HWDATA are routed to extWDATA.

The (asynchronous) external device has detected a low-to-high transition on extWeN sometime between T23 and T24 while extCeN = 1’b0. The write is executed to the memory upon the transition of extWeN.

Time T25:

The EBI drives HREADYOUT = 1’b1, indicating the end of the wait-stated transaction, and drives extWeN = 1’b1, indicating to the memory that the final 8-bit write should be executed.

Time T26:

The AHB Master samples HREADY = 1’b1, but does nothing of consequence here.

The EBI module drives extCeN = 1’b1, indicating the end of the transfer. (In the case of back-to-back transfers, the next transfer would be registered here, and extCeN would remain 1’b0.)

The (asynchronous) external device has detected a low-to-high transition on extWeN sometime between T25 and T26 while extCeN = 1’b0. The write is executed to the memory upon the transition of extWeN.

10. DMA-AHB Direct Memory Access Controller

10.1 Overview

The **IPC-DMA-AHB** is a configurable single channel direct memory access controller. The IPC-DMA-AHB IP Core is a Verilog HDL design that can be used in ASIC, Structured ASIC and FPGA designs. The design is intended to be used with AMBA based systems as a controller to transfer data directly from system memory to memory or system memory to peripheral device or IP Core.

Once set up, the IPC-DMA-AHB controller is primarily an AHB Master, which initiates data transfers across the AHB bus to/from a peripheral device through the DMA Buffer. The DMA Buffer is a scalable x32bit FIFO, which is useful for peripheral devices requiring a steady stream of data such as an LCD Controller, Ethernet MAC or other communication device.

The IPC-DMA-AHB controller contains useful features such incrementing and non-incrementing addressing and link list operation. Linked list support is useful for non contiguous memory transfer operations. Multiple DMA controllers can be placed in the AHB System to provide multiple channel DMA control.

10.2 Block Diagram

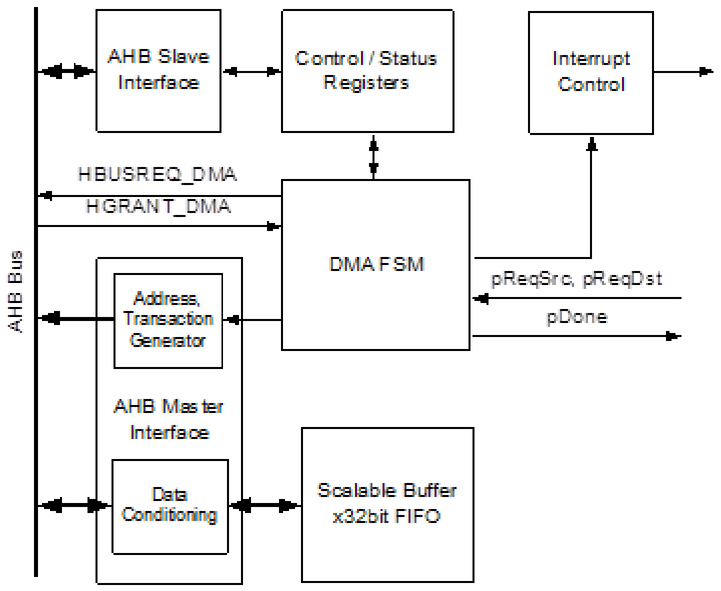


Figure 10-1: IPC-DMA-AHB Block Diagram

10.3 Functional Description

10.3.1 General

The **IPC-DMA-AHB** Controller is a totally synchronous design. All control timing is derived from the AHB clock input. The IPC-DMA-AHB has two AHB interfaces: an AHB Slave interface and an AHB Master interface. The AHB Slave interface provides access to the control/status registers of the IPC-DMA-AHB, and is used to program the specifics of the desired DMA transfer. Some of these specifics include source base address, destination base address, transfer length in bytes, and AHB bus control information. The SoC DMA Controller executes the DMA transfer via its AHB Master interface.

The DMA Controller serves as the flow controller for all transfers: from the source (memory or peripheral) to the destination (memory or peripheral). A DMA transfer to/from a peripheral is identical to a DMA transfer to/from a memory, with the exception of the handshaking signals **pReqX** and **pDone**. **pReqSrc** is a signal from the source peripheral to the DMA Controller that indicates that the peripheral is ready to start the transfer. **pReqDest** is a signal from the destination peripheral to the DMA Controller that indicates that the destination peripheral is ready to start the transfer. **pDone** is a signal from the DMA Controller to the peripheral(s) that indicates that the DMA Controller has completed the requested transfer. Memories are defined as AHB devices that do not use a handshaking mechanism.

A DMA transfer begins with the DMA Controller executing AHB reads from the source starting from the source base address. Data retrieved from the source is transferred to an internal FIFO. After the series of AHB reads, the DMA Controller reads the internal FIFO and executes a series of AHB writes to the destination peripheral starting with the destination base address. Reads and writes alternate in this fashion until the number of requested bytes has been transferred.

The DMA Controller supports a Linked-List operation mode, in which the DMA Controller uses its AHB Master interface to retrieve control information from a linked list element AHB memory. The DMA Controller uses the control information from the linked list element to execute the DMA transfer.

The DMA Controller is designed to interface with a microprocessor. The DMA Controller issues an interrupt upon linked-list element completion and/or DMA block transfer completion by asserting the blkInt signal. Block completion is determined by the address equaling the descriptor address + the transfer length.

10.3.2 AMBA (AHB Master / AHB Slave) Interfaces

The IPC-DMA-AHB Controller is connected to the microprocessor bus through the AMBA bus interface signals. The Control Registers are designed to be connected to an AMBA AHB Slave bus. The DMA transfer path is designed to be connected to an AHB Master; DMA transfers are executed by the DMA Controller via its AHB Master interface. For more information, refer to the AMBA 2.0 Bus Specification from ARM Ltd.

10.3.3 DMA Finite State Machine (FSM) and FIFO Control

After the IPC-DMA-AHB Controller has been programmed and enabled, its DMA Finite State Machine (FSM) transfers data from one AHB device to another AHB device through the following method:

Executes an AHB burst read, or a series of single reads, from the source device (initial read of DMA transfer is at the source base address). The exact number of bytes read is dependent on the source transfer width, the destination transfer width, and the number of bytes remaining in the transfer.

Captures the appropriate byte(s) of data from the AHB reads and writes the data into an internal 16x32 FIFO. In the case where the source width < destination width, some packing is done to the data in order to facilitate the transfer from the FIFO to the destination memory.

After the appropriate number of AHB reads has completed and their data stored in the internal FIFO, the FSM reads the internal FIFO and executes an AHB burst write, or a series of single writes, to the destination starting at the destination base address. The exact number of bytes written is dependent on the source transfer width, the destination transfer width, and the number of bytes remaining in the transfer.

If the correct number of bytes has been transferred, indicate to the register set that the DMA transfer is complete. If there are still more data to be transferred, the DMA Controller again executes step 1.

The DMA Controller automatically determines the number of AHB reads/writes from/to the source/destination. The user does not control the exact size of each AHB burst or the number of transfers associated with the AHB read or AHB write phase of the cycle; rather, the DMA Controller works this out based on the relative transfer widths (HSIZE) of source and destination, as well as the number of remaining transfers (which is an internal down-counter initialized to the value in the ***XferLength*** register). The following tables describe what happens in the different scenarios:

Table 10-1: Source and Destination with Same Transfer Width

Xfer Widths	S=32,D=32		S=16,D=16		S=8,D=8	
# Remaining Bytes	<64	>=64	<32	>=32	<16	>=16
# AHB Reads	#RB/4	16	#RB/2	16	#RB	16
# AHB Writes	#RB/4	16	#RB/2	16	#RB	16

Table 10-2: Source and Destination Different Same Transfer Widths

Xfer Widths	S=32,D=16		S=32,D=8		S=16,D=32	
# Remaining Bytes	<32	>=32	<16	>=16	<32	>=32
# AHB Reads	#RB/4	8	#RB/4	4	#RB/2	16

# AHB Writes	#RB/2	16	#RB	16	#RB/4	8
Xfer Widths	S=16,D=8		S=8,D=32		S=8,D=16	
# Remaining Bytes	<16	>=16	<16	>=16	<16	>=16
# AHB Reads	#RB/2	8	#RB	16	#RB	16
# AHB Writes	#RB	16	#RB/4	4	#RB/2	8

10.3.4 Memory to Memory Transfer

On a memory to memory transfer, data is read from the source memory using the source descriptor address. The data is read and loaded into the IPC-DMA-AHB 16x32Bit FIFO. If the FIFO is full or if the necessary data has been read from the source, the FSM will then write the data to the destination memory using the destination descriptor address. A maximum AHB burst value may be specified for the source/destination with the DMACtrl ***srcMaxBurst, dstMaxBurst*** bits. The DMActrl ***srcAutoIncrN*** and ***dstAutoIncrN*** bits should be cleared in order to automatically increment the source and destination addresses by 0x01, 0x02, or 0x04, depending on the source/destination transaction widths.

10.3.5 Memory to Peripheral Transfer

On a memory to peripheral transfer, data is read from the source memory using the source descriptor address. The data is read and loaded into the IPC-DMA-AHB 16x32Bit FIFO. If the FIFO is full or if the necessary data has been read from the source, the FSM will then write the data to the destination memory using the destination descriptor address.

The peripheral controls the transfer initiation by asserting the ***pReqSrc*** signal. A maximum AHB burst value may be specified for the source/destination with the DMACtrl ***srcMaxBurst, dstMaxBurst*** bits. The DMActrl ***srcAutoIncrN*** bit should be cleared to have the DMA Controller increment the source addresses by 0x01, 0x02, or 0x04, depending on the source transaction width. Depending on the register map of the peripheral, the DMActrl ***dstAutoIncrN*** bit may need to be set to prevent the destination address from incrementing.

