PrimeCell™ DDR2 Dynamic Memory Controller (PL341)

Revision: r0p0

Technical Reference Manual

ARM®

ARM DDI 0418A

PrimeCell DDR2 Dynamic Memory Controller (PL341)
Technical Reference Manual

Release Information

Change history

Date	Issue	Confidentiality	Change
29 March 2007	A	Non-Confidential	First release for r0p0

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
PrimeCell DDR2 Dynamic Memory Controller (PL341) Technical Reference Manual

List of Tables
PrimeCell DDR2 Dynamic Memory Controller (PL341) Technical Reference Manual

List of Figures
PrimeCell DDR2 Dynamic Memory Controller (PL341) Technical Reference Manual

Preface

This preface introduces the *PrimeCell AXI DDR2 Dynamic Memory Controller (PL341) Revision r0p0 Technical Reference Manual*. It contains the following sections:

About this manual

This is the Technical Reference Manual for the PrimeCell AXI *DDR2 Dynamic Memory Controller* (DDR2 DMC). The PrimeCell DDR2 DMC comprises various systems that you can render to support a single type and size of SDRAM on the memory interface.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this manual, where:

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

Intended audience

This manual is written for implementation engineers and architects, and provides a description of an optimal DDR2 DMC architecture. The controller provides an interface between the *Advanced eXtensible Interface* (AXI) system bus and external, off-chip, memory devices.

Using this manual

This manual is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the controller and its features.

Chapter 2 *Functional Overview*

Read this chapter for an overview of the major functional blocks and the operation of the controller.

Chapter 3 *Programmer's Model*

Read this chapter for a description of the registers.

Chapter 4 *Programmer's Model for Test*

Read this chapter for a description of the additional logic for integration testing.

Appendix A *Signal Descriptions*

Read this appendix for a description of the signals.

Glossary

Read the Glossary for definitions of terms used in this manual.

Conventions

Conventions that this manual can use are described in:

- *Typographical*
- *Timing diagrams* on page xii
- *Signals* on page xii
- *Numbering* on page xiii.

Typographical

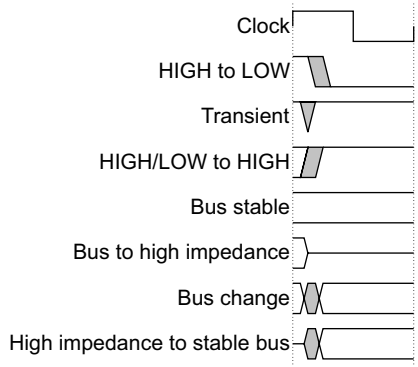
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example: • `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>` • The Opcode_2 value selects which register is accessed.

Timing diagrams

The figure named *Key to timing diagram conventions* explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



Key to timing diagram conventions

Signals

The signal conventions are:

Signal level The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.

Lower-case n Denotes an active-LOW signal.

Prefix A Denotes global *Advanced eXtensible Interface* (AXI) signals:

Prefix AR Denotes AXI read address channel signals.

Prefix AW Denotes AXI write address channel signals.

Prefix B Denotes AXI write response channel signals.

Prefix C Denotes AXI low-power interface signals.

Prefix H Denotes *Advanced High-performance Bus* (AHB) signals.

Prefix P Denotes Advanced Peripheral Bus (APB) signals.

Prefix R Denotes AXI read data channel signals.

Prefix W Denotes AXI write data channel signals.

Numbering

The numbering convention is:

<size in bits>'<base><number>

This is a Verilog method of abbreviating constant numbers. For example:

- 'h7B4 is an unsized hexadecimal value.
- 'o7654 is an unsized octal value.
- 8'd9 is an eight-bit wide decimal value of 9.
- 8'h3F is an eight-bit wide hexadecimal value of `0x3F`. This is equivalent to b00111111.
- 8'b1111 is an eight-bit wide binary value of b00001111.

Further reading

This section lists publications by ARM.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets, addenda, and the Frequently Asked Questions list.

ARM publications

This manual contains information that is specific to the DDR2 DMC. See the following documents for other relevant information:

- *PrimeCell AXI DDR2 Dynamic Memory Controller (PL341) Integration Manual* (ARM DII 0184)
- *PrimeCell AXI DDR2 Dynamic Memory Controller (PL341) Implementation Guide* (ARM DII 0183)
- *AMBA Designer (FD001) User Guide* (ARM DUI 0333)
- *AMBA AXI Protocol Specification* (ARM IHI 0022)
- *AMBA 3 APB Protocol Specification* (ARM IHI 0024).

Feedback

ARM welcomes feedback on the DDR2 DMC and its documentation.

Feedback on this DDR2 DMC

If you have any comments or suggestions about this product, contact your supplier giving:

- the product name
- a concise explanation of your comments.

Feedback on this manual

If you have any comments on this manual, send e-mail to `errata@arm.com` giving:

- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the memory controller and contains the following sections:

1.1 About the PrimeCell AXI DDR2 DMC (PL341)

The DDR2 DMC is an *Advanced Microcontroller Bus Architecture* (AMBA) compliant *System-on-Chip* (SoC) peripheral that is developed, tested, and licensed by ARM.

The device is a high-performance, area-optimized DDR2 SDRAM memory controller compatible with the AMBA AXI protocol.

You can configure the memory controller with a number of options:

- the AXI and SDRAM memory data width
- the number of SDRAM memory devices
- the number of outstanding AXI addresses.

For more information on AMBA see:

- *AMBA AXI Protocol Specification*
- *AMBA 3 APB Protocol Specification*.

Figure 1-1 shows an example system.

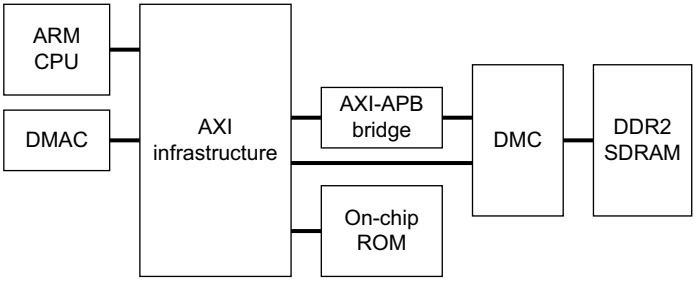


Figure 1-1 Example system

1.1.1 Features of the device

The memory controller block has the following features:

- compatible with AMBA AXI for accesses to DDR2 SDRAM
- compatible with AMBA APB for programming the device
- soft macrocell available in Verilog
- synchronous n:1 clocking between AXI and APB
- synchronous n:1 and 1:n operation between AXI bus infrastructure and external memory bus

- asynchronous operation between AXI bus infrastructure and external memory bus
- active and precharge power-down supported in the SDRAM
- *Quality of Service* (QoS) features for low latency transfers
- optimized utilization of external memory bus
- programmable selection of external memory width, see *Supported memory widths*
- multiple outstanding transactions
- write data interleaving supported
- hardware resource that can be rendered to optimize area versus performance
- support for a configurable number of ARMv6 architecture outstanding exclusive access transfers
- each controller can be configured to have between 1 and 4 memory chip selects.

1.1.2 Supported memory widths

In each case, the write data FIFO width is equal to the AXI data width. The read data FIFO depth is equal to the greater of the AXI or the effective memory data width.

Table 1-1 lists the supported combinations of memory and AXI bus widths.

Table 1-1 Supported combinations of memory and AXI bus widths

Combination	Memory data width	Effective memory data width[a]	AXI data width
a	16-bit	32-bit	32-bit
b	16-bit	32-bit	64-bit
c	32-bit	64-bit	32-bit
d	32-bit	64-bit	64-bit
e	32-bit	64-bit	128-bit
f	64-bit	128-bit	64-bit
g	64-bit	128-bit	128-bit

a. Effective memory data width is equal to the size of transfer on a per-cycle basis on the memory interface.

Note

In addition to the choice of memory data widths at render time, you can select to use the controller with half the configured memory width. You can configure the controller either:

- statically, using a tie-off
- by programming, using the APB interface.

The restrictions on this are as follows:

You can use each controller configuration at half of its memory width, by programming the memory_width via the APB interface, provided that:

- the new memory width is not less than 16 bits
- the effective memory width is not less than half the AXI interface width.

The APB interface implements the register file as Chapter 3 *Programmer's Model* describes. For detailed information about the APB interface, see the *AMBA 3 APB Protocol Specification*.

1.1.3 AXI interface attributes

The slave interface has the following attributes:

Combined Acceptance Capability

The maximum number of active transactions that a slave can accept. This is indicated where the slave has combined read and write transaction storage so this is mutually exclusive to the write or read acceptance capability.

Write Interleave Depth

The number of active write transactions for which the slave can receive data. This is counted from the earliest transaction.

Read Data Reordering Depth

The number of active read transactions for which a slave can transmit data. This is counted from the earliest transaction.

Table 1-2 lists the attribute formats.

Table 1-2 Attribute formats

Attribute	Value
Combined acceptance capability	Arbiter queue depth
Write interleave depth	Combined acceptance capability
Read data reorder depth	Combined acceptance capability

1.2 Supported devices

See the product *Release Note* for more information.

Chapter 2
Functional Overview

This chapter describes the DDR2 DMC operation. It contains the following sections:

- *Functional description* on page 2-2
- *Functional operation* on page 2-9.

2.1 Functional description

Figure 2-1 shows a block diagram of the device.

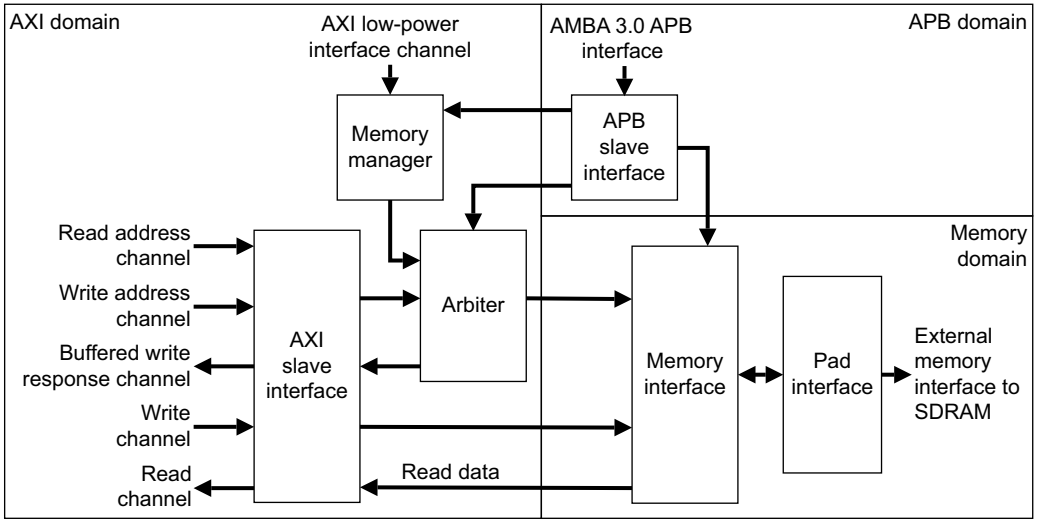


Figure 2-1 Block diagram

The main blocks in the memory controller design are:

- *AXI slave interface*
- *Arbiter* on page 2-5
- *Memory manager* on page 2-5
- *APB slave interface* on page 2-6
- *Memory interface* on page 2-7
- *Pad interface* on page 2-8.

2.1.1 AXI slave interface

For detailed information about the AXI interface, see the *AMBA AXI Protocol Specification*. The AXI slave interface comprises the following AXI channels:

AXI write address channel

Enables the transfer of the address and all other control data required for the device to carry out an AXI write transaction.

AXI write data channel

Enables the transfer of write data and validating data byte strobes to the device.

AXI buffered write response channel

Enables the transfer of response information associated with a write transaction and a write data channel transaction.

AXI read address channel

Enables the transfer of the address and all other control data required for the device to carry out an AXI read transaction.

AXI read data channel

Enables the transfer of read data and response information associated with a read transaction.

Note

It is possible for refreshes to be missed if **rready** is held LOW for longer than one refresh period and the read data FIFO, command FIFO, and arbiter queue become full. An OVL error is triggered if this occurs in simulation. Ensure that the device has a sufficiently high system priority to prevent this.

Figure 2-2 on page 2-4 shows the AXI slave interface external connections.

Note

Figure 2-2 on page 2-4 to Figure 2-6 on page 2-7 do not show reset signals. See Table A-1 on page A-2 for details.