10.3.6 Peripheral to Memory Transfer

On a peripheral to memory transfer, data is read from the source peripheral using the source descriptor address. The data is loaded into the IPC-DMA-AHB 16x32Bit FIFO. If the FIFO is full or if the necessary data has been read from the source, the FSM will then write the data to the destination memory using the destination descriptor address. The peripheral controls the transfer initiation by asserting the ***pReqSrc*** signal. A maximum AHB burst value may be specified for the source/destination with the DMACtrl ***srcMaxBurst, dstMaxBurst*** bits. Depending on the register map of the peripheral, the DMACtrl ***srcAutoIncrN*** bit may need to be set to prevent the source address from incrementing. DMACtrl ***dstAutoIncrN*** should be cleared to automatically increment the destination addresses by 0x01, 0x02, or 0x04, depending on the destination transaction width.

10.3.7 Peripheral to Peripheral Transfer

On a peripheral to peripheral transfer, data is read from the source peripheral using the source descriptor address. The data is loaded into the IPC-DMA-AHB 16x32Bit FIFO. If the FIFO is full or if the necessary data has been read from the source, the FSM will then write the data to the destination peripheral using the destination descriptor address. The peripherals control the transfer initiation by asserting the ***pReqSrc, pReqDst*** signals. A maximum AHB burst value may be specified for the source/destination with the DMACtrl ***srcMaxBurst, dstMaxBurst*** bits. Depending on the register map of the peripherals, the DMACtrl ***srcAutoIncrN, dstAutoIncrN*** bits may need to be set to prevent the source/destination address from incrementing.

10.3.8 Control and Status Registers

The Control and Status Registers are connected directly to a bus through an AHB Slave interface. The registers are selected from the processor via the HSEL block select signal. Individual registers are offset from the base address at 32-bit boundaries via the HADDR signals.

The control registers are used by the programmer to set up the specifics of the DMA transfer(s), including AHB bus control information such as HSIZE and HPROT, address incrementing, transfer type, source base address, destination base address, transfer length, and the desired interrupting conditions.

After the DMA control registers have been programmed, writing 0x00000001 to the Enable Register starts the DMA transfer.

Concerning the DMACtrl fields ***srcMaxBurst*** and ***dstMaxBurst,*** note that these fields do not control the number of AHB transactions that the DMA Controller executes in order to fill/empty its internal FIFO. Neither does it guarantee that the specified burst value will be used on the AHB bus to execute the read/write. Rather, these fields simply indicate to the DMA Controller the maximum AHB Burst size that will be used to or from the device. If the burst size specified in these fields is less than the optimal burst value that the DMA Controller would otherwise use, the DMA Controller simply will implement the necessary transfers as a series of non-sequential (single) transfers. For this reason, unless there is a compelling reason to limit the burst size, it is recommended that these fields be programmed with the default value indicating a 16-beat incrementing transfer.

The ***srcMaxBurst*** and ***dstMaxBurst*** fields are also used to specify a wrapping burst. Since there is no guarantee that a 4, 8, or 16-beat wrapping burst will actually be implemented on the AHB Bus, the DMA Controller simply uses these fields for wrapping the generated addresses at the appropriate boundary and implements the desired burst transfer as a series of non-sequential (single) transfers. Note that a DMA transfer involving a wrapping burst may be better specified as (at most) two separate DMA transfers.

Concerning the Source/Destination descriptors and the Transfer Length, there are two things that the programmer must be aware of:

It is the programmer’s responsibility to ensure that the Source Base Address is aligned with the value specified in the srcWidth subfield of the Control Register and the Destination Base Address is aligned with the value specified in the dstWidth subfield of the Control Register. For example, if srcWidth specifies 32-bit transfers, the Source Base Address must be on a 32-bit boundary, e.g. 32’h08000424. A Source Base Address of 32’h08000425 is illegal in this example, but would be legal if srcWidth instead specified 8-bit transfers.

It is the programmer’s responsibility to ensure that the Transfer Length is aligned with the value specified in the dstWidth subfield of the Control Register. For example, if dstWidth specifies 16-bit transfers, the Transfer Length (Bytes) field must specify a multiple of16-bits, e.g. 0x22. A Transfer Length of 0x21 is illegal in this example, but would be legal if dstWidth instead specified 8-bit transfers.

In cases of unaligned transfers from one 32-bit memory to another, it is recommended to implement these as 8-bit DMA transfers to and from the 32-bit devices. In some cases, it may also be possible to implement the bulk of the transfers as 32-bit transactions and the remainder as 8-bit transactions in two separate DMA transfers.

10.3.9 Interrupt Controller

Interrupts are generated from the following conditions, which are enabled under program control.

- Block Transfer Completed
- Source Peripheral HRESP_DMA = ERROR
- Destination Peripheral HRESP_DMA = ERROR
- LLI Element Transfer Completed

Interrupts are enabled by setting bits in the **IntrCtrl** register. The processor can determine the status of each interrupt by reading the **DMAStatus** register. The respective interrupt conditions may be cleared by writing a “1” to the appropriate bit(s) in the **IntrClear** register.

10.3.10 Peripheral Handshaking

There are two peripheral request signals to the DMA controller, **pReqSrc** and **pReqDst** for the source and destination peripherals respectively. Once the DMA is enabled, the source peripheral read transfer is enabled when **pReqSrc** is asserted. Likewise the destination peripheral write transfer is enabled when **pReqDst** is asserted. **pReqSrc** and **pReqDst** must remain asserted until the respective peripheral samples the handshake signal from the DMA controller, **pDone**, as a logic ‘1’. After sampling **pDone** as logic ‘1’, the peripheral must de-assert **pReqSrc/pReqDst** before another transfer can commence.

Because there is no mechanism to pause a transfer once it has begun, it is the responsibility of the programmer to determine the maximum atomic transfer size (in bytes) supported by each peripheral, and program the ***XferLength*** register of the DMA Controller accordingly. An additional mechanism exists to minimize system involvement in instances where DMA transfers occur to/from slow, low-volume peripherals. This mechanism is accessed by setting the ***repeatEnable*** bit in the ***DMACtrl*** register, along with the ***RepeatVal*** register. Setting the ***repeatEnable*** bit to a logic ‘1’, and programming the ***RepeatVal*** register enables the DMA Controller to execute the desired transfer RepeatVal+1 times. Each time the DMA transfer is repeated, the **pReqSrc/pReqDst/pDone** handshake is executed, but the **Block Transfer Complete** interrupt or the **LLI Element Transfer Complete** interrupt is not generated until after iteration number repeatVal+1. Note that this feature is available for DMA transfers of any length, and is not limited to peripherals. However, it is recommended that the repeat mechanism be used only in instances where hardware handshaking to/from a peripheral needs to pace the transfer of a large number of bytes. In such a case, the repeat mechanism accomplishes the desired behavior by essentially executing a series of small (e.g. ***XferLength*** < 16), handshaked transfers without involving the system. It should be noted that when ***repeatEnable*** is a logic ‘1’, the effective block transfer length in bytes is: ((***repeatVal***+1) * ***XferLength***).

Several peripherals may be connected to the DMA Controller simultaneously with a wired “AND” of the individual requests to **pReqSrc** and/***or pReqDst***, and by fanning out ***pDone*** to the several peripherals. The ***pDone*** signal is used to signal both the source peripheral and destination peripheral that the DMA transfer has completed.

10.3.11 Peripheral HRESP=ERROR response

If source peripheral issues a HRESP=ERROR response to a source read transaction, then the DMA controller will issue an interrupt if the ***IntrCtrl*** register ***bit 1*** is set to ***enable***.

If the destination peripheral issues a HRESP=ERROR response to a destination write transaction, then the DMA controller will issue an interrupt if the ***IntrCtrl*** register ***bit 2*** is set to ***enable***.

If the LLI memory issues a HRESP=ERROR response to an AHB Read or AHB Write, then the DMA controller will issue an interrupt if either the ***IntrCtrl*** register ***bit 1*** or ***bit 2*** is set to ***enable***. And the ***Status*** register will indicate an error on both the ***srcPerErr*** and ***dstPerErr*** bits.

In any of these cases, the DMA transfer is immediately aborted and **pDone** is asserted.

10.3.12 Peripheral HRESP=SPLIT and HRESP=RETRY responses

If the source or destination peripheral issues a HRESP=SPLIT or HRESP=RETRY response to a read or write transaction respectively, the AHB Master will regenerate the transaction to the peripheral that caused the SPLIT response. Because SPLIT and RETRY may occur at any time during a burst transaction, the DMA Controller will complete the original burst as a series of non-sequential (single) transfers.

The AHB arbiter is free to grant the AHB bus to other masters after a SPLIT response is detected. The transaction is completed when the AHB arbiter re-grants the AHB bus to the DMA controller and the peripheral has issued a HRESP=OKAY response to the DMA controller. There is no such requirement for the arbiter to de-grant for a RETRY response, but in either case the signaling for the AHB Master is the same.

10.3.13 Linked List Support

If the linkListEnable bit is set in the DMACtrl register, then the srcDescriptor register contains the base address of the initial element of a linked list in system memory. After the DMA Controller has been enabled in linked-list mode, the DMA Controller reads system memory at the address specified in the srcDescriptor register for the first list element.

Upon reading the Linked List memory, the DMA controller automatically updates the DMACtrl, SrcDescriptor, DestDescriptor, XferLength, and RepeatVal registers with the contents of the linked list element before executing the desired transfer. Other writes to these five registers while the DMA Controller is enabled in linked-list mode are strongly discouraged, as unpredictable results may occur.

Linked List Elements are always read as 8-beat incrementing bursts from a 32-bit memory. The Linked List Elements are organized as follows.

Table 10-3: Linked List Elements

Field	Width	Offset	R/W	Description
SrcAddress	31:0	0x00	R	Source Address Descriptor
DestAddress	31:0	0x04	R	Destination Address Descriptor
XferLengthBytes	31:0	0x08	R	Transfer Length in bytes
Control	31:0	0x0C	R	Control Information mirroring DMACtrl register; it is the programmer's responsibility to ensure the validity of each of the subfields, and to ensure that the linkedListEnable bit is set.
NextPtr	31:0	0x10	R	Pointer to Memory location containing the next Linked List Element. A value of 0xFFFFFFFF in this field indicates that this element is the final element in the linked list.
SharedStatus	31:0	0x14	R/W	Status element should be initialized to 32'd0 by programmer; this field is updated with 32'd1 once the element transfer is complete.
RepeatVal	31:0	0x18	R	This field mirrors the RepeatVal register. When the repeatEnable bit of the Control Field is 1, the RepeatVal field determines how many times this linked-list element is executed before the element transfer is complete, as well as how the source/destination address is incremented after each repeated transfer. When the repeatEnable bit of the Control Field is 0, this field is reserved but present.
Reserved	31:0	0x1C	R	Reserved but present

10.4 Register Descriptions

Table 10-4: : IPC-DMA-AHB Register Descriptions

Register	Bits	Offset Address	R/W	Bit Description
DMACtrl		0x00	R/W	Default = 0x0E200E20
srcAutoIncrN	31			1 = Do not increment the source address 0 = Increment the source address
srcWidth	30:28			Width of Source device in terms of the AHB signal HSIZE: 000 = 8-bit device 001 = 16-bit device 010 = 32-bit device All others reserved.
srcMaxBurst	27:25			Maximum Source Burst according to AHB signal HBURST: 000 = No bursting, single transfers only 001 = Reserved, not supported 010 = 4-beat Wrapping Burst 011 = 4-beat Incrementing Burst 100 = 8-beat Wrapping Burst 101 = 8-beat Incrementing Burst 110 = 16-beat Wrapping Burst 111 = 16-beat Incrementing Burst Note: Wrapping Bursts are implemented as single transfers. Note: If amount of data to be transferred to the internal FIFO is greater than the maximum source burst size, the transactions will be implemented as single transfers.

Register	Bits	Offset Address	R/W	Bit Description
srcProt	24:21			Source Protection coding according to AHB signal HPROT: HPROT[3]: 1 = cacheable, 0 = non-cacheable HPROT[2]: 1 = bufferable, 0 = non-bufferable HPROT[1]: 1 = Privileged Access, 0 = User Access HPROT[0]: 1 = Opcode Fetch, 0 = Data Access
dstAutoIncN	15			1 = Do not increment the destination address 0 = Increment the destination address
dstWidth	14:12			Width of Destination device in terms of the AHB parameter HSIZE: 000 = 8-bit device 001 = 16-bit device 010 = 32-bit device All others reserved.
dstMaxBurst	11:9			Maximum Destination Burst according to AHB signal HBURST: 000 = No bursting, single transfers 001 = Reserved, not supported 010 = 4-beat Wrapping Burst 011 = 4-beat Incrementing Burst 100 = 8-beat Wrapping Burst 101 = 8-beat Incrementing Burst 110 = 16-beat Wrapping Burst 111 = 16-beat Incrementing Burst Note: Wrapping Bursts are implemented as single transfers. Note: If amount of data to be transferred from the internal FIFO is greater than the maximum destination burst size, the transactions will be implemented as single transfers.
dstProt	8:5			Source Protection coding according to AHB signal HPROT: HPROT[3]: 1 = cacheable, 0 = non-cacheable HPROT[2]: 1 = bufferable, 0 = non-bufferable HPROT[1]: 1 = Privileged Access, 0 = User Access HPROT[0]: 1 = Opcode Fetch, 0 = Data Access
repeatEnable	3			1 = transfer of XferLength bytes is repeated according to the value in the RepeatVal register 0 = transfer is not repeated
mode	2:1			Modes: 00 = Source Memory to Destination Memory 01 = Source Memory to Destination Peripheral 10 = Source Peripheral to Destination Memory 11 = Source Peripheral to Destination Peripheral
linkListEnable	0			1 = enable the linked list pointers 0 = use registers; no linked lists
DMAStatus		0x04	RO	Default = 0x00000000
reserved	31:16			
lliElementCount	15:8			8-bit rollover counter that tracks the number of completed linked list transfers. This counter is reset upon a write to the Enable register.
enable	7			1 = enabled 0 = not enabled
reserved	6:4			
lliElementDone	3			1 = the DMA controller has processed at least one list element in LLI mode since the last DMAStatus Register read

Register	Bits	Offset Address	R/W	Bit Description
				0 = the DMA controller has not processed any list elements in LLI mode since the last DMAStatus Register read This bit is automatically cleared upon reading the DMAStatus Register
dstPerErr	2			Destination peripheral HRESP=ERROR; this bit is cleared upon writing a 0 to the Enable Register to disable the DMA controller operation.
srcPerErr	1			Source peripheral HRESP=ERROR; this bit is cleared upon writing a 0 to the Enable Register to disable the DMA controller operation.
xferDone	0			Transfer complete; this bit is cleared upon writing a 0 to the Enable Register to disable the DMA controller operation.
SrcDescriptor		0x08	R/W	Default = 0x00000000
	31:0			Source Address Descriptor Note: Must agree with srcWidth field of Control Register. Note: If linkListEnable = 1, this register is interpreted as the initial linked list pointer.
DestDescriptor		0x0C	R/W	Default = 0x00000000
	31:0			Destination Address Descriptor Note: Must agree with dstWidth field of Control Register.
XferLength		0x10	R/W	Default = 0x00000000
	31:0			Transfer length in bytes Note: Must agree with dstWidth field of Control Register.
CurSrcAddr		0x14	RO	Default = 0x00000000
	31:0			Current source address
CurDestAddr		0x18	RO	Default = 0x00000000
	31:0			Current destination address
IntrCtrl		0x1C	R/W	Default = 0x00000000
enaLliElementDone	3			Enable linked list element complete interrupt
enaDstPerErr	2			Enable Destination peripheral HRESP=ERROR interrupt
enaSrcPerErr	1			Enable Source peripheral HRESP=ERROR interrupt
enaXferDone	0			Enable Transfer complete interrupt
IntrClear		0x20	W1-CLR	Default = 0x00000000
clrLliElementDone	3			Clear linked list element complete interrupt
clrDstPerErr	2			Clear Destination peripheral HRESP=ERROR interrupt
clrSrcPerErr	1			Clear Source peripheral HRESP=ERROR interrupt
clrXferDone	0			Clear Transfer complete interrupt
Enable		0x24	R/W	Default = 0x00000000
	0			1 = DMA Enable 0 = DMA Disable
LliNextPtr		0x28	RO	Default = 0x00000000
	31:0			Displays the current LLI Next Pointer when DMA transfer is under link-list control (linkListEnable = 1). When link-list control is disabled, this register is read as 0.
RepeatVal		0x2C	R/W	Default = 0x00000000
rptSrcAddrIncrMult	28:24			Allowed values: 0, 1, 2, 4, 8, 16 If the repeatEnable bit (in the DMACtrl register) is 1, this

Register	Bits	Offset Address	R/W	Bit Description
				subfield determines how the source address is incremented after each iteration of the repeated transfer. rptSrcAddrIncrMult is right-shifted by srcWidth to get the increment value. For example, if srcWidth = 3'b001 and rptSrcAddrIncrMult = 16, the source address will be incremented by 32 after each iteration. A value of 0 in this subfield has the effect of not incrementing the base source address between iterations.
rptDstAddrIncrMult	20:16			Allowed values: 0, 1, 2, 4, 8, 16 If the repeatEnable bit (in the DMACtrl register) is 1, this subfield determines how the destination address is incremented after each iteration of the repeated transfer. rptDstAddrIncrMult is right-shifted by dstWidth to get the increment value. For example, if dstWidth = 3'b010 and rptDstAddrIncrMult = 4, the destination address will be incremented by 16 after each iteration. A value of 0 in this subfield has the effect of not incrementing the base destination address between iterations.
repeatVal	9:0			If the repeatEnable bit (in the DMACtrl register) is 1, this subfield determines the number of times a given DMA transfer of XferLength bytes is repeated. repeatVal=0 means that the requested transfer will execute once, and will not be repeated. A maximum value of 1023 means that the requested transfer will be executed 1024 times.
RepeatStatus		0x30	RO	Default = 0x00000000
	9:0			If the repeatEnable bit (in the DMACtrl register) is 1, this field represents a current status of the down-counter that keeps track of the number of times a DMA transfer is repeated. RepeatStatus will start with the value in RepeatVal and will be decremented by 1 upon completion of the transfer of XferLength bytes until RepeatStatus is 0. If the repeatEnable bit is 0, this field has no meaning and can be ignored.