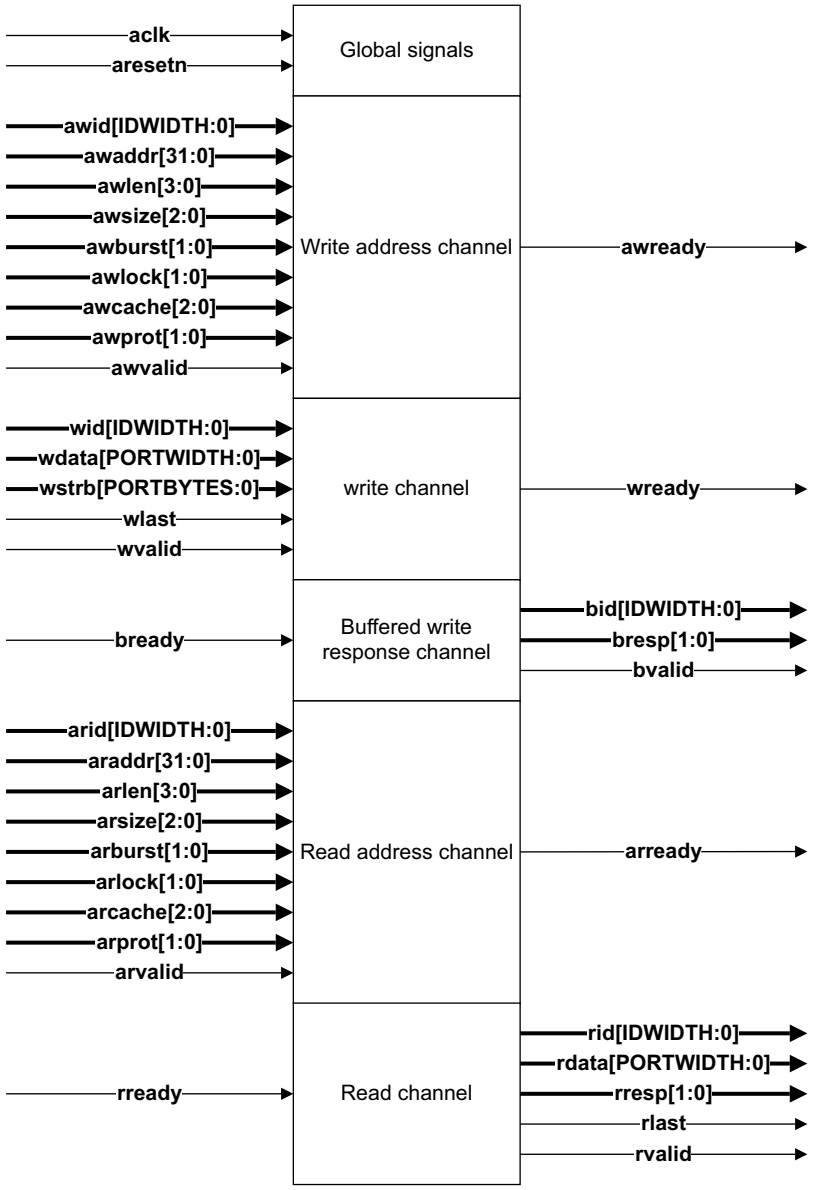


Figure 2-2 AXI slave interface connections

2.1.2 Arbiter

The arbiter receives memory access commands from the AXI slave interface and the memory manager. It passes the highest priority command to the memory interface after arbitration. Read data is passed from the memory interface to the AXI slave interface.

2.1.3 Memory manager

The memory manager tracks and controls the current state of the device using a *Finite State Machine* (FSM). See Figure 2-3.

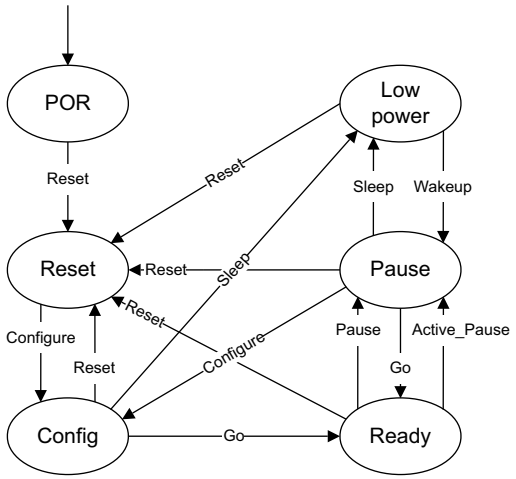


Figure 2-3 clk domain state diagram

In Figure 2-3, non-state moving transitions are omitted for clarity. See Table 2-2 on page 2-29 for valid system states.

Note

If you move into the Pause state using Active_Pause, you are not permitted to enter the Config state.

APB commands to the Direct Command Register, see *Direct Command Register* on page 3-10, or the AXI low-power interface control the state of the device. Figure 2-4 on page 2-6 shows AXI low-power interface channel signals.

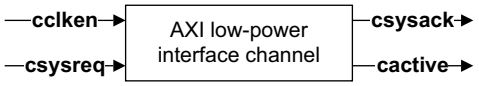


Figure 2-4 AXI low-power interface channel signals

The AXI low-power interface channel enables low-power mode to be entered using discrete lines. You can tie-off this interface to be inactive if it is not required.

The APB slave interface stalls the **psel** and **penable** signals using the **pready** signal if a previous command has not completed.

If an APB command is received that is illegal to carry out from the current state, then it is ignored and the FSM stays in the current state.

The memory manager enables the APB slave interface to directly send initialization commands to the external memory SDRAM and periodically generate refresh commands for the external memory SDRAM. This is done using the direct_cmd register. See the following sections for more information on this register:

- *Direct Command Register* on page 3-10
- *Memory manager* on page 2-18.

2.1.4 APB slave interface

The APB slave is a fully compliant APB slave. The device has 4KB of memory allocated to it. For detailed information about the APB interface, see the *AMBA 3 APB Protocol Specification*. The APB interface implements the register file as Chapter 3 *Programmer's Model* describes.

Note

The APB only supports single-word 32-bit accesses. Bits [1:0] of the **paddr** signal are not used within the device, resulting in byte and half-word accesses being treated as word accesses.

Figure 2-5 on page 2-7 shows the APB external connections.

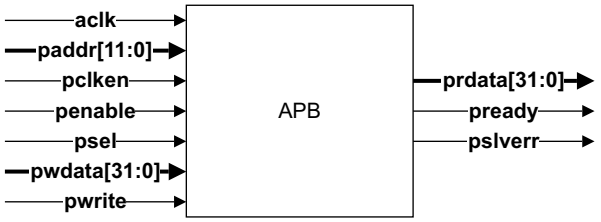


Figure 2-5 APB external connections

2.1.5 Memory interface

The memory interface provides a clean and defined interface between the pad interface and the arbiter, ensuring that the external memory interface command protocols are met in accordance with the programmed timings in the register block. See Chapter 3 *Programmer's Model*.

The external I/Os to this block are:

mclk Clock for **mclk** domain.

mresetn Reset for **mclk** domain. This signal is active LOW.

The memory interface tracks and controls the state of the external memories using an FSM. See Figure 2-6.

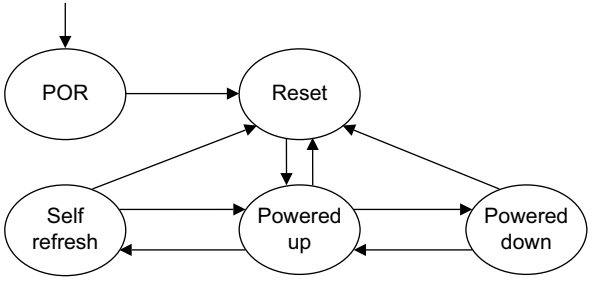


Figure 2-6 mclk domain FSM

See Table 2-2 on page 2-29 for valid system states.

2.1.6 Pad interface

The pad interface is a replaceable block designed to operate with DDR2 SDRAM memory. It provides a flip-flop for each external signal.

Figure 2-7 shows the pad interface external connections.

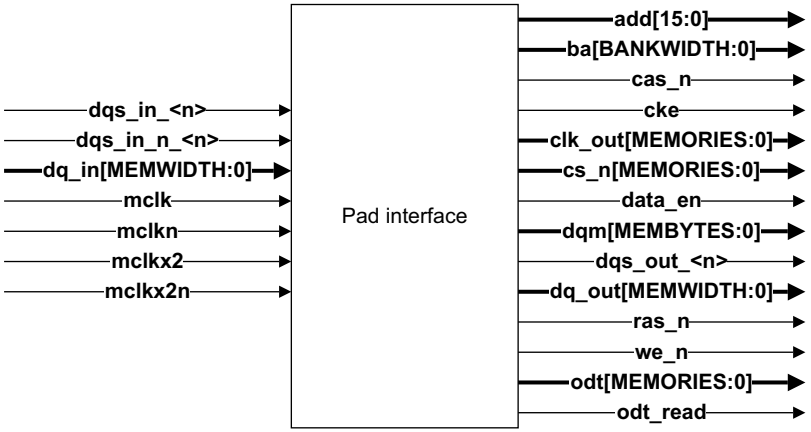


Figure 2-7 Pad interface external connections

Pad interface to external memory devices

The pad interface block registers the relevant command signals with clocks that enable the external memory device timing to be met.

It is expected that a *Delay-Locked Loop* (DLL) is required to delay the **dqs** signals coming back from the memories with respect to the dq data bus. The standard delay for the DQS signals is a quarter clock period of **mclk**. The DLL is not included in the device. The **asetbok** signal enables DLLs to update at safe points in time. See *Miscellaneous signals* on page A-3.

2.2 Functional operation

This section describes:

- *Clocking and resets*
- *Miscellaneous signals* on page 2-10
- *AXI slave interface* on page 2-11
- *Arbiter* on page 2-12
- *Memory manager* on page 2-18
- *APB slave interface* on page 2-19
- *Memory interface* on page 2-19
- *Pad interface* on page 2-25
- *Initialization* on page 2-26
- *Power-down support and usage model* on page 2-29.

2.2.1 Clocking and resets

This section describes:

- *Clocking*
- *Reset* on page 2-10.

Clocking

The device has the following functional clock inputs:

- **aclk**
- **mclk**
- **mclkn**
- **mclkx2**
- **mclkx2n**
- **dqs_in_<n>** where n is 0<n<bytes of external memory data bus
- **dqs_in_n_<n>** where n is 0<n<bytes of external memory data bus.

These clocks can be grouped into two clock domains:

aclk domain **aclk** is in this domain. The **aclk** domain signals can only be stopped if the external memories are put in self-refresh mode.

mclk domain All clocks except **aclk** are in this domain. The **mclk** signal must be clocked at the rate of the external memory clock speed. The **mclk** domain signals can only be stopped if the external memories are put in self-refresh mode.

Note

A clock output is provided for every external memory device.

Reset

The device has two reset inputs:

aresetn This is the reset signal for the **aclk** domain.

mresetn This is the reset signal for the **mclk** domain.

You can change both reset signals asynchronously to their respective clock domain. Internally to the device, the deassertion of the **aresetn** signal is synchronized to **aclk**, and the deassertion of the **mresetn** signal is synchronized to the **mclk**, **mclkn**, **mclkx2**, and **mclkx2n** clock signals.

2.2.2 Miscellaneous signals

You can use the following signals as general-purpose control signals for logic external to the device:

user_status[7:0] General-purpose ports that are readable from the APB interface. If you do not require these ports you must tie them either HIGH or LOW. These ports are connected directly to the APB interface block. Therefore, if they are driven from external logic that is not clocked by the **aclk** signal, then external synchronization registers are required.

user_config0 General-purpose port that is driven directly from a write-only APB register. If you do not require this port, leave it unconnected.

user_config1 General-purpose port that is driven directly from a write-only APB register. If you do not require this port, leave it unconnected.

You can use the following miscellaneous signals as tie-offs to change the operational behavior of the device:

rst_bypass Use this signal for ATPG testing only. You must tie it LOW for normal operation.

dft_en_clk_out

Use this signal for ATPG testing only. You must tie it LOW for normal operation.

2.2.3 AXI slave interface

The AXI programmer's view is of a flat area of memory. The full range of AXI operations are supported.

The base addresses of the external memory devices are programmable using the chip_cfg Registers. See *chip_<n>_cfg Registers* on page 3-24.

In addition to reads and writes, exclusive reads and writes are supported in accordance with the *AMBA AXI Protocol Specification*.

Successful exclusive accesses have an `EXOKAY` response. All other accesses, including exclusive fail accesses, receive an `OKAY` response.

Note

The **arcache** and **arprot** signals are included in the memory controller port list for completeness only. There is no requirement for the device to use these signals.

This section describes:

- *Configuration options*
- *Write data merging*
- *Early BRESP*.

Configuration options

You can choose 128-bit, 64-bit, or 32-bit data width.

Write data merging

The device merges interleaved write data to optimize the utilization of the memory interface.

Early BRESP

To enable early write response timing, the device employs write data buffering and issues the BRESP transfer before the data has been committed to the SDRAM. The response is sent after the last data beat is accepted by the AXI interface and stored in the write data buffer. You can disable this feature via the APB feature control register. See *feature_ctrl Register* on page 3-26.

Note

Exclusive write accesses always wait until the write transaction has been committed to memory before issuing the BRESP transfer.

2.2.4 Arbiter

This section describes:

- *Formatting from AXI address channels*
- *Formatting from memory manager* on page 2-14
- *Arbiter access mux* on page 2-14
- *QoS* on page 2-14
- *Hazard detection* on page 2-15
- *Scheduler* on page 2-16
- *Arbitration algorithm* on page 2-16
- *Command formatting* on page 2-17
- *Exclusive access monitor* on page 2-17.

Formatting from AXI address channels

Formatting is as follows:

Chip select decoding

Using the programmed values in the chip_<n>_cfg Registers, see *chip_<n>_cfg Registers* on page 3-24, that Chapter 3 *Programmer's Model* defines, an incoming address has the most significant eight address bits compared with the address match bits using the address mask to ignore any don't care bits to select an external chip.

The transfer is still carried out if there is no match, but the result is undefined.

Row select decoding

The row address is determined from the AXI address using bits [5:3] of the memory_cfg Register, and also the brc_n_rbc bit for the selected chip defined in the chip_<n>_cfg Register. See *Memory Configuration Register* on page 3-11 and *chip_<n>_cfg Registers* on page 3-24.

Column select decoding

The column address is determined from the AXI address using bits [2:0] of the memory_cfg Register. See *Memory Configuration Register* on page 3-11.

Bank select decoding

The chip bank is determined from the AXI address using bits [5:3] of the memory_cfg2 Register, and also the brc_n_rbc bit for the selected chip defined in the chip_<n>_cfg Register. See *memory_cfg2 Register* on page 3-21 and *chip_<n>_cfg Registers* on page 3-24.

Number of beats

The number of memory beats is determined, depending on the effective external memory width and the burst size of the AXI access. AXI wrapping bursts are split into two incrementing bursts. AXI fixed bursts are split into AXI burst length memory bursts.

Note

The device only supports a burst length of four on the memory interface.

Quality of Service (QoS) selection

QoS is defined for the device as a method of increasing the arbitration priority of a read access that requires low-latency read data. The QoS for an AXI read access is determined when it is received by the arbiter. There is no QoS for write accesses.

There are two forms of QoS tracking:

- qos_max time-out
- qos_min time-out.

The following example shows that the type of QoS, and the QoS value are determined by the id_<n>_cfg Register values pointed to from the **ARID** bits selected by the qos_master_bits in the memory_cfg Register. See *Memory Configuration Register* on page 3-11 and *id_<n>_cfg Registers* on page 3-23.

If a transfer is received that has an **ARID** of `0x5A`, and the qos_master_bits in the memory_cfg Register are set to b010, then the four ID bits used to match to a QoS pointer are **ARID[5:2]**, giving a value `0x6`. See *Memory Configuration Register* on page 3-11.

This means that when the transfer is received, the id_6_cfg Register determines the qos_enable, qos_min, and qos_max values for this transfer. If the qos_enable bit is HIGH, then the new arbiter entry that is created for this transfer is assigned the qos_min value and qos_max value from the id_6_cfg Register. In addition to this, if the qos_override bit associated with this transfers ID, qos_override[6], is HIGH when the **arvalid** and **arready** signals are HIGH, then the qos_min arbiter entry is forced HIGH irrespective of whether the qos_enable bit is HIGH.

Formatting from memory manager

The direct command uses the chip select bits [21:20] in the direct_cmd register to select the required memory chip. See *Direct Command Register* on page 3-10.

The command to be carried out is either:

- a self-refresh request from the AXI-C interface
- an auto-refresh
- a direct command from the APB interface.

It is encoded by the memory manager to match the format that the arbiter requires.

Arbiter access mux

The selection of a command from the AXI interface or the memory manager is fixed, with the memory manager having a higher priority.

The selection between an AXI read access and AXI write access is made using a round-robin arbitration, unless a read access has a low latency QoS value, or if any of the accesses are to a row already open. For more information on arbitration algorithm, see *Arbitration algorithm* on page 2-16.

See also *Formatting from AXI address channels* on page 2-12. In this case, it is arbitrated immediately.

QoS

For write accesses, no QoS is provided. However, any write access that is a dependency for a QoS read access receives a promoted priority to complete as soon as possible. Dependencies are formed based on the hazard detection logic that *Hazard detection* on page 2-15 describes.

See *Formatting from AXI address channels* on page 2-12 for how the QoS is selected for an AXI read access.

If the QoS enable bit for the **ARID** is set in the register bank, the QoS maximum latency value is decremented every cycle until it reaches zero.

If the entry is still in the queue when the QoS maximum latency value reaches zero, then the entry becomes high priority. This is called a *time-out*. Also, any entry in the queue with a minimum latency QoS also produces a time-out. Minimum latency time-outs have priority over maximum latency time-outs.

When an entry times out in this way it forces a time-out onto any entries that it has dependencies against. In normal operation, these entries have already timed out because they have received the same initial QoS value, but been decrementing for longer. The highest priority arbiter entry is serviced next.

One special case exists. This is when or if the qos_override function forces a minimum latency time-out. In this instance, any accesses that the new entry has dependencies against might not have timed out and are forced to time out so that the high-priority entry can start as soon as possible.

A QoS is also provided for the auto-refresh commands from the memory manager. The arbiter keeps track of the number of outstanding auto-refresh commands with a simple increment-decrement counter per chip. If the number of auto-refresh commands reaches a set limit of six, a refresh time-out is signalled. This time-out is sticky, and does not disappear when the number of time-outs drops back below the threshold. Instead, it remains asserted until all of the auto-refreshes have been serviced. This provides a guaranteed refresh rate in the SDRAM.

Hazard detection

The following types of hazard exist:

Read After Read (RAR)

There is a read already in the arbiter queue with the same ID as the incoming entry, that is also a read.

Write After Write (WAW)

There is a write already in the arbiter queue with the same ID as the incoming entry, that is also a write.

Read After Write (RAW)

There is a write in the arbiter queue, that has received an early write response, accessing the same location as the incoming read entry.

The arbiter entry is flagged as having a dependency if a hazard is detected. There might be dependencies against a number of other arbiter entries. As the arbiter entries are invalidated, so the dependencies are reduced until finally, there are no outstanding dependencies, and the entry is free to start.

Note

There are no *Write-After-Read* (WAR) hazard checks in the device. If an AXI master requires ordering between reads and writes to certain memory locations, it must wait for read data before issuing a write to a location it has read from. Similarly, the only RAW hazard checking is that performed when the write response has been issued. If an AXI master required ordering between writes and reads to certain memory locations, it must wait for the write response before issuing the read to the same location.

Scheduler

The scheduler keeps track of the activity of the bank FSMs in the memory interface. This enables the arbiter to select an entry from the queue that does not stall the memory pipeline.

Arbitration algorithm

The ordering of commands to be carried out from the arbiter queue is arbitrated with a priority scheme of the following order:

- refresh timeouts
- minimum latency timeouts
- maximum latency timeouts
- open-row hits in the same direction
- open-row hits in the opposite direction
- preparation operations
- refreshes.

Note

Preparation operations can be interleaved between memory operations to other banks to improve memory interface utilization.

Command formatting

For every memory burst access necessary to complete an arbiter queue entry, a memory interface command is required.

Command formatting calculates the number of memory interface commands and memory cycles for each command to complete the next arbiter queue entry that is to be sent to the memory interface. It contains an address incrementor and a beat decrementor so that the arbiter entry can be interrupted and restarted.

Exclusive access monitor

This keeps track of a configurable number of outstanding exclusive accesses, up to four, as the *AMBA AXI Protocol Specification* describes. If an exclusive write fails, the data mask for the write is forced LOW, so the data is not written.

When monitoring an exclusive access, the address of any write from another master is compared with the monitored address to check that the location is not being updated.

If the write crosses an address boundary, then all of the least-significant address bits up to that boundary are not compared. Example 2-1 provides information about three transactions.

Example 2-1

Exclusive Read	Address = `0x000`, size = WORD, length = 1, ID = 0.
Write	Address = `0x004`, size = WORD, length = 2, ID = 1.
Exclusive Write	Address = `0x000`, size = WORD, length = 1, ID = 0.

The device marks the exclusive write as having failed because the write crosses the `0x010` address boundary. The write, for comparison purposes, has therefore accessed the region `0x000-0x01f` because bits 4:0 have not been compared.

The burst type is always taken to be INCR when calculating the address region accessed by the write, Therefore, a wrapped transaction in Example 2-1 that wraps down to `0x0` rather than cross the boundary, is treated in the same way. This is the same for a fixed burst that does not cross the boundary or wrap down to `0x0`.

2.2.5 Memory manager

The memory manager issues commands from one of the following sources:

Direct commands

These are received over the APB interface as a result of a write to the direct_cmd Register. See *Direct Command Register* on page 3-10. They initialize the SDRAM. The only valid commands that the memory manager can handle are:

- NOP
- PRECHARGEALL
- AUTOREFRESH
- MODEREG
- EXTENDED MODEREG.

Commands from the status FSM

You can traverse the memory manager status FSM by writing to the memc_cmd Register. See *Memory Controller Command Register* on page 3-8. You can only traverse the status FSM states when the device is idle. For example, the ready state can only be entered from the Config state when all direct commands have been completed. The exception to this is the ACTIVE_PAUSE command. You can issue this command when the device is active. When you issue the command, any memory accesses that have not been arbitrated remain in the arbiter until the FSM receives a GO command.

Refresh commands

The refresh FSM can issue commands to the arbiter to refresh the SDRAM chips. The refresh counter is clocked by the memory clock to enable the frequency of the device to be scaled without affecting the refresh rate. The refresh rate period is programmable using the RefreshPrd Register. See *Refresh Period Register* on page 3-12. The value of this register is the count value in **mclk** cycles.

Programmable options

Auto Refresh Request Period, RefreshPrd, is the time period in **mclk** clock cycles that the memory manager generates a request for the arbiter to generate an auto-refresh command. This request is arbitrated as another command and is not necessarily initiated immediately. See *Arbiter* on page 2-12.

2.2.6 APB slave interface

The APB interface is clocked by the same clock as the AXI domain clock, **aclk**, but has a clock enable so that it can be slowed down to execute at an integer divisor of **aclk**.

To enable a clean registered interface to the external infrastructure, the APB interface always adds a wait state for all reads and writes by driving **pready** LOW. In the following instances, a delay of more than one wait state can be generated:

- when a direct command is received and there are outstanding commands that prevent a new command being stored in the command FIFO
- when a memory command is received, and a previous memory command has not been completed.

The only registers that can be accessed when the device is not in the Config or low-power state are:

- the Memory Controller Status Register, to read the current state, see *Memory Controller Status Register* on page 3-7
- the Memory Controller Command Register, to change state, see *Memory Controller Command Register* on page 3-8.

To guarantee no missed auto-refresh commands, it is recommended that any change of **mclk** period, and therefore update of the refresh period, is carried out when the device is in the low-power state. This is because the refresh rate depends on the **mclk** period. Only write direct commands to the external memories when the device is in the Config state and not in the low-power state.

2.2.7 Memory interface

The memory interface is separated from the arbiter using the following configurable FIFOs:

- command FIFO
- read data FIFO
- write data FIFO.

There is also a static interface that has configuration signals that cannot be changed when the interface is operating.

The memory interface reads commands from the arbiter using a FIFO, but only when that command can be executed. The memory interface ensures a command is only executed when all the inter-command delays, defined in this section, for that bank or chip are met. The memory interface enables multiple banks to be active at any one time.

However, only one bank can be carrying out a data transfer at any one time. If the command at the head of the FIFO cannot be executed, then the command pipeline stalls until it can be executed.

The timing parameters in Figure 2-8 on page 2-21 to Figure 2-20 on page 2-25 can all be programmed using the APB interface. See Chapter 3 *Programmer's Model*.

When the auto_power_down Register bit is set, see *Memory Configuration Register* on page 3-11, then the **cke** output pin is negated to take the external memories into active or precharge power-down depending on whether there is a row open. See Figure 2-17 on page 2-24. When exiting power down mode, the delay before the next command is issued is defined by the register value t_{XP}.