10.5 Signal Descriptions

Table 10-5: IPC-DMA-AHB Interface Signals

Signal	Width	I/O	Active	Description
AHB Master Interface				
HCLK_DMA		I		AHB Clock; Must be the same as HCLK
HRESETn_DMA		I	Low	AHB Reset; Must be the same as HRESETn
HBUSREQ_DMA		O	High	AHB Bus Request as Master
HLOCK_DMA		O	High	AHB Lock transfer - Always Low. Locked transfers are not supported.
HADDR_DMA	[31:0]	O		AHB Address as Master to Arbiter
HWDATA_DMA	[31:0]	O		AHB Write Data
HWRITE_DMA		O	High	AHB Write Control
HTRANS_DMA	[1:0]	O		AHB Transfer Type
HSIZE_DMA	[2:0]	O		AHB Transfer Size
HBURST_DMA	[2:0]	O		AHB Burst
HPROT_DMA	[3:0]	O		AHB Protection - No protection required.
HRDATA_DMA	[31:0]	I		AHB Read Data
HRESP_DMA	[1:0]	I		AHB Response from Memory/Peripheral • HRESP = 00 - OKAY • HRESP = 01 - ERROR

Signal	Width	I/O	Active	Description
				• HRESP = 10 - RETRY • HRESP = 11 - SPLIT
HGRANT_DMA		I	High	AHB Bus Grant
HREADY_DMA		I	High	AHB Subsystem Combined Slave Ready
AHB Slave Interface				
HCLK		I		AHB Clock; Must be the same as HCLK_DMA
HRESETn		I	Low	AHB Reset; Must be the same as HRESETn
HADDR	[3:0]	I		AHB Address
HSEL		I	High	AHB Block Select
HSIZE	[2:0]	I		AHB Transfer Size; only 32-bit accesses to the register set are supported.
HWRITE		I	High	AHB Write Control
HTRANS	[1:0]	I		AHB Transfer Type
HWDATA	[31:0]	I		AHB Write Data Input from the processor
HREADY		I	High	AHB wait signal to stretch write cycles.
HRDATA	[31:0]	O		AHB Read Data to the processor
HREADYOUT		O	High	AHB Data ready output to AHB Subsystem – Always High.
HRESP	[1:0]	O		AHB Transfer response to AHB Master Always HRESP = 00 - OKAY
Peripheral Interface				
pReqSrc		I	High	Source peripheral transfer request (or ready to transfer).
pReqDest		I	High	Destination peripheral transfer request (or ready to transfer)
pDone		O	High	DMA Done flag to peripherals
Processor Interface				
blkInt		O	High	Interrupt Request

10.6 Timing Diagrams

10.6.1 AHB Master Interface Timing

The diagram below shows a typical DMA transaction where HGRANT_DMA remains with the DMA Controller’s AHB Master Interface for the duration of the burst read with 8 phases. Depending on the implementation of the AHB Arbiter, it is possible that HGRANT_DMA can be revoked at any phase of the burst transaction. If the burst is interrupted, after the DMA Controller’s AHB Master Interface regains HGRANT_DMA = 1 through arbitration, the signaling is the same as shown in the diagram except for two signals: HTRANS_DMA = NONSEQ and HBURST_DMA = 3’b000. HBUSREQ_DMA will be de-asserted when the final (8th) address is driven on the AHB bus.

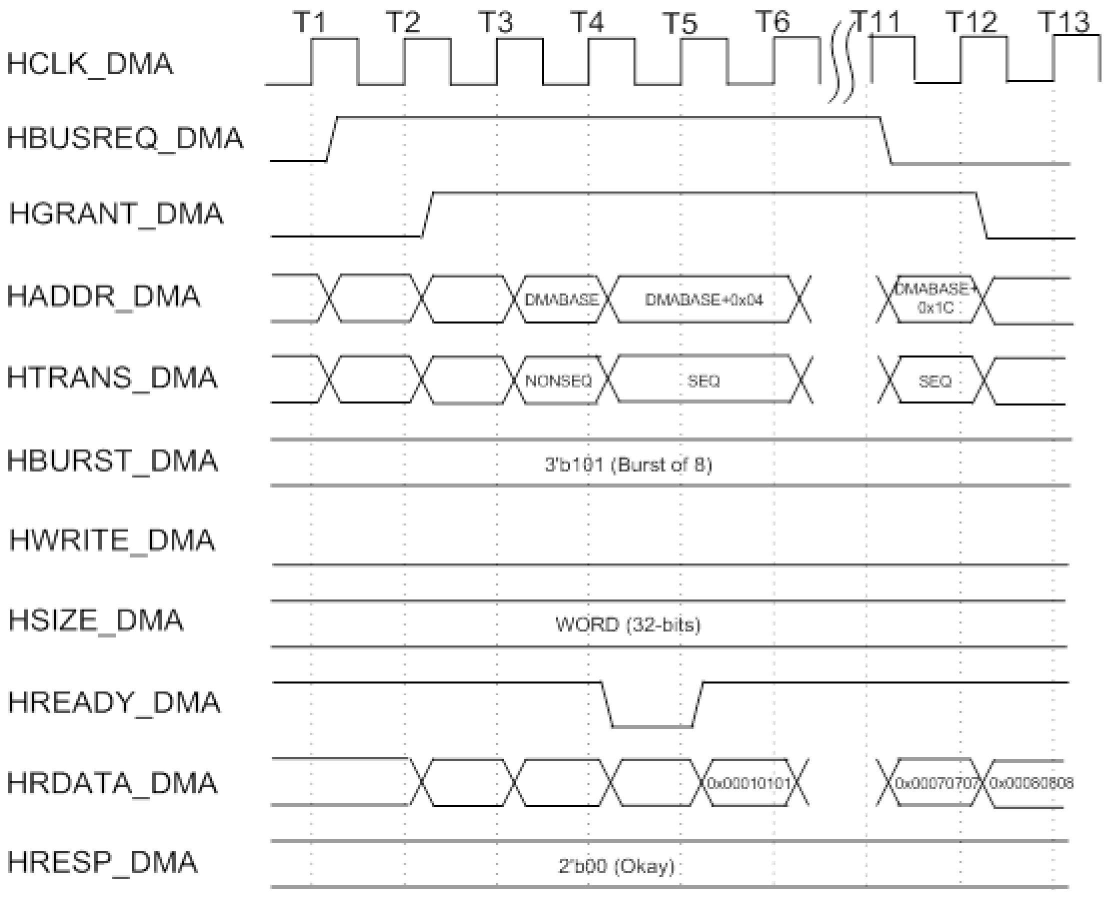


Figure 10-2: AHB Master Interface Timing

10.6.2 Peripheral Interface Timing

The diagram below shows the signaling between the peripheral and the DMA Controller. pReqSrc/pReqDest signals from the peripheral to the DMA Controller initiate the transfer; pDone is asserted by the DMA Controller upon completion of the DMA transfer. Once the peripheral has sampled pDone as a logic '1', the peripheral may de-assert pReqSrc/pReqDest. pDone will be de-asserted sometime after pReqSrc/pReqDest are sampled low. The peripheral must not re-assert pReqSrc/pReqDest while pDone is still high.

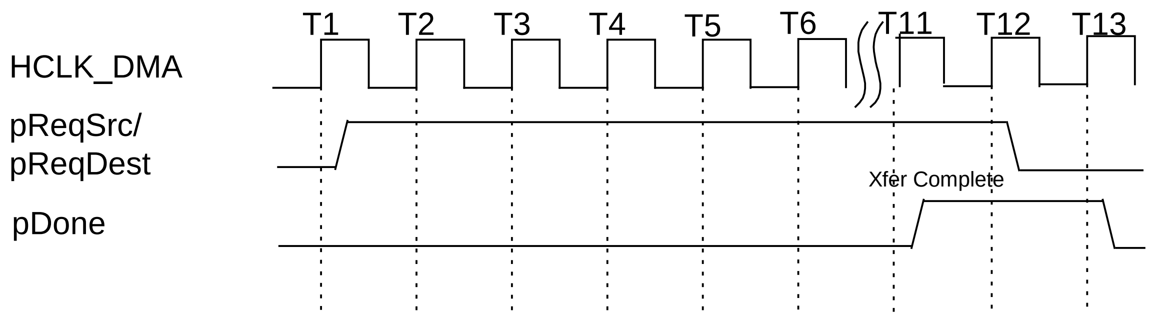


Figure 10-3: Peripheral Interface Timing

11. SSRAM Controller

11.1 Overview

The AHB SRAM Controller provides a standard AHB interface to translate AHB bus reads and writes into reads and writes with the signaling and timing of a standard 32-bit synchronous SRAM.

The AHB SRAM Controller provides zero-wait-state AHB access to the synchronous SRAM in all cases except for the following back-to-back events: an AHB write directly followed by an AHB read. In this case, a single wait state is asserted. Because of the zero-wait-state operation, this interface is intended to drive an on-chip memory (as opposed to off-chip memory where the return path for the read data might require wait states).

The AHB SRAM Controller contains two interfaces: an AHB Slave interface (for connecting to a Mirrored-Slave interface of an AHB Channel module), and a Memory interface (for connecting to a standard synchronous SRAM).