There is an FSM to control the operation of the power-down mode. This FSM has a state that is entered when the SDRAM is put into self-refresh mode. This is used so that if power is removed from all of the device apart from the memory interface and pad interface, the state of the memory is known. When the rest of the device is powered-up, the status FSM enters the low-power state rather than the Config state.

Memory interface to pad interface timing

All command control outputs are clocked on the same edge. In Figure 2-8 on page 2-21 to Figure 2-20 on page 2-25, the control outputs to the external memory are always clocked on the falling edge of the memory clock.

The relative times between control signals from the memory interface are maintained when output from the pad interface to the actual SDRAM devices. Therefore, the timing register values required for a particular SDRAM device can be determined from that SDRAM device's data sheet. Figure 2-8 on page 2-21 to Figure 2-20 on page 2-25 show how the data sheet timings map onto the device timing registers.

The times in Figure 2-8 on page 2-21 to Figure 2-20 on page 2-25 are not necessarily the default timing values, but are values that are small enough to show the entire delay in one figure.

Note

The following signals are internal to the device:

- **asetbok**
- **command_en**
- **data_cntl_en**
- **odt_read**
- **read_en**.

Figure 2-8 shows the command control output timing.

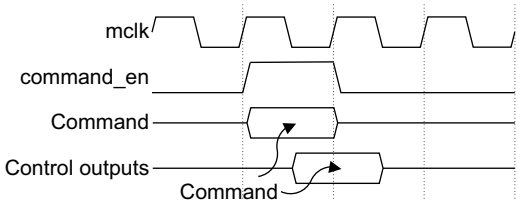


Figure 2-8 Command control output timing

Figure 2-9 shows the activate to read or write command timing.

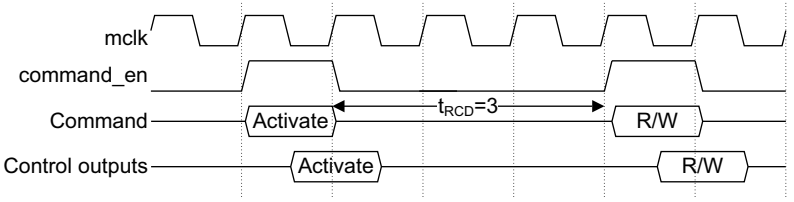


Figure 2-9 Activate to read or write command timing, t_{RCD}

Figure 2-10 shows the four activate window command timing.

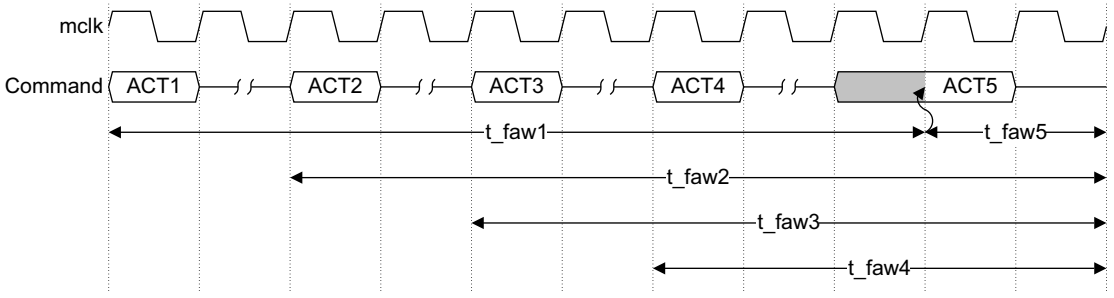


Figure 2-10 Four activate window command timing, t_{FAW}

Figure 2-11 shows the bank activate to bank activate or auto-refresh command timing.

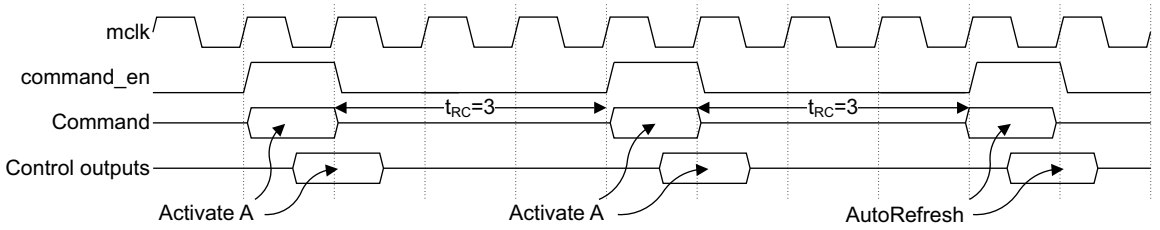


Figure 2-11 Bank activate to bank activate or auto-refresh command timing, t_{RC}

Figure 2-12 shows the bank activate to different bank activate for a memory timing.

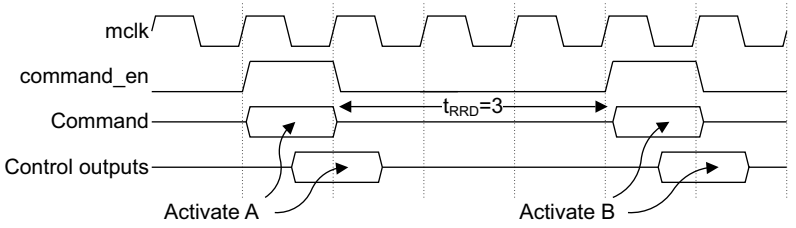


Figure 2-12 Bank activate to different bank activate for a memory timing, t_{RRD}

Figure 2-13 shows the precharge to command and auto-refresh timing.

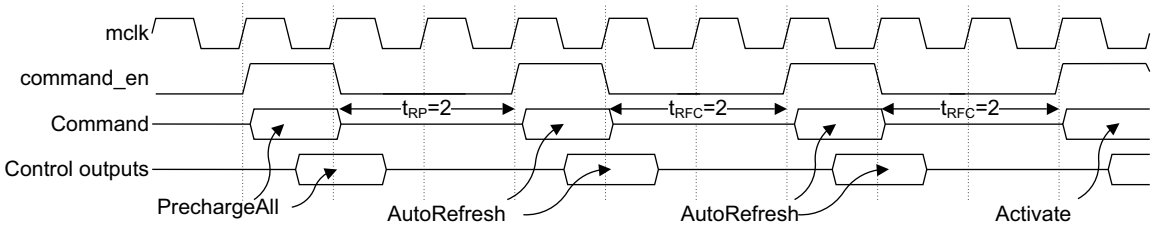


Figure 2-13 Precharge to command and auto-refresh timing, t_{RP} and t_{RFC}

Figure 2-14 shows activate to precharge, and precharge to precharge timing.

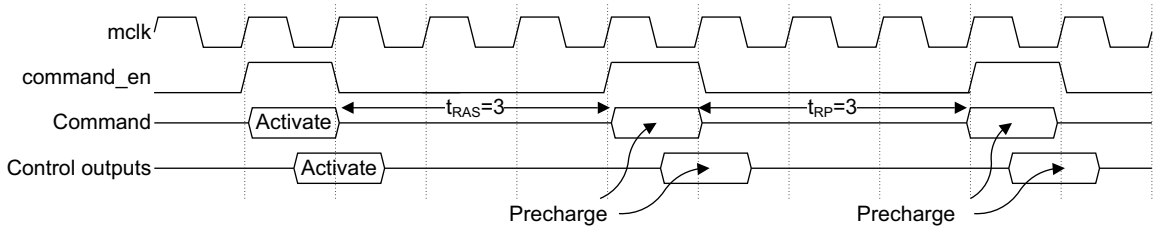


Figure 2-14 Activate to precharge, and precharge to precharge timing, t_{RAS} and t_{RP}

Figure 2-15 shows mode register write to command timing.

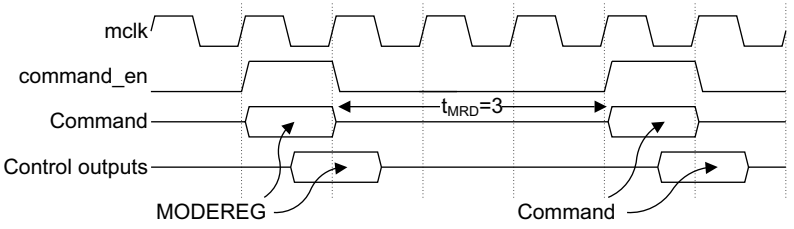


Figure 2-15 Mode register write to command timing, t_{MRD}

Figure 2-16 shows self-refresh entry and exit timing.

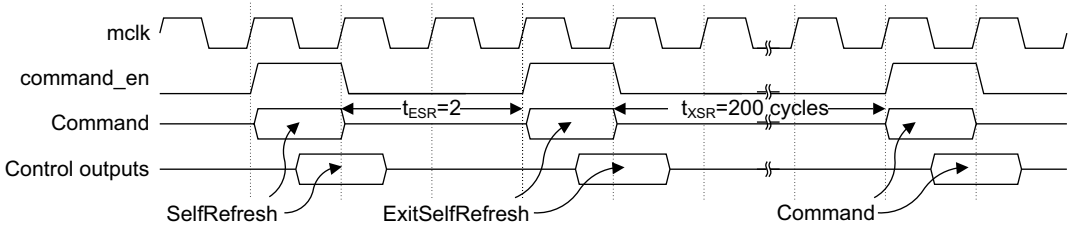


Figure 2-16 Self-refresh entry and exit timing, t_{ESR} and t_{XSR}

Figure 2-17 shows power-down entry and exit timing.

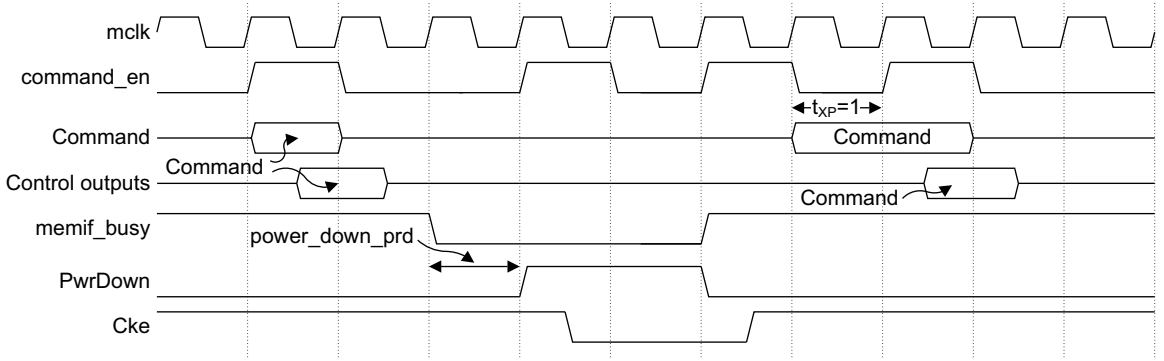


Figure 2-17 Power down entry and exit timing, t_{XP}

The **pwr_down_prd** count is timed from the memory interface becoming idle, that is, after a command delay has timed out or the read data FIFO is emptied. **cke** is asserted when the command FIFO is not empty.

Figure 2-18 shows the turnaround time, t_{WTR}, for the memory interface to output a Write command followed immediately by a Read command.

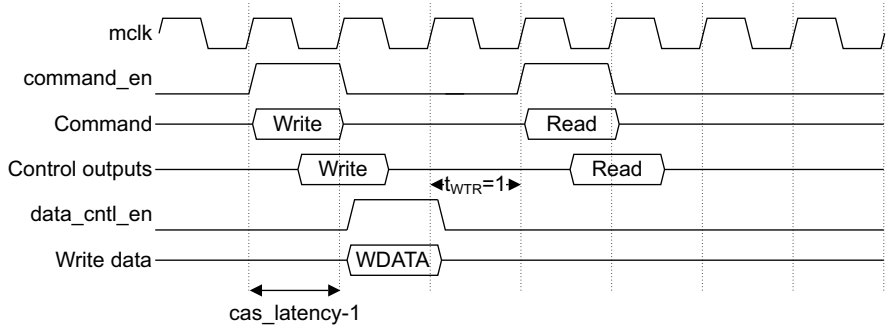


Figure 2-18 Data output timing, t_{WTR}

Figure 2-19 on page 2-25 shows the relationship between memory interface outputting the Write command and the WDATA when CAS latency is set to 3, resulting in a write latency of 2. It also highlights the t_{WR} minimum time between a Write and a Precharge command.