11.2 Block Diagram

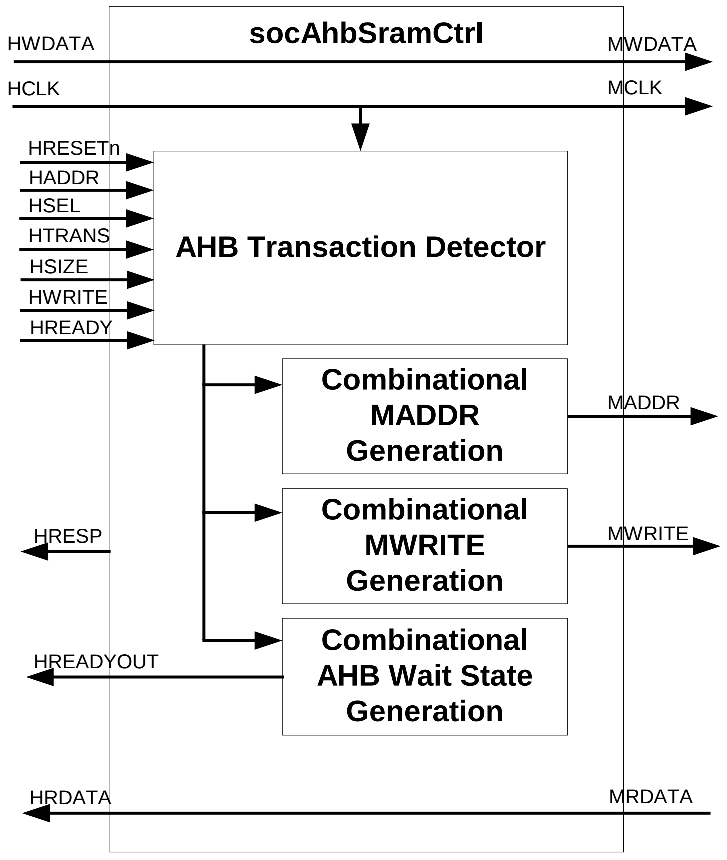


Figure 11-1: AHB SRAM Controller Block Diagram

11.3 Signal Descriptions

Table 11-1: AHB SRAM Controller Interface Signals

Signal Name	I/O	Description
AHB Slave Interface		
HRESETn	I	Active low system reset
HCLK	I	System clock
HWRITE	I	AHB write (0=read 1=write)
HWDATA[31:0]	I	AHB write data

Signal Name	I/O	Description
HADDR[31:0]	I	AHB Address bus
HSEL	I	AHB Block select, enables the interrupt controller
HTRANS[1:0]	I	AHB Transfer type
HSIZE[2:0]	I	AHB Data Size
HREADY	I	Combined AHB ready signal from all AHB slaves
HREADYOUT	O	Ready output to AHB
HRDATA[31:0]	O	AHB read data bus
HRESP[1:0]	O	AHB Slave response
Memory Interface		
MRDATA[31:0]	I	Memory read data bus
MWDATA[31:0]	O	Memory write data
MADDR[N:0]	O	Memory Address bus (size depends on amount of memory)
HWRITE[3:0]	O	Memory byte writes (0=read 1=write)
MCLK	O	Memory clock

11.4 Functional Description

The AHB SRAM Controller provides an interface between the AHB and the target Synchronous SRAM. The Synchronous SRAM should be either composed of 4 byte-wide RAMs or should be byte lane addressable.

Clock, write data, and read data are passed straight through this module with no modification (MCLK = HCLK, MWDATA = HWDATA, HRDATA = MRDATA).

The AHB SRAM Controller detects valid AHB transactions, and stores the necessary AHB transaction signals in a set of registers. Various combinations of the non-registered and the registered versions of HADDR, HTRANS, HSIZE, and HSEL are used to generate the different outputs of the module.

MADDR is either a registered or non-registered version of HADDR, depending on whether the transaction is a write (registered) or a read (non-registered). This is necessary because AHB bus is pipelined for reads and writes, while the Synchronous SRAM interface is effectively pipelined for reads only. Writes are not pipelined since address & write data are presented to the SRAM on the same clock cycle. An AHB read directly after one or more AHB writes necessitates the insertion of an AHB wait state during the data phase of the last AHB write. The wait state is necessary because the SRAM Controller needs a clock cycle to re-establish the pipeline for the upcoming read.

For any AHB write (HSEL = 1'b1 and HWRITE = 1'b1), the following table describes the generation of MWRITE based on the AHB signals HSIZE and HADDR.

Table 11-2: MWRITE Singal

HSIZE	HADDR[1:0]	MWRITE[3:0]
Single Byte Write		
BYTE	2'b00	4'b0001
BYTE	2'b01	4'b0010
BYTE	2'b10	4'b0100
BYTE	2'b11	4'b1000
Two-byte Write		
HWORD	2'b0x	4'b0011
HWORD	2'b1x	4'b1100
Four-byte Write		
WORD	2'bxx	4'b1111
No Write (HSEL = 1'b0 or HWRITE =		
X	2'bxx	4'b0000

Lastly, HRESP is assigned the constant value 2’b00 to indicate to the AHB Master that there was no problem with the requested AHB transfer.

11.5 Timing

The timing diagram below shows several standard AHB transactions. The example transactions show the following back-to-back AHB transactions: read, write; write, write; write, read. The relationships between these AHB transactions and their counterparts on the memory bus are also shown.

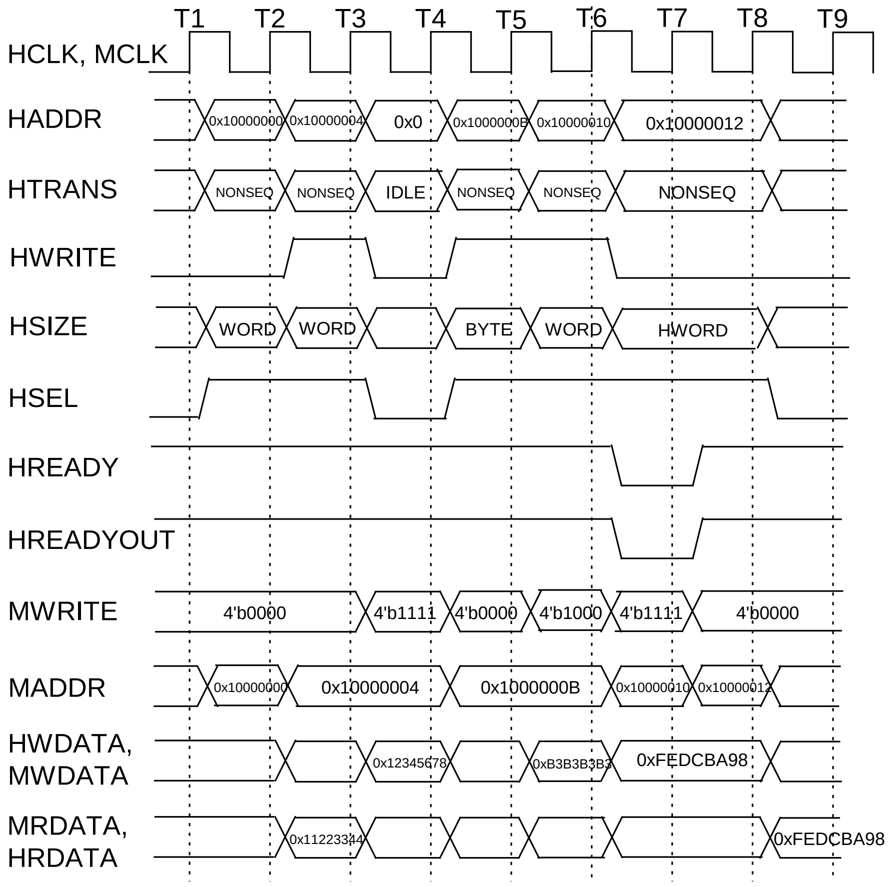


Figure 11-2: AHB SRAM Controller Timing

Time T1:

The AHB Master samples HREADY = 1’b1, and drives a new (NONSEQ) transfer on the AHB bus. The transfer is a 32-bit read at address 0x10000000. Because there are no wait states for this transaction, the read data will be sampled at time T3.

After the clock edge, the Address Decoder in the AHB Channel combinationally decodes HADDR, and asserts HSEL = 1’b1 to the SRAM Controller.

By default, the AHB SRAM controller assigns MWRITE = 4’b0000, and HADDR flows through to MADDR.

Time T2:

The AHB Master samples HREADY = 1’b1, and drives a new (NONSEQ) transfer on the AHB bus. The transfer is a 32-bit write to address 0x10000004.

After the clock edge, the Address Decoder in the AHB Channel combinationally decodes HADDR, and asserts HSEL = 1’b1 to the SRAM Controller.

The AHB SRAM Controller samples HREADY = 1’b1, HSEL = 1’b1, HTRANS = NONSEQ, and HWRITE = 1’b0. The SRAM Controller has detected a read transaction. The control signals for the transaction are registered, but HADDR is allowed to flow directly through to the memory, The memory samples HADDR = 0x10000000 and drives HRDATA with the stored value at that location (0x11223344).

Time T3:

The AHB Master drives an IDLE cycle on the bus. The AHB Master also drives HWDATA = 0x12345678 for the write transaction that was initiated at time T2. Lastly, the AHB Master samples HRDATA = 0x11223344.

After the clock edge, the Address Decoder in the AHB Channel combinationally decodes HADDR for the IDLE cycle; the result is HSEL = 1’b0 to the SRAM Controller.

The AHB SRAM Controller samples HREADY = 1’b1, HSEL = 1’b1, HTRANS = NONSEQ, and HWRITE = 1’b1. The SRAM Controller has detected a write. The control signals for the transaction are registered, and MADDR and MWRITE are driven based on these registered signals. Note that MWDATA, MADDR, and MWRITE are all driven at this time so that the memory may sample these signals altogether at time T4.

Time T4:

The AHB Master samples HREADY = 1’b1, and drives a new (NONSEQ) transfer on the AHB bus. The transfer is an 8-bit write to address 0x1000000B.

After the clock edge, the Address Decoder in the AHB Channel combinationally decodes HADDR, and asserts HSEL = 1’b1 to the SRAM Controller.

The AHB SRAM Controller samples HREADY = 1’b1, HSEL = 1’b0, HTRANS = IDLE, and HWRITE = 1’b0. The SRAM Controller has detected an IDLE transaction. HADDR is again allowed to flow directly through to the memory, and MWRITE is assigned its default value: 4’b0000.

The memory samples MADDR = 0x10000004, MWRITE = 4’b1111, and MWDATA = 0x12345678 and executes the 32-bit write.

Time T5:

The AHB Master samples HREADY = 1’b1, and drives a new (NONSEQ) transfer on the AHB bus. The transfer is a 32-bit write to address 0x10000010. The AHB Master also drives HWDATA = 0xB3B3B3B3 for the write transaction that was initiated at time T4.

After the clock edge, the Address Decoder in the AHB Channel combinationally decodes HADDR, and asserts HSEL = 1’b1 to the SRAM Controller.

The AHB SRAM Controller samples HREADY = 1’b1, HSEL = 1’b1, HTRANS = NONSEQ, and HWRITE = 1’b1. The SRAM Controller has detected a write. The control signals for the transaction are registered, and MADDR and MWRITE are driven based on these registered signals. Note that MWDATA, MADDR, and MWRITE are all driven at this time so that the memory may sample these signals altogether at time T6.

The memory samples MADDR = 0x1000000B, MWRITE = 4’b1000, and MWDATA = 0xB3B3B3B3 and executes the 8-bit write.

Time T6:

The AHB Master samples HREADY = 1’b1, and drives a new (NONSEQ) transfer on the AHB bus. The transfer is a 16-bit read from address 0x10000012. The AHB Master also drives HWDATA = 0xFEDCBA98 for the write transaction that was initiated at time T5.

After the clock edge, the Address Decoder in the AHB Channel combinationally decodes HADDR, and asserts HSEL = 1’b1 to the SRAM Controller.

The AHB SRAM Controller samples HREADY = 1’b1, HSEL = 1’b1, HTRANS = NONSEQ, and HWRITE = 1’b1. The SRAM Controller has detected a write. The control signals for the transaction are registered, and MADDR and MWRITE are driven based on these registered signals. Note that MWDATA, MADDR, and MWRITE are all driven at this time so that the memory may sample these signals altogether at time T7. ***** After the T6 clock edge, combinational logic determines that the next transaction is a read while the current transaction is a write. This causes HREADYOUT to be driven to 1’b0 for a single cycle. Asserting a wait state in the data phase of the write has the side effect of keeping

HADDR (and other AHB control signals) on the bus for an additional clock cycle. The SRAM Controller uses this additional clock cycle to rebuild the pipeline that is necessary for a read transaction. *****

The memory samples MADDR = 0x1000000B, MWRITE = 4’b1000, and MWDATA = 0xB3B3B3B3 and executes the 8-bit write.

Time T7:

The AHB Master Samples HREADY = 1’b0, and cannot drive a new transaction on the AHB bus; address and control signals from the read at time T6 remain.

The AHB Channel maintains HSEL = 1 to the SRAM Controller through its combinational decode of HADDR

The AHB SRAM Controller does not sample a new transaction since HREADY = 1’b0. However, HREADYOUT is driven to 1’b1 after the clock edge, MADDR is assigned HADDR, and MWRITE is driven with 4’b0000 in preparation for the upcoming read.

The memory samples MADDR = 0x10000010, MWRITE = 4’b1111, and MWDATA = 0xFEDCBA98 and executes the 32-bit write.

Time T8:

The AHB Master samples HREADY = 1’b1, and drives a transfer on the AHB bus.

After the clock edge, the Address Decoder in the AHB Channel combinationally decodes HADDR for the new transfer; the result is HSEL = 1’b0 to the SRAM Controller.

The AHB SRAM Controller samples HREADY = 1’b1, HSEL = 1’b1, HTRANS = NONSEQ, and HWRITE = 1’b0. The SRAM Controller has detected a read transaction. The control signals for the transaction are registered, but HADDR is allowed to flow directly through to the memory and MWRITE is driven with its default value: 4’b0000.

The memory samples HADDR = 0x10000012 and drives MRDATA with the stored value at that location (0xFEDCBA98).

Time T9:

The AHB Master samples the upper two bytes of HRDATA = 0xFEDCBA98 to complete the read initiated at time T6.

12. AHB to APB Bridge

12.1 Overview

The AHB to APB Bridge translates an AHB bus transaction (read or write) to an APB bus transaction. This is accomplished via a state machine.

The AHB to APB Bridge acts as an AHB Slave, and an APB Master in an AHB/APB subsystem. Typically, the AHB to APB Bridge has its AHB interface connected to a Slave port on an AHB Channel module, and its APB interface connected to the Master port on an APB Channel module.

12.2 Block Diagram

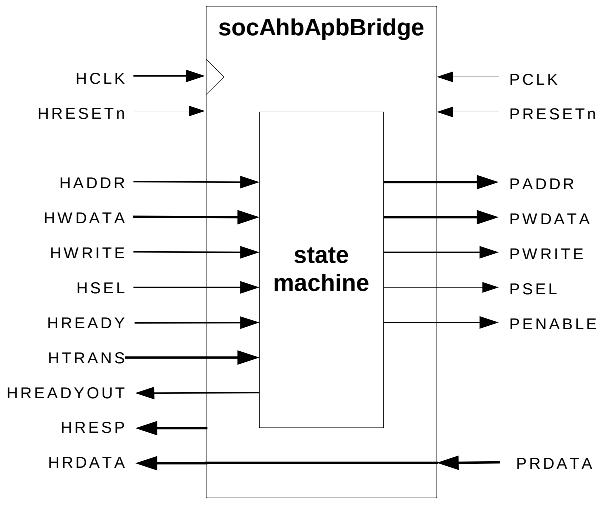


Figure 12-1 AHB to APB Bridge Block Digaram

12.3 Signal Descriptions

Table 12-1: AHB to APB Bridge Interface Signals

Signal Name	I/O	Description
AHB Interface		
HRESETn	I	Active low system reset
HCLK	I	System clock
HWRITE	I	AHB write (0=read 1=write)
HWDATA[31:0]	I	AHB write data
HADDR[31:0]	I	AHB Address bus
HSEL	I	AHB Block select, enables the interrupt controller
HTRANS[1:0]	I	AHB Transfer type
HSIZE[2:0]	I	AHB Data Size
HREADY	I	Combined AHB ready signal from all AHB slaves
HREADYOUT	O	Ready output to AHB
HRDATA[31:0]	O	AHB read data bus
HRESP[1:0]	O	AHB Slave response
APB Interface		
PRESETn	I	Active low APB reset (unused)
PCLK	I	APB clock (unused)
PRDATA[31:0]	I	Read data from the APB (32 bits)
PWRITE	O	APB Write (0=read 1=write)

PWDATA[31:0]	O	Write Data to the APB (32 bits)
PADDR[31:0]	O	APB Address bus (32 bits)
PSEL	O	APB Block select
PENABLE	O	APB signal indicating 2nd cycle of APB transfer

12.4 Functional Description

The AHB to APB Bridge contains a small state machine that implements the timing of the APB bus based on the timing of the AHB signals that it receives from the AHB Channel module. Because APB peripherals are not allowed to request wait states, the APB Bridge asserts a single wait state (HREADYOUT=0) back to the AHB bus in the 1st cycle of the AHB Data Phase. Read Data is sampled by the AHB master at the end of the next clock cycle.

The AHB to APB Bridge does not use the AHB Split/Retry mechanism; HRESP is assigned a constant 2'b00, indicating "OKAY" (a successful transfer).

The AHB bus and the APB bus must operate at the same frequency with the same clock (HCLK = PCLK).

The state machine drives the following registers that are the sources of the following APB signals: PADDR, PWRITE, PSEL, PENABLE, and PWDATA. Additionally, the state machine drives the register source for the AHB signal HREADYOUT.

The timing of the state machine is illustrated fully in a timing diagram which appears in another section of this document.

12.5 Timing

The AHB to APB Bridge implements the timing of the APB bus based on the timing of the AHB signals that it receives from the AHB Channel module. Because APB peripherals are not allowed to request wait states, the APB Bridge asserts a single wait state (HREADYOUT=0) back to the AHB bus in the 1st cycle of the AHB Data Phase. This is APB cycle C1. Data is sampled by the AHB master at the end of the next clock cycle, which is APB cycle C2.

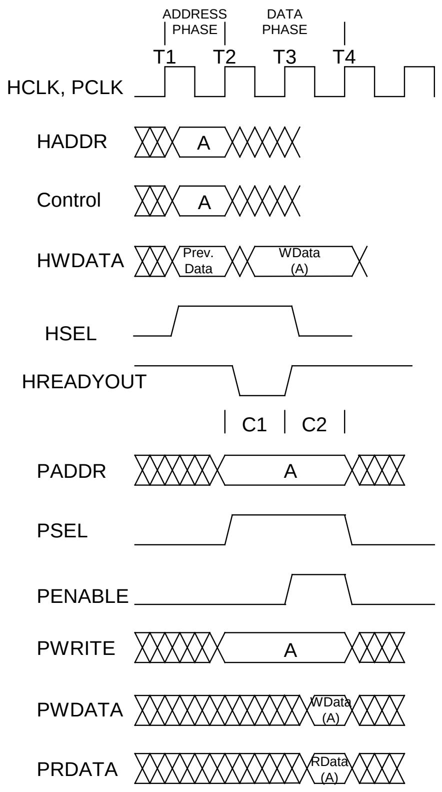


Figure 12-2: AHB to APB Bridge Interface Timing

Time T1:

The AHB master drives address and control onto the AHB bus for transaction ‘A’.

Time T2:

The AHB master drives HWDATA (if a write).

The AHB to APB bridge samples HSEL at a logic ‘1’, and latches the AHB Address and Control signals for transaction ‘A’. The bridge also drives HREADYOUT to logic level ‘0’ to assert the necessary wait state on the AHB bus. On the APB side, the bridge drives APB Address and Control signals (PADDR, PWRITE, PSEL) to initiate the corresponding APB transaction.

Time T3:

The AHB master samples HREADY at a logic level ‘0’, thus extending the data phase of AHB transaction ‘A’.

The AHB to APB bridge latches HWDATA and drives PWDATA (if a write), and also drives PENABLE to logic level ‘1’. The bridge drives HREADYOUT to logic level ‘1’, which signals the end of the one-cycle wait state on the AHB bus.