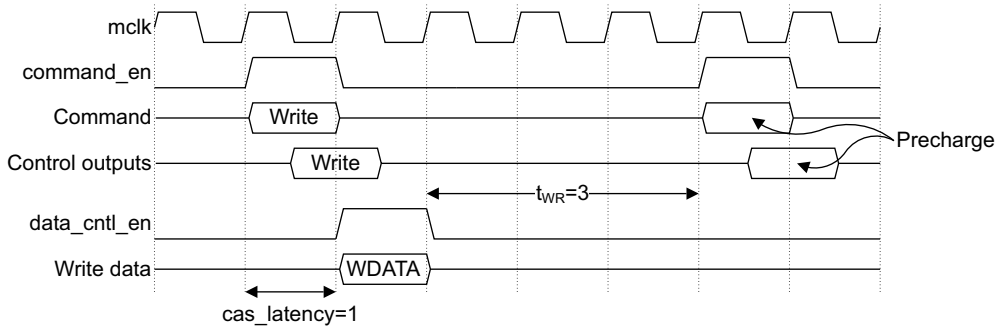


Figure 2-19 Data output timing, write latency = 2

Figure 2-20 shows the timing relationship between the Read command being output from the memory interface and the RDATA being returned to the memory interface from the pad interface.

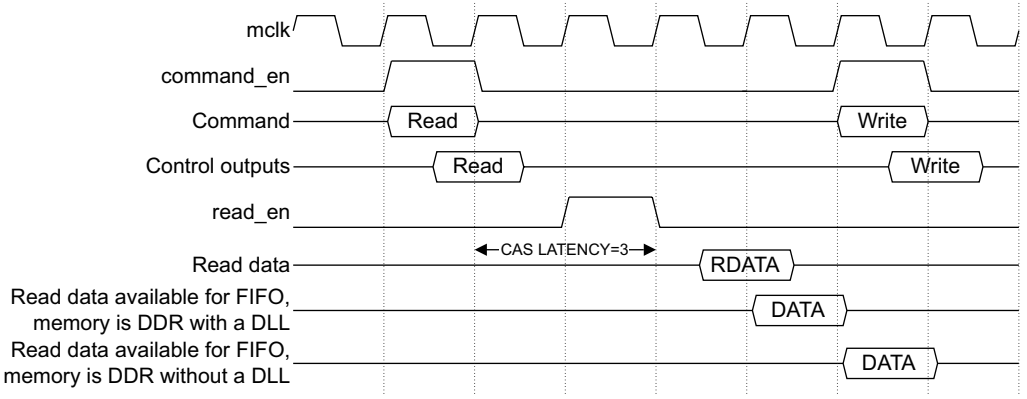


Figure 2-20 Data input timing

2.2.8 Pad interface

You can replace or modify the pad interface, if required, for additional optimization for a particular memory type or target library, or to use a hard macro.

It is important if the pad interface is modified or replaced that the relative timings of the control output signals enabled by **command_en** and **data_cntl_en** signals are maintained to ensure the timings carried out in the memory interface block are still correct at the external memory bus interface. The **read_en** signal is always asserted one **mclk** period before the expected read data. Therefore, the timing of **read_en** changes as CAS latency is changed using the APB interface.

When read commands are being executed, the data is clock-enabled into a FIFO clocked by **mclk** a set time from the command being clocked-out. See Figure 2-20 on page 2-25. This delay is dependent on the current **cas_latency** value programmed and the delays in the pad interface.

2.2.9 Initialization

Before you can use the device operationally to access external memory, you must:

- set up the configuration registers
- initialize the external memory devices.

You might not have to configure all the registers because some might power-up to the correct value. See Chapter 3 *Programmer's Model*. For completeness, Table 2-1 includes all register values.

Note

You might create a deadlock situation if the AXI port is accessed by a master before that master has configured the device using the APB port. A master that cannot access the APB port but accesses the AXI port before the device has been configured is held off until another master configures the device.

Example setup

Table 2-1 lists an example DDR2 setup.

Table 2-1 Example DDR2 setup

Register address	Write data	Description
0x80000014	0x00000006	Set cas_latency to 3
0x8000001C	0x00000002	Set t_mrd to 2
0x80000020	0x00000008	Set t_ras to 8
0x80000024	0x0000000B	Set t_rc to 11
0x80000028	0x00000103	Set t_rcd to 3 and schedule_rcd to 1
0x8000002C	0x00001315	Set t_rfc to 21 and schedule_rfc to 19
0x80000030	0x00000103	Set t_rp to 3 and schedule_rp to 1
0x80000034	0x00000002	Set t_rrd to 2

Table 2-1 Example DDR2 setup (continued)

Register address	Write data	Description
0x80000038	0x00000003	Set t_wr to 3
0x8000003C	0x00000002	Set t_wtr to 2
0x80000040	0x00000002	Set t_xp to 2
0x80000044	0x00000003	Set t_xsr to 3
0x80000048	0x000000C8	Set t_esr to 200
0x8000000C	0x0001A411	Set memory configuration[a]
0x80000010	0x00000618	Set auto refresh period to be every 1560 **mclk** periods
0x8000004C	0x00000411	Set memory configuration[b]
0x80000200	0x000000FF	Set chip select for chip 0 to be 0x00XXXXXX, *Row Bank Column* (RBC) configuration
0x80000204	0x000022FF	Set chip select for chip 1 to be 0x22XXXXXX, RBC configuration
0x80000208	0x000055FF	Set chip select for chip 2 to be 0x55XXXXXX, RBC configuration
0x8000020C	0x00007FFF	Set chip select for chip 3 to be 0x7FXXXXXX, RBC configuration
0x80000008	0x000C0000	Carry out chip0 Nop command
0x80000008	0x00000000	Carry out chip0 Prechargeall command
0x80000008	0x00040000	Carry out chip0 Autorefresh command
0x80000008	0x00040000	Carry out chip0 Autorefresh command
0x80000008	0x00080032	Carry out chip0 Mode Reg command 0x32 mapped to low add bits
0x80000008	0x00090000	Carry out chip0 extended Mode Reg command 0x0 mapped to low address bits
0x80000008	0x001C0000	Carry out chip1 Nop command
0x80000008	0x00100000	Carry out chip1 Prechargeall command
0x80000008	0x00140000	Carry out chip1 Autorefresh command
0x80000008	0x00140000	Carry out chip1 Autorefresh command
0x80000008	0x00180032	Carry out chip1 Mode Reg command 0x32 mapped to low add bits
0x80000008	0x00190000	Carry out chip1 extended Mode Reg command 0x0 mapped to low address bits

Table 2-1 Example DDR2 setup (continued)

Register address	Write data	Description
0x80000008	0x002C0000	Carry out chip2 Nop command
0x80000008	0x00200000	Carry out chip2 Prechargeall command
0x80000008	0x00240000	Carry out chip2 Autorefresh command
0x80000008	0x00240000	Carry out chip2 Autorefresh command
0x80000008	0x00280032	Carry out chip2 Mode Reg command 0x32 mapped to low add bits
0x80000008	0x00290000	Carry out chip2 extended Mode Reg command 0x0 mapped to low address bits
0x80000008	0x003C0000	Carry out chip3 Nop command
0x80000008	0x00300000	Carry out chip3 Prechargeall command
0x80000008	0x00340000	Carry out chip3 Autorefresh command
0x80000008	0x00340000	Carry out chip3 Autorefresh command
0x80000008	0x00380032	Carry out chip3 Mode Reg command 0x32 mapped to low add bits
0x80000008	0x00390000	Carry out chip3 extended Mode Reg command 0x0 mapped to low address bits
0x80000004	0x00000000	Change device state to Ready

a. The memory is configured as follows:
- 9 column bits, 13 row bits
- power-down period of 0
- auto power-down is disabled
- dynamic clock stopping is disabled
- memory burst size of 4
- ARID[5:2] bits to be used for QoS - four active chips.

b. The memory is additionally configured as follows:
- **aclk** and **mclk** are synchronous
- DQM is LOW at reset
- CKE is LOW at reset
- two bank bits are in use
- 16-bit data lines are in use
- DDR2 SDRAM is connected.

2.2.10 Power-down support and usage model

The device provides architectural support for low-power operation in the following ways:

- By using the memory_cfg Register at address offset `0xC`. See *Memory Configuration Register* on page 3-11. The device can automatically place the SDRAM into either the precharge power-down or active power-down states by deasserting **cke** when the SDRAM has been inactive for a number of clock cycles. This occurs with the device in its READY state.
- By using the memc_cmd and memc_status Registers at address offsets `0x4` and `0x0`. The device can place the SDRAM into the self-refresh state under software control. See *Memory Controller Status Register* on page 3-7 and *Memory Controller Command Register* on page 3-8.
- By using the AXI low-power interface, the device can place the SDRAM into the self-refresh state under hardware control.

Additionally, the microarchitecture provides additional power savings through extensive use of clock gating.

It is possible to implement the device with two power domains:

- APB and AXI, **aclk**
- memory, **mclk**.

See Figure 2-1 on page 2-2.

Table 2-2 lists the valid system states of the **aclk** domain FSM and the **mclk** domain FSM. It also lists the valid power, clock, and reset states in the **aclk** and **mclk** domains. Table 2-3 on page 2-31 lists the valid transitions, and the text following it explains how to traverse the system states.

Table 2-2 Valid system states for FSMs

SDRAM		DDR2 DMC								System state
		aclk FSM				mclk FSM				
V_{DD}	State	V_{DD}	Clock	Reset	State	V_{DD}	Clock	Reset	State	
0	Null	0	N/a	N/a	Null	0	N/a	N/a	Null	1
0	Null	>0	Running	No	POR	>0	Running	No	POR	2
0	Null	>0	Running	Yes	Reset	>0	Running	Yes	Reset	3

Table 2-2 Valid system states for FSMs (continued)

SDRAM		DDR2 DMC								System state
		aclk FSM				mclk FSM				
V_{DD}	State	V_{DD}	Clock	Reset	State	V_{DD}	Clock	Reset	State	
0	Null	>0	Running	No	Config	>0	Running	No	Powered_ up	4
>0	Accessible	>0	Running	No	Config	>0	Running	No	Powered_ up	5
>0	Accessible	>0	Running	No	Ready	>0	Running	No	Powered_ up	6
>0	Powered-down	>0	Running	No	Ready	>0	Running	No	Powered_ down	7
>0	Self_ refresh	>0	Running	No	Low_ power	>0	Running	No	Self_ refresh	8
>0	Self_ refresh	>0	Running	No	Low_ power	>0	Stopped	No	Self_ refresh	9
>0	Self_ refresh	>0	Stopped	No	Low_ power	>0	Running	No	Self_ refresh	10
>0	Self_ refresh	>0	Stopped	No	Low_ power	>0	Stopped	No	Self_ refresh	11
>0	Self_ refresh	0	N/a	N/a	Null	>0	Stopped	No	Self_ refresh	12
>0	Self_ refresh	0	N/a	N/a	Null	>0	Running	No	Self_ refresh	13
>0	Self_ refresh	>0	Running	No	POR	>0	Stopped	No	Self_ refresh	14
>0	Self_ refresh	>0	Running	No	POR	>0	Running	No	Self_ refresh	15
>0	Self_ refresh	>0	Running	Yes	Reset	>0	Stopped	No	Self_ refresh	16
>0	Self_ refresh	>0	Running	Yes	Reset	>0	Running	No	Self_ refresh	17

The ranking of system power states, from highest power to lowest power, is as follows:

6, 7, 8, 10, 9, 11, 13, 12.

However, states 8-11 are similar and the recommendation is to use state 11 from this group if clock-stopping techniques are available. Similarly, states 12 & 13 are similar and the recommendation is to use state 12 from this pair. Table 2-3 lists a recommended set of power states.

Table 2-3 Recommended power states

System state	Power name
6	Running
7	Auto power-down
11	Shallow self-refresh
12	Deep self-refresh

These states and arcs are highlighted in Figure 2-21.

Note

States 1-5, 9, 14, and 16 are only used as transitional states.

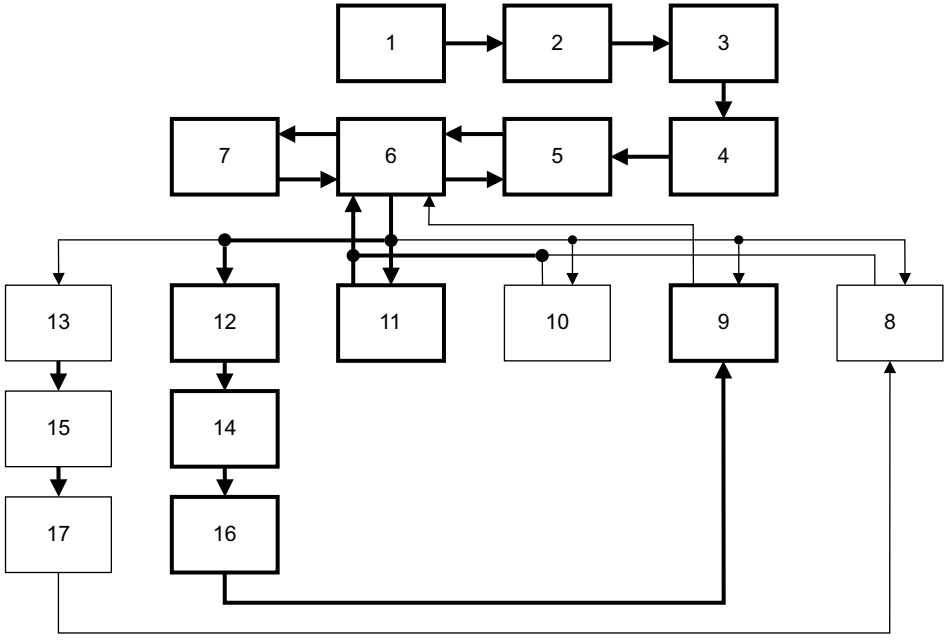


Figure 2-21 System state transitions

State transitions are as follows:

Arc 1 to 2 Apply power to all DDR2 DMC power domains, and ensure that **aclk** and **mclk** are running.

Arc 2 to 3 Assert reset in both the **aclk** reset domain and the **mclk** reset domain.

Arc 3 to 4 Deassert reset in both the **aclk** reset domain and the **mclk** reset domain.

Arc 4 to 5 Apply power to the SDRAM power domain.

Arc 5 to 6 You must:

1. Write to all of the memory timing parameters, address offsets `0x14` to `0x44`.
2. Write to the memory_cfg and refresh_prd Registers, address offsets `0xC` and `0x10`. See *Memory Configuration Register* on page 3-11 and *Refresh Period Register* on page 3-12.
3. Initialize the memory, using the direct_cmd Register, see *Direct Command Register* on page 3-10, offset `0x8`, with the sequence of commands specified by the memory vendor. When you have sent these commands to the memory, you can write to the memc_cmd Register, offset `0x4` with the GO command, `0x0`. See *Memory Controller Command Register* on page 3-8.
4. Poll the memc_status Register until the value of `0x1` is returned, READY, signifying that the device is ready to accept AXI accesses to the SDRAM.

Arc 6 to 5 If you want to reconfigure either the DDR2 DMC or SDRAM, you must first write to the memc_cmd Register, offset `0x4`, with the Pause command, `0x3`, and poll the memc_status Register until the value of `0x2` is returned, Paused. Then you can write to the memc_cmd Register with the Configure command, `0x4` and poll the memc_status Register until the value of `0x0` is returned, Config. See *Memory Controller Status Register* on page 3-7 and *Memory Controller Command Register* on page 3-8.

Arc 6 to 7 If auto_power_down is set in the memory_cfg Register, see *Memory Configuration Register* on page 3-11, then this arc is automatically taken when the SDRAM has been idle for power_down_prd **mclk** cycles.

Arc 7 to 6 When an SDRAM access command has been received in the **mclk** domain, this arc is taken.

Arc 6 to 8 You can take this arc under either hardware or software control:

- To take this arc under software control:
  1. Issue the Pause command.
  2. Poll for the Paused state.
  3. Issue the Sleep command.
- To take this arc under hardware control, use the AXI low-power interface to request a low-power state.

Arc 6 to 9 The same as arc 6 to 8, but additionally stop the **mclk** domain clock.

Arc 6 to 10 The same as arc 6 to 8, but additionally stop the **aclk** domain clock.

Arc 6 to 11 The same as arc 6 to 8, but additionally stop both the **mclk** and the **aclk** domain clocks.

Arc 6 to 12 The same as arc 6 to 8, but additionally stop the **mclk** domain clock and remove power from the **aclk** power domain. This can only be done if the DDR2 DMC implementation has separate power domains for **aclk** and **mclk**.

Arc 6 to 13 The same as arc 6 to 8, but additionally remove power from the **aclk** power domain. This can only be done if the DDR2 DMC implementation has separate power domains for **aclk** and **mclk**.

Arc 8 to 6 You can take this arc under either hardware or software control:

- To take this arc under software control:
  1. Issue the Wakeup command to the memc_cmd Register.
  2. Poll the memc_status Register for the Paused state.
  3. Issue the Go command and poll for the Ready state.
- To take this arc under hardware control, use the AXI low-power interface to bring the device out of a low-power state.

Arc 9 to 6 The same as arc 8 to 6, but you must first start the **mclk** domain clock.

Arc 10 to 6 The same as arc 8 to 6, but you must first start the **aclk** domain clock.

Arc 11 to 6 The same as arc 8 to 6, but you must first start both the **aclk** and **mclk** domain clocks.

Arc 12 to 14 Apply power to the **aclk** power domain.

Arc 14 to 16 Assert reset to the **aclk** reset domain.

Arc 16 to 9 De-assert reset to the **aclk** reset domain.

Arc 13 to 15 Apply power to the **aclk** power domain.

Arc 15 to 17 Assert reset to the **aclk** reset domain.

Arc 17 to 8 De-assert reset to the **aclk** reset domain.

Chapter 3
Programmer's Model

This chapter describes the DDR2 DMC (PL341) registers and provides information for programming the device. It contains the following sections:

3.1 About the programmer's model

Figure 3-1 shows the register map.

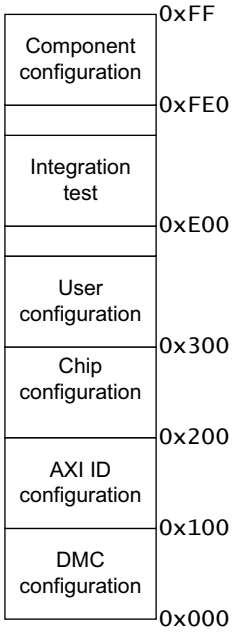


Figure 3-1 Register map

3.2 Register summary

Figure 3-2 shows the configuration register map.

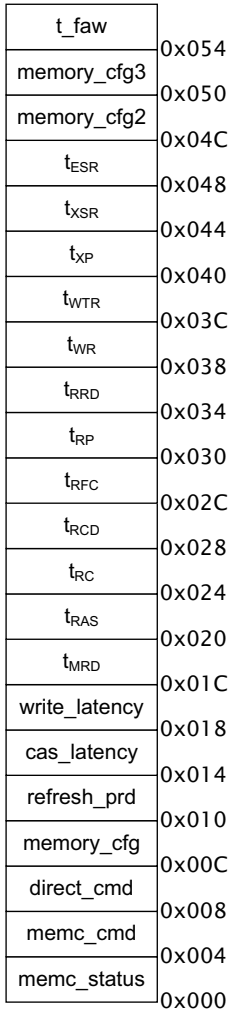


Figure 3-2 Configuration register map

Figure 3-3 shows the AXI ID configuration registers map.

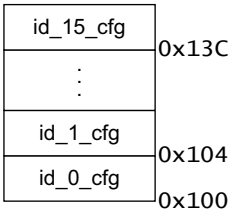


Figure 3-3 AXI ID configuration register map

Figure 3-4 shows the chip configuration register map.

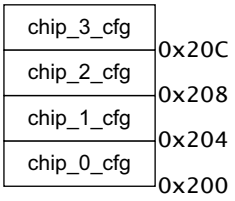


Figure 3-4 Chip configuration registers map

Figure 3-5 shows the user configuration memory map.

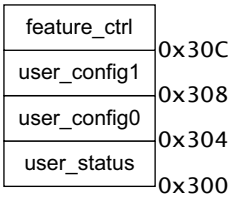


Figure 3-5 User configuration memory map

Figure 3-6 shows the integration test register map.

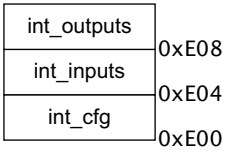


Figure 3-6 Integration test register map

Figure 3-7 shows the identification register map.

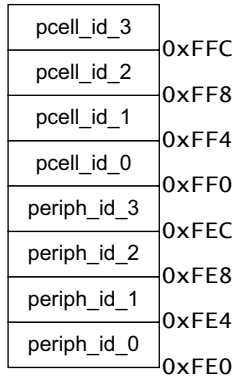


Figure 3-7 Identification register map

Table 3-1 lists the memory controller registers.

Table 3-1 Memory controller register summary

Name	Base offset	Type	Reset value	Description
memc_status	0x000	RO	-[a]	*Memory Controller Status Register* on page 3-7
memc_cmd	0x004	WO	-	*Memory Controller Command Register* on page 3-8
direct_cmd	0x008	WO	-	*Direct Command Register* on page 3-10
memory_cfg	0x00C	RW	0x00010020	*Memory Configuration Register* on page 3-11
refresh_prd	0x010	RW	0x00000A2C	*Refresh Period Register* on page 3-12
cas_latency	0x014	RW	0x0000000A	*CAS Latency Register* on page 3-13
write_latency	0x018	RO	0x00000004	*write_latency Register* on page 3-14
t_mrd	0x01C	RW	0x00000002	*t_mrd Register* on page 3-14
t_ras	0x020	RW	0x0000000E	*t_ras Register* on page 3-15
t_rc	0x024	RW	0x00000012	*t_rc Register* on page 3-15
t_rcd	0x028	RW	0x00000305	*t_rcd Register* on page 3-16
t_rfc	0x02C	RW	0x00002123	*t_rfc Register* on page 3-16
t_rp	0x030	RW	0x00000305	*t_rp Register* on page 3-17
t_rrd	0x034	RW	0x00000004	*t_rrd Register* on page 3-18

Table 3-1 Memory controller register summary (continued)

Name	Base offset	Type	Reset value	Description
t_wr	0x038	RW	0x00000005	*t_wr Register* on page 3-18
t_wtr	0x03C	RW	0x00000004	*t_wtr Register* on page 3-19
t_xp	0x040	RW	0x00000002	*t_xp Register* on page 3-19
t_xsr	0x044	RW	0x00000027	*t_xsr Register* on page 3-20
t_esr	0x048	RW	0x00000014	*t_esr Register* on page 3-20
memory_cfg2	0x04C	RW	-[a]	*memory_cfg2 Register* on page 3-21
memory_cfg3	0x050	RW	0x00000007	*memory_cfg3 Register* on page 3-22
t_faw	0x054	RW	0x00000011	*t_faw Register* on page 3-23
id_n_cfg	0x100	RW	0x00000000	*id_<n>_cfg Registers* on page 3-23
chip_n_cfg	0x200	RW	0x0000FF00	*chip_<n>_cfg Registers* on page 3-24
user_status	0x300	RO	-	*user_status Register* on page 3-25
user_config0	0x304	WO	-	*user_config0 Register* on page 3-25
user_config1	0x308	WO	-	*user_config1 Register* on page 3-26
feature_ctrl	0x30C	RW	0x00000000	*feature_ctrl Register* on page 3-26
periph_id_n	0xFE0-0xFEC	RO	-	*Peripheral Identification Registers 0-3* on page 3-27
pcell_id_n	0xFF0-0xFFC	RO	-	*PrimeCell Identification Registers 0-3* on page 3-29

a. Dependent on configuration.

3.3 Register descriptions

This section describes the memory controller registers.

3.3.1 Memory Controller Status Register

The memc_status Register provides information on the configuration of the device, and also on the current state of the device. It cannot be read in either the Reset or POR states. Figure 3-8 shows the register bit assignments.

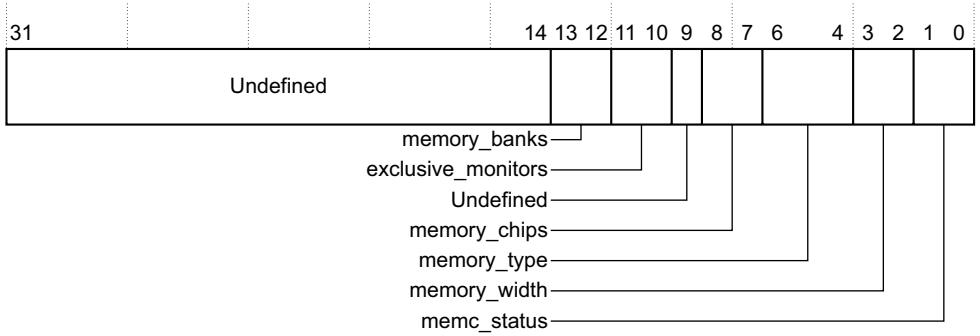


Figure 3-8 memc_status Register bit assignments

Table 3-2 lists the register bit assignments.

Table 3-2 memc_status Register bit assignments

Bits	Name	Function
[31:14]	-	Read undefined.
[13:12]	memory_banks	Returns the maximum number of banks per memory chip. This is a rendered option, and is static for a given configuration: 2'b00 = 4 banks 2'b01 = 2 banks, not supported 2'b10 = 1 banks, not supported 2'b11 = 8 banks.
[11:10]	exclusive_monitors	Returns the number of exclusive access monitor resources implemented in the device: 2'b00 = 0 monitors 2'b01 = 1 monitor 2'b10 = 2 monitors 2'b11 = 4 monitors.