The target APB peripheral samples the APB Address and Control signals. If the APB transaction is a read, the APB peripheral must make the read data available before time T4, when it will be sampled by the AHB master.

Time T4:

The AHB master samples HREADY at a logic level ‘1’, which ends the data phase of the AHB transaction. If transaction ‘A’ is a read, the AHB master samples HRDATA at this time.

The AHB to APB bridge drives PSEL and PENABLE each to logic level ‘0’, which ends the APB transaction.

The APB peripheral samples PSEL and PENABLE each to a logic level ‘1’. If transaction ‘A’ is a write, the APB peripheral samples PWDATA at this time.

13. AHB Arbiter

13.1 Overview

The AHB Arbiter arbitrates for the AHB bus among as many as three AHB masters. The AHB Arbiter implements a round-robin arbitration algorithm to AHB Masters that are requesting use of the bus. Only one master may control a given phase of the AHB transaction at a given time.

AHB is a pipelined bus in which there are three distinct phases: Arbitration Phase, Address (or Control) Phase, and Data Phase. A consequence of pipelining is that these phases overlap in time. For example, it is possible that Master A wins the Arbitration Phase that is coincident with the Address Phase owned by Master B and the Data Phase owned by Master C. It is in these terms that an AHB system must be discussed.

The AHB Arbiter has three distinct Mirrored-Master Ports, each of which can be connected to an AHB Master (e.g. a processor or a DMA Controller), and one Port that connects to an AHB subsystem, typically through an AHB Channel module.

13.2 Block Diagram

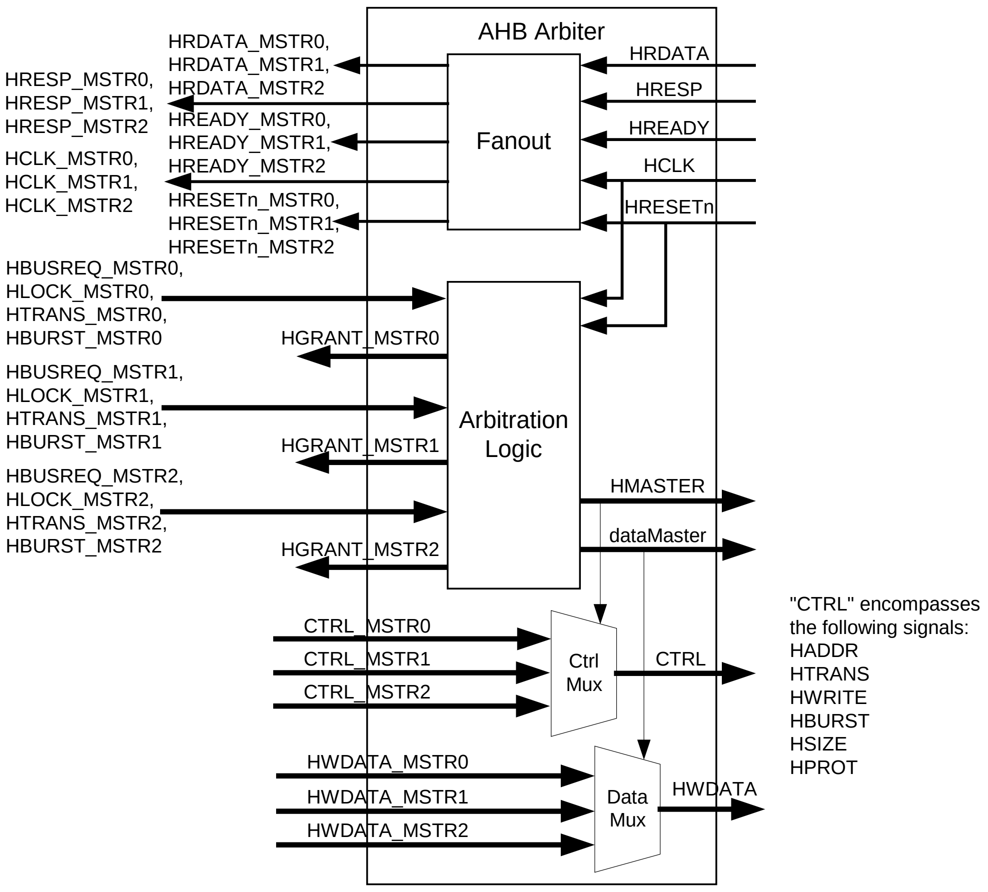

AHB Arbiter
HRDATA_MSTR0, HRDATA_MSTR1, HRDATA_MSTR2
HRESP_MSTR0, HRESP_MSTR1, HRESP_MSTR2
HREADY_MSTR0, HREADY_MSTR1, HREADY_MSTR2
HCLK_MSTR0, HCLK_MSTR1, HCLK_MSTR2
HRESETn_MSTR0, HRESETn_MSTR1, HRESETn_MSTR2
Fanout
HRDATA
HRESP
HREADY
HCLK
HRESETn
HBUSREQ_MSTR0, HLOCK_MSTR0, HTRANS_MSTR0, HBURST_MSTR0
HGRANT_MSTR0
HBUSREQ_MSTR1, HLOCK_MSTR1, HTRANS_MSTR1, HBURST_MSTR1
HGRANT_MSTR1
HBUSREQ_MSTR2, HLOCK_MSTR2, HTRANS_MSTR2, HBURST_MSTR2
HGRANT_MSTR2
Arbitration Logic
HMASTER
dataMaster
CTRL_MSTR0
CTRL_MSTR1
CTRL_MSTR2
Ctrl Mux
CTRL
HWDATA_MSTR0
HWDATA_MSTR1
HWDATA_MSTR2
Data Mux
HWDATA
"CTRL" encompasses the following signals: HADDR HTRANS HWRITE HBURST HSIZE HPROT

Figure 13-1: AHB Arbiter Block Diagram

13.3 Signal Descriptions

Table 13-1: AHB Arbiter Interface SIgnals

Signal Name	I/O	Description
Miscellaneous Signals		
HMASTER[3:0]	O	Status of which Master controls Transaction Address Phase
dataMaster[3:0]	O	Status of which Master controls Transaction Data Phase

Signal Name	I/O	Description
AHB Subsystem Interface		
HRESETn	I	Active low system reset
HCLK	I	System clock
HWRITE	O	AHB write (0=read 1=write)
HWDATA[31:0]	O	AHB write data
HADDR[31:0]	O	AHB Address bus
HBURST[2:0]	O	AHB Block select, enables the interrupt controller
HTRANS[1:0]	O	AHB Transfer type
HSIZE[2:0]	O	AHB Data Size
HPROT[3:0]	O	AHB Protection
HREADY	I	Combined AHB ready signal from all AHB slaves
HRDATA[31:0]	I	AHB read data bus
HRESP[1:0]	I	AHB Slave response
AHB Master 0 Interface		
HCLK_MSTR0	O	Active low system reset
HRESETn_MSTR0	O	System clock
HADDR_MSTR0[31:0]	I	AHB Address bus
HWDATA_MSTR0[31:0]	I	AHB Write Data
HSIZE_MSTR0[2:0]	I	AHB Data Size
HBURST_MSTR0[2:0]	I	AHB Burst
HPROT_MSTR0[3:0]	I	AHB Protection
HTRANS_MSTR0[1:0]	I	AHB Transfer Type
HWRITE_MSTR0	I	AHB Write (0=Read, 1=Write)
HBUSREQ_MSTR0	I	AHB Bus Request
HLOCK_MSTR0	I	AHB Locked Transfer
HGRANT_MSTR0	O	AHB Bus Grant
HREADY_MSTR0	O	Combined AHB ready signal from all AHB slaves
HRESP_MSTR0[1:0]	O	AHB Slave Response
HRDATA_MSTR0[31:0]	O	AHB Read Data
AHB Master 1 Interface		
HCLK_MSTR1	O	Active low system reset
HRESETn_MSTR1	O	System clock
HADDR_MSTR1[31:0]	I	AHB Address bus
HWDATA_MSTR1[31:0]	I	AHB Write Data
HSIZE_MSTR1[2:0]	I	AHB Data Size
HBURST_MSTR1[2:0]	I	AHB Burst
HPROT_MSTR1[3:0]	I	AHB Protection
HTRANS_MSTR1[1:0]	I	AHB Transfer Type
HWRITE_MSTR1	I	AHB Write (0=Read, 1=Write)
HBUSREQ_MSTR1	I	AHB Bus Request
HLOCK_MSTR1	I	AHB Locked Transfer
HGRANT_MSTR1	O	AHB Bus Grant
HREADY_MSTR1	O	Combined AHB ready signal from all AHB slaves
HRESP_MSTR1[1:0]	O	AHB Slave Response
HRDATA_MSTR1[31:0]	O	AHB Read Data
AHB Master 2 Interface		

Signal Name	I/O	Description
HCLK_MSTR2	O	Active low system reset
HRESETn_MSTR2	O	System clock
HADDR_MSTR2[31:0]	I	AHB Address bus
HWDATA_MSTR2[31:0]	I	AHB Write Data
HSIZE_MSTR2[2:0]	I	AHB Data Size
HBURST_MSTR2[2:0]	I	AHB Burst
HPROT_MSTR2[3:0]	I	AHB Protection
HTRANS_MSTR2[1:0]	I	AHB Transfer Type
HWRITE_MSTR2	I	AHB Write (0=Read, 1=Write)
HBUSREQ_MSTR2	I	AHB Bus Request
HLOCK_MSTR2	I	AHB Locked Transfer
HGRANT_MSTR2	O	AHB Bus Grant
HREADY_MSTR2	O	Combined AHB ready signal from all AHB slaves
HRESP_MSTR2[1:0]	O	AHB Slave Response
HRDATA_MSTR2[31:0]	O	AHB Read Data

13.4 Functional Description

The AHB Arbiter arbitrates for use of the AHB Subsystem among as many as three AHB Masters. AHB defines the arbitration mechanism as follows. An AHB Master requests use of the AHB Subsystem by asserting HBUSREQ_MSTRx = 1’b1. The Arbiter grants use of the AHB Subsystem by asserting HGRANT_MSTRx = 1’b1 back to the requesting AHB Master.