Table 3-2 memc_status Register bit assignments (continued)

Bits	Name	Function
[9]	-	Undefined, read as zero.
[8:7]	memory_chips	Returns the number of different chip selects that the DMC supports: 2'b00 = 1 chip 2'b01 = 2 chips 2'b10 = 3 chips 2'b11 = 4 chips.
[6:4]	memory_type	Returns the SDRAM that the device supports: 3'b101 = DDR2 SDRAM.
[3:2]	memory_width	Returns the effective width of the external memory: 2'b00 = Reserved 2'b01 = 32-bit 2'b10 = 64-bit 2'b11 = 128-bit.
[1:0]	memc_status	Returns the state of the memory controller: 2'b00 = Config 2'b01 = Ready 2'b10 = Paused 2'b11 = Low_power.

3.3.2 Memory Controller Command Register

Writing to the memc_cmd Register enables the programmer's view FSM to be traversed. If a new command is received to change state, and a previous command to change state has not completed, the **pready** signal is held LOW until the new command can be carried out.

Figure 3-9 shows the register bit assignments.

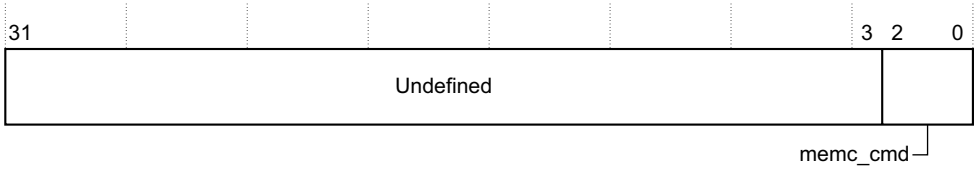


Figure 3-9 memc_cmd Register bit assignments

Table 3-3 lists the register bit assignments.

Table 3-3 memc_cmd Register bit assignments

Bits	Name	Function
[31:3]	-	Undefined, write as zero.
[2:0]	memc_cmd	Changes the state of the memory controller: 3'b000 = Go 3'b001 = Sleep 3'b010 = Wakeup 3'b011 = Pause 3'b100 = Configure 3'b111 = Active_Pause.

Note

- Active_Pause puts the device into the Paused state without draining the arbiter queue. This enables you to enter low-power state to change configuration settings such as memory frequency or timing register values without requiring co-ordination between masters in a multi-master system.
- If the DMC is put into low-power state after using the Active_Pause command, you must not remove power from the device because this results in data loss and violation of the AXI protocol.
- The device does not issue refreshes while in the Config state. It is recommended therefore that you use low-power mode to make register updates because this ensures that the memory is put into self-refresh rather than entering the Config state when the memory contains valid data.
- If you entered the Paused state using the Active_Pause command, you must not attempt to move to the Config state using the configure command.

3.3.3 Direct Command Register

The direct_cmd Register passes commands to the external memory. The configuration of the DirectCmd register enables you to write to any type of Mode register supported by the external memory device, and also to generate NOP, Prechargeall, and Autorefresh commands. The DirectCmd Register therefore enables any initialization sequence that an external memory device might require. The only timing information associated with the DirectCmd Register are the command delays defined in the timing registers. Therefore, if an initialization sequence requires additional delays between commands, they must be timed by the master driving the initialization sequence.

This register can only be written to in the Config state. Figure 3-10 shows the register bit assignments.

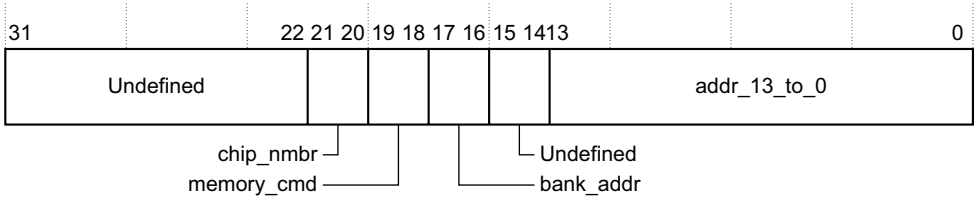


Figure 3-10 direct_cmd Register bit assignments

Table 3-4 lists the register bit assignments.

Table 3-4 direct_cmd Register bit assignments

Bits	Name	Function
[31:22]	-	Undefined, write as zero.
[21:20]	chip_nmbr	Bits mapped to external memory chip address bits.
[19:18]	memory_cmd	Determines the command required: 2'b00 = Prechargeall 2'b01 = Autorefresh 2'b10 = Modereg or Extended modereg access 2'b11 = NOP.
[17:16]	bank_addr	Bits mapped to external memory bank address bits when command is Modereg access.
[15:14]	-	Undefined, write as zero.
[13:0]	addr_13_to_0	Bits mapped to external memory address bits [13:0] when command is Modereg access.

3.3.4 Memory Configuration Register

The memory_cfg Register configures the memory controller. It can only be read from and written to in the Config or Low-power state. Figure 3-11 shows the register bit assignments.

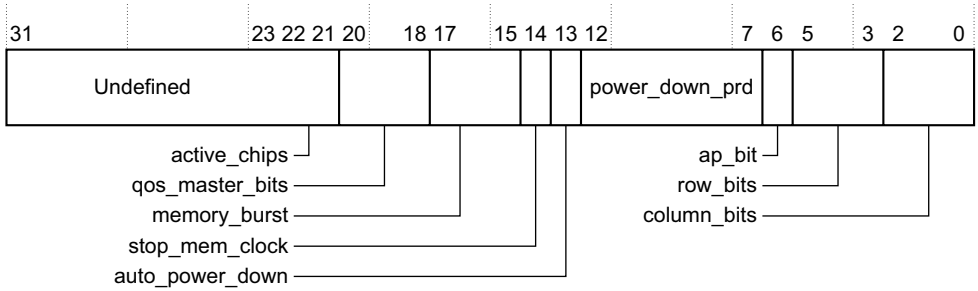


Figure 3-11 memory_cfg Register bit assignments

Table 3-5 lists the register bit assignments.

Table 3-5 memory_cfg Register bit assignments

Bits	Name	Function
[31:23]	-	Read undefined, write as zero.
[22:21]	active_chips	Enables the refresh command generation for the number of memory chips. It is only possible to generate commands up to and including the number of chips in the configuration that the memc_status Register defines: 2'b00 = 1 chip 2'b01 = 2 chips 2'b10 = 3 chips 2'b11 = 4 chips.
[20:18]	qos_master_bits	Encodes the four bits of the 8-bit AXI ARID that select one of the 16 QOS values: 3'b000 = ARID[3:0] 3'b001 = ARID[4:1] 3'b010 = ARID[5:2] 3'b011 = ARID[6:3] 3'b100 = ARID[7:4] 3'b101 = ARID[8:5] 3'b110 = ARID[9:6] 3'b111 = ARID[10:7].

Table 3-5 memory_cfg Register bit assignments (continued)

Bits	Name	Function
[17:15]	memory_burst	Encodes the number of data accesses that are performed to the SDRAM for each Read and Write command: 3'b010 = Burst 4. This value must also be programmed into the SDRAM mode register using the direct_cmd register at offset `0x8`, and must match it.
[14]	-	Reserved. Ignored for writes, read as zero.
[13]	auto_power_down	When this is set, the memory interface automatically places the SDRAM into power-down state by deasserting **CKE** when the command FIFO has been empty for PowerDownPrd memory clock cycles.
[12:7]	power_down_prd	Number of memory clock cycles for auto power-down of the SDRAM.
[6]	-	Reserved. Ignored for writes, read as zero.
[5:3]	row_bits	Encodes the number of bits of the AXI address that comprise the row address: 3'b000 = 11 bits 3'b001 = 12 bits 3'b010 = 13 bits 3'b011 = 14 bits 3'b100 = 15 bits 3'b101 = 16 bits.
[2:0]	column_bits	Encodes the number of bits of the AXI address that comprise the column address: 3'b000 = Reserved. 3'b001 = 9 bits. 3'b010 = 10 bits. 3'b011 = 11 bits. This means that A0-A9, and A11 are used for column address because A10 is a dedicated AP bit. 3'b100 = Reserved.

3.3.5 Refresh Period Register

The refresh_prd Register sets the memory refresh period. It can only be read from and written to in the Config or Low-power state. Figure 3-12 on page 3-13 shows the register bit assignments.

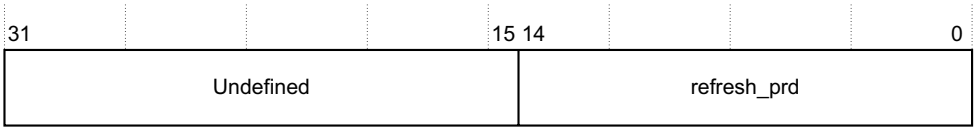


Figure 3-12 refresh_prd Register bit assignments

Table 3-6 lists the register bit assignments.

Table 3-6 refresh_prd Register bit assignments

Bits	Name	Function
[31:15]	-	Read undefined, write as zero
[14:0]	refresh_prd	Memory refresh period in memory clock cycles

3.3.6 CAS Latency Register

The cas_latency Register sets the CAS latency in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-13 shows the register bit assignments.

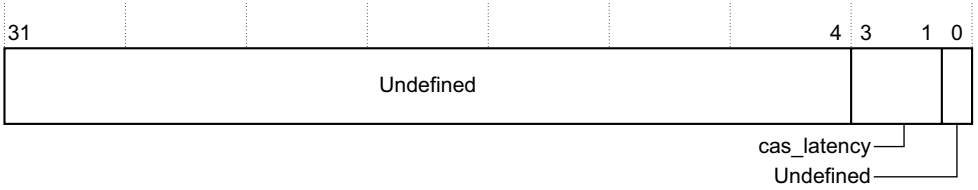


Figure 3-13 cas_latency Register bit assignments

Table 3-7 lists the register bit assignments.

Table 3-7 cas_latency Register bit assignments

Bits	Name	Function
[31:4]	-	Read undefined, write as zero.
[3:1]	cas_latency	CAS latency in memory clock cycles. Values between 2-6 are supported.
[0]	-	Undefined, read and write as zero.

3.3.7 write_latency Register

The write_latency Register indicates the write latency in memory clock cycles. It can only be read from in the Config or Low-power state, and is always cas_latency, in clock cycles, minus 1. Figure 3-14 shows the register bit assignments.

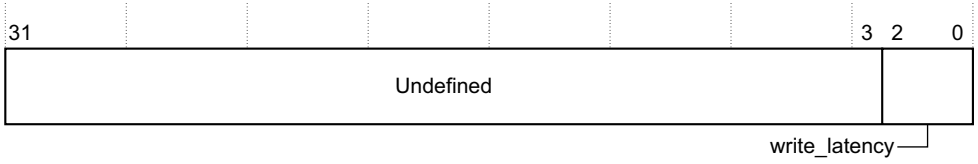


Figure 3-14 write_latency Register bit assignments

Table 3-8 lists the register bit assignments.

Table 3-8 write_latency Register bit assignments

Bits	Name	Function
[31:3]	-	Read undefined, write as zero
[2:0]	write_latency	Write latency in memory clock cycles

3.3.8 t_mrd Register

The t_mrd Register sets the mode register command time in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-15 shows the register bit assignments.

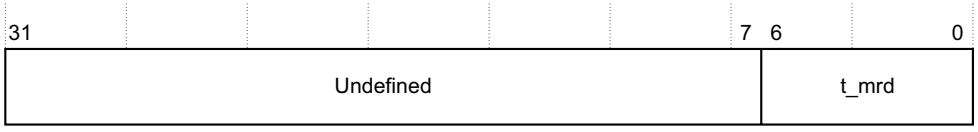


Figure 3-15 t_mrd Register bit assignments

Table 3-9 lists the register bit assignments.

Table 3-9 t_mrd Register bit assignments

Bits	Name	Function
[31:7]	-	Read undefined, write as zero
[6:0]	t_mrd	Sets mode register command time in memory clock cycles

3.3.9 t_ras Register

The t_ras Register sets the RAS to precharge delay in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-16 shows the register bit assignments.

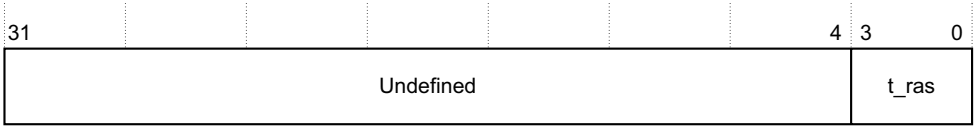


Figure 3-16 t_ras Register bit assignments

Table 3-10 lists the register bit assignments.