The AHB Arbiter implements a round-robin arbitration scheme in which all Masters have equal priority. An AHB Master will not be granted use of the AHB Subsystem without having requested use of the bus. The only exception to this rule may occur with AHB Master 0; AHB Master 0 is treated as the default Master if no other AHB Masters are requesting the bus.

The AHB Arbiter has three main functions.

To determine the next AHB Master to be granted access to the AHB Subsystem.

To keep track of which AHB Master currently has control of the AHB Subsystem’s Address (or Control) Phase, and Data Phase.

To distribute, or fan out, various signals from the AHB Subsystem to the various AHB Masters.

13.4.1 Determining the Next AHB Master

As mentioned earlier, a round-robin scheme among requesting Masters determines the order of the granted Masters. For example, if the current master is Master 0, then Master 1 will be given first priority at the next changeover; if the current Master is Master 1, then Master 2 will be given priority at the next changeover; if the current Master is Master 2, then Master 0 will be given priority at the next changeover. A new Master may be granted at when all of the following are true:

HREADY = 1’b1 from the AHB Subsystem; The AHB Arbiter will not change Masters when HREADY is low.

HLOCK_MSTRx = 1’b1 from the currently granted Master; The AHB Arbiter will not change Masters in the middle of a locked transfer.

The currently granted Master starts a new transfer with HTRANS_MSTRx indicating NONSEQ, BUSY, or IDLE, ***AND*** the currently granted Master has owned the AHB Subsystem for at least two AHB Bus Cycles.

Note that the Arbiter may terminate a burst early (EBT) if the new NONSEQ transfer is a burst transfer. Note also that a newly granted Master will be allowed to start and complete a burst transfer. The lone exception to this rule defining initial burst completion is that the AHB “Incrementing Burst of Unspecified

Length" is not treated as a burst transfer by the AHB Arbiter. As such, this type of burst is not guaranteed to complete, even if it is the initial transaction of a newly granted Master.

13.4.2 Keeping Track of Address Phase and Data Phase

The AHB Arbiter also must keep track of the Master that is granted the current Control Phase and Data Phase present on the AHB Subsystem. The Arbitration logic produces two signals, HMASTER and dataMaster, which accomplish this goal. HMASTER controls the output of the AHB Address (or Control) Mux, and dataMaster controls the output of the AHB Data Mux. The Master that controls the current Control Phase will always control the next Data Phase. The Control Phase and the Data Phase for a given transfer always occur on successive AHB Bus Cycles.

13.4.3 Distribution (Fanout)

The unmodified signals that are distributed from the AHB Subsystem to the various AHB Masters are: HCLK, HRESETn, HREADY, HRESP, and HRDATA. This is done to facilitate the connection of multiple AHB Masters to the AHB Arbiter module.

13.5 Timing

The diagram below shows an example bus changeover from Master 0 to Master 1. Note that multiple masters may be requesting the bus, but that only one Master will have its HGRANT_MSTRx asserted at any given time. Note also that because AHB is a pipelined bus, the changeover happens in stages also, corresponding to the Address Phase and Data Phase of AHB.

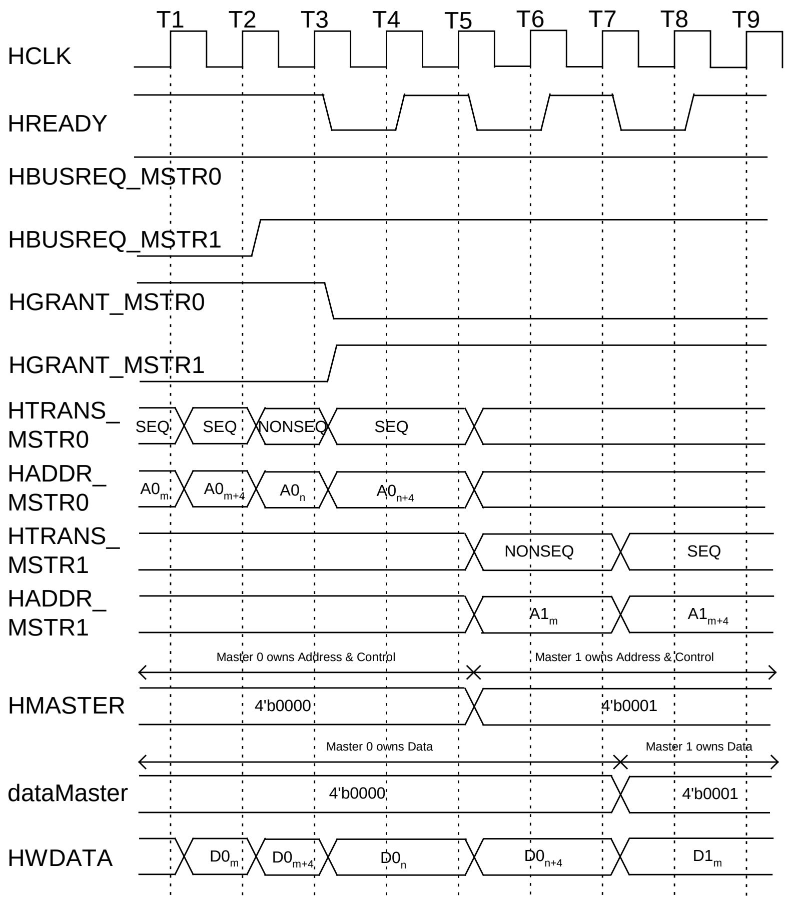


Figure 13-2: AHB Bus Changeover from Master 0 to Master 1

Time T1:

The Arbiter has granted AHB Master 0 use of the bus.

AHB Master 0 samples HREADY = 1'b1, and drives the final address phase for the burst transfer.

Time T2:

The Arbiter has granted AHB Master 0 use of the bus.

AHB Master 0 samples HREADY = 1'b1, and starts a new transaction (HADDR = A0n) by driving HTRANS_MSTR0 = NONSEQ.

AHB Master 1 begins to drive HBUSREQ_MSTR1 = 1'b1.

Time T3:

The AHB Arbiter samples HBUSREQ_MSTR1 = 1'b1, and detects the start of a new transfer for the currently granted master (HTRANS_MSTR0 = NONSEQ). This triggers a bus changeover event; the arbiter drives HGRANT_MSTR0 = 1'b0, and drives HGRANT_MSTR1 = 1'b1.

AHB Master 0 samples its HGRANT_MSTR0 = 1'b1 and HREADY = 1'b1, and drives the next transaction.

AHB Master 1 samples its HGRANT_MSTR1 = 1'b0, so it must wait to drive Address & Control signals.

The AHB Subsystem samples HADDR = A0n and inserts a wait state by driving HREADY = 1'b0.

Time T4:

HREADY is low at this time, indicating that the AHB subsystem has requested a wait state to finish the data phase for the transaction with address A0n. The AHB Subsystem drives HREADY = 1'b1 again.

Time T5:

The AHB Arbiter drives HMASTER to 4'b0001, indicating Master 1 has control of the Address phase of the AHB bus.

AHB Master 0 samples its HGRANT_MSTR0 = 1'b0 and HREADY = 1'b1, and cannot start any more transactions. Note that the burst transfer characterized by A0n and A0n+4 is terminated early by the AHB Arbiter (EBT). In general, EBT is an unavoidable consequence of AHB arbitration. However, AHB Master 0 still owns the data phase of the AHB bus here, and drives D0n+4.

AHB Master 1 samples its HGRANT_MSTR1 = 1'b1 and HREADY = 1'b1, so it drives Address & Control signals for transaction A1m.

The AHB Subsystem samples HADDR = A0n+4, and inserts a wait state by driving HREADY = 1'b0. The AHB Subsystem also samples HWDATA = D0n thus completing that transaction.

Time T6:

HREADY is low at this time, indicating that the AHB subsystem has requested a wait state to finish the data phase for the transaction with address A0n+4. The AHB Subsystem drives HREADY = 1'b1 again.

Time T7:

The AHB Arbiter drives dataMaster = 4'b0001, indicating that Master 1 has control of the data phase of the AHB bus.

AHB Master 0 samples its HGRANT_MSTR0 = 1'b0 and HREADY = 1'b1, and cannot start any more transactions. AHB Master 0 no longer owns the data phase of the AHB bus here, and cannot drive HWDATA.

AHB Master 1 samples its HGRANT_MSTR1 = 1'b1 and HREADY = 1'b1, so it drives Address & Control signals for transaction A1m+4.

The AHB Subsystem samples HADDR = A1m, and inserts a wait state by driving HREADY = 1'b0. The AHB Subsystem also samples HWDATA = D0n+4 thus completing that transaction.

Time T8:

HREADY is low at this time, indicating that the AHB subsystem has requested a wait state to finish the data phase for the transaction with address A1m. The AHB Subsystem drives HREADY = 1'b1 again.

Time T9:

The AHB Arbiter has granted Master 1 use of the bus.

AHB Master 0 samples its HGRANT_MSTR0 = 1’b0 and HREADY = 1’b1, and cannot start any more transactions. AHB Master 0 no longer owns either phase of the AHB bus.

AHB Master 1 samples its HGRANT_MSTR1 = 1’b1 and HREADY = 1’b1, so it drives Address & Control signals for transaction A1m+4.

The AHB Subsystem samples HWDATA = D1m thus completing that transaction.