Table 3-10 t_ras Register bit assignments

Bits	Name	Function
[31:4]	-	Read undefined, write as zero
[3:0]	t_ras	Sets RAS to precharge delay in memory clock cycles

3.3.10 t_rc Register

The t_rc Register sets the Active bank x to Active bank x delay in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-17 shows the register bit assignments.

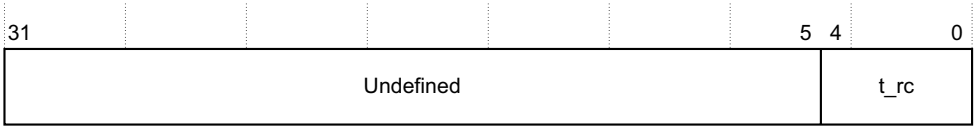


Figure 3-17 t_rc Register bit assignments

Table 3-11 lists the register bit assignments.

Table 3-11 t_rc Register bit assignments

Bits	Name	Function
[31:5]	-	Read undefined, write as zero
[4:0]	t_rc	Sets Active bank x to Active bank x delay in memory clock cycles

3.3.11 t_rcd Register

The t_rcd Register sets the RAS to CAS minimum delay in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-18 shows the register bit assignments.

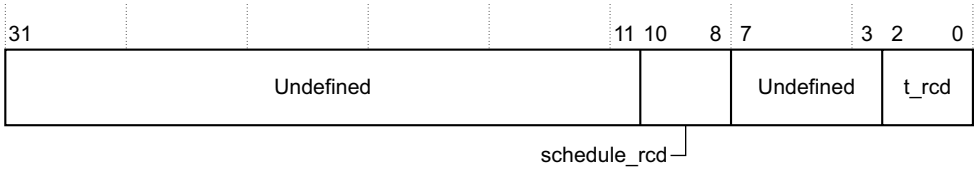


Figure 3-18 t_rcd Register bit assignments

Table 3-12 lists the register bit assignments.

Table 3-12 t_rcd Register bit assignments

Bits	Name	Function
[31:11]	-	Read undefined, write as zero.
[10:8]	schedule_rcd	Sets the RAS to CAS minimum delay in clock cycles minus 3. It is used as a scheduler delay.
[7:3]	-	Read undefined, write as zero.
[2:0]	t_rcd	Sets the RAS to CAS minimum delay in memory clock cycles.

3.3.12 t_rfc Register

The t_rfc Register sets the auto-refresh command time in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-19 shows the register bit assignments.

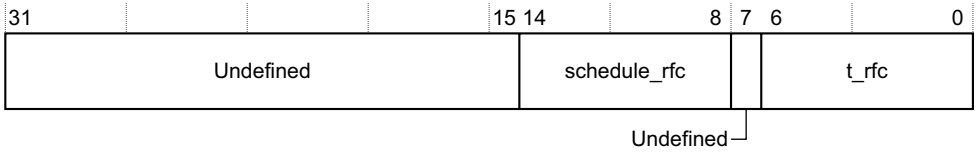


Figure 3-19 t_rfc Register bit assignments

Table 3-13 lists the register bit assignments.

Table 3-13 t_rfc Register bit assignments

Bits	Name	Function
[31:15]	-	Read undefined, write as zero.
[14:8]	schedule_rfc	Sets the autorefresh command time in clock cycles minus 3. It is used as a scheduler delay.
[7]	-	Read undefined, write as zero.
[6:0]	t_rfc	Sets the auto-refresh command time in memory clock cycles.

3.3.13 t_rp Register

The t_rp Register sets the precharge to RAS delay in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-20 shows the register bit assignments.

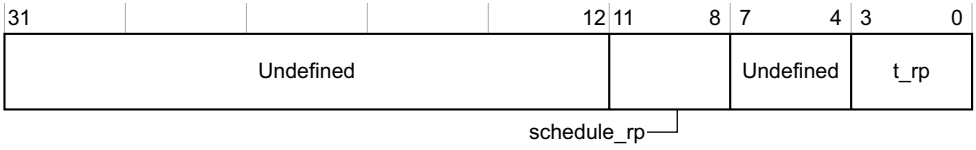


Figure 3-20 t_rp Register bit assignments

Table 3-14 lists the register bit assignments.

Table 3-14 t_rp Register bit assignments

Bits	Name	Function
[31:12]	-	Read undefined, write as zero.
[11:8]	schedule_rp	Sets the precharge to RAS delay in clock cycles, minus 3. It is used as a scheduler delay.
[7:4]	-	Read undefined, write as zero.
[3:0]	t_rp	Sets the precharge to RAS delay in memory clock cycles.

3.3.14 t_rrd Register

The t_rrd Register sets the Active bank x to Active bank y delay in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-21 shows the register bit assignments.

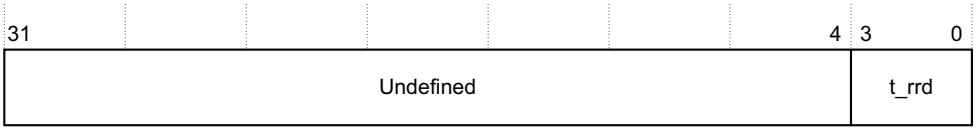


Figure 3-21 t_rrd Register bit assignments

Table 3-15 lists the register bit assignments.

Table 3-15 t_rrd Register bit assignments

Bits	Name	Function
[31:4]	-	Read undefined, write as zero
[3:0]	t_rrd	Sets Active bank x to Active bank y delay in memory clock cycles

3.3.15 t_wr Register

The t_wr Register sets the write to precharge delay in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-22 shows the register bit assignments.

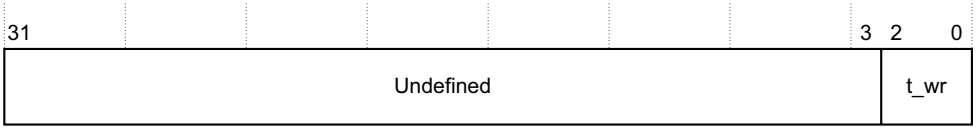


Figure 3-22 t_wr Register bit assignments

Table 3-16 lists the register bit assignments.

Table 3-16 t_wr Register bit assignments

Bits	Name	Function
[31:3]	-	Read undefined, write as zero
[2:0]	t_wr	Sets the write to precharge delay in memory clock cycles

3.3.16 t_wtr Register

The t_wtr Register sets the write to read delay in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-23 shows the register bit assignments.

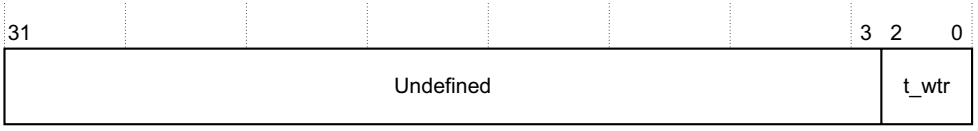


Figure 3-23 t_wtr Register bit assignments

Table 3-17 lists the register bit assignments.

Table 3-17 t_wtr Register bit assignments

Bits	Name	Function
[31:3]	-	Read undefined, write as zero
[2:0]	t_wtr	Sets the write to read delay in memory clock cycles

3.3.17 t_xp Register

The t_xp Register sets exit power-down command time in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-24 shows the register bit assignments.

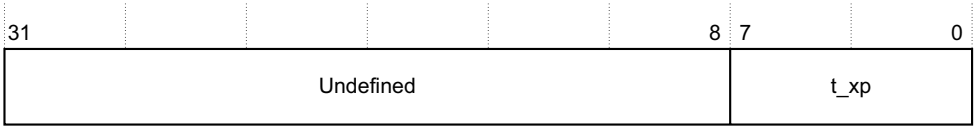


Figure 3-24 t_xp Register bit assignments

Table 3-18 lists the register bit assignments.

Table 3-18 t_xp Register bit assignments

Bits	Name	Function
[31:8]	-	Read undefined, write as zero
[7:0]	t_xp	Sets the exit power-down command time in memory clock cycles

3.3.18 t_xsr Register

The t_xsr Register sets exit self-refresh command time in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-25 shows the register bit assignments.

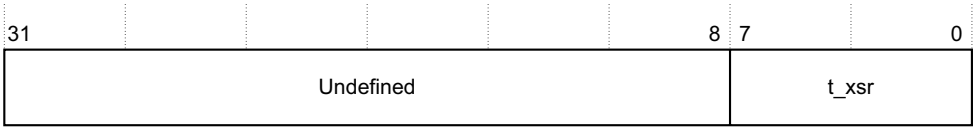


Figure 3-25 t_xsr Register bit assignments

Table 3-19 lists the register bit assignments.

Table 3-19 t_xsr Register bit assignments

Bits	Name	Function
[31:8]	-	Read undefined, write as zero.
[7:0]	t_xsr	Sets the exit self-refresh command time in memory clock cycles.

3.3.19 t_esr Register

The t_esr Register sets self-refresh command time in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-26 shows the register bit assignments.

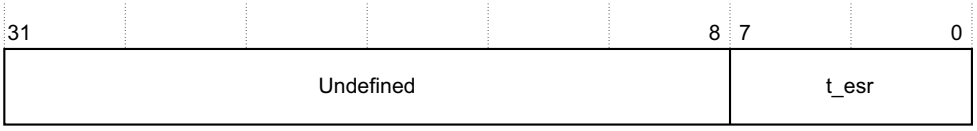


Figure 3-26 t_esr Register bit assignments

Table 3-20 lists the register bit assignments.

Table 3-20 t_esr Register bit assignments

Bits	Name	Function
[31:8]	-	Read undefined, write as zero
[7:0]	t_esr	Sets the self-refresh command time in memory clock cycles

3.3.20 memory_cfg2 Register

The read/write memory_cfg2 register configures the memory controller. It can only be read from and written to in the Config or Low-power state. Figure 3-27 shows the register bit assignments.

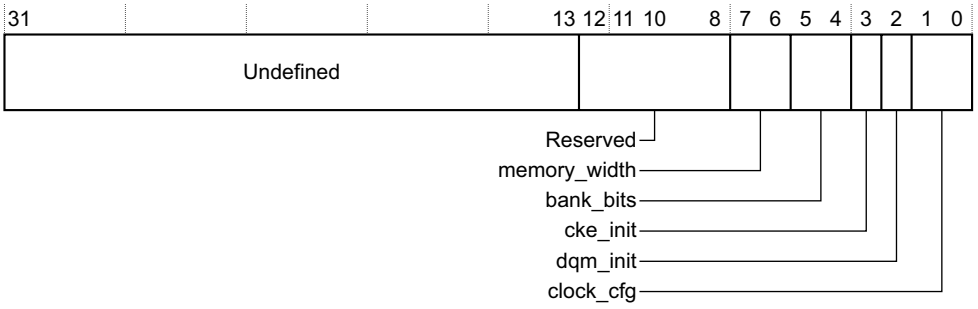


Figure 3-27 memory_cfg2 Register bit assignments

Table 3-21 lists the register bit assignments.

Table 3-21 memory_cfg2 Register bit assignments

Bits	Name	Function
[31:13]	-	Read undefined, write as zero.
[12:8]	-	Reserved, bits used in PL340.
[7:6]	memory_width	Encodes the physical memory width of the attached device. This is half the memory_width in the memc_status register, see *Memory Controller Status Register* on page 3-7. The value is dependent on the rendered configuration of the implementation: 2'b00 = 16-bit 2'b01 = 32-bit 2'b10 = 64-bit 2'b11 = reserved.
[5:4]	bank_bits	Encodes the number of bits of the AXI address that comprise the bank address. The value is dependent on the rendered configuration of the implementation: 2'b10 = 0 bits, not supported by PL341 2'b01 = 1 bits, not supported by PL341 2'b00 = 2 bits 2'b11 = 3 bits.

Table 3-21 memory_cfg2 Register bit assignments (continued)

Bits	Name	Function
[3]	cke_init	State of **CKE** when **mresetn** is de-asserted.
[2]	dqm_init	State of **DQM** when **mresetn** is de-asserted.
[1:0]	clock_cfg	Encodes the clocking scheme: 2'b00 = **aclk** and **mclk** are asynchronous 2'b01 = **aclk** and **mclk** are synchronous, and **aclk** is the same frequency or slower than **mclk** 2'b10 = reserved 2'b11 = **aclk** and **mclk** are synchronous, and **aclk** is greater than **mclk**.

3.3.21 memory_cfg3 Register

The memory_cfg3 Register determines the number of acceptable outstanding refreshes on a chip before a refresh timeout occurs and refreshes are raised to the highest priority in the queue. It can only be read from and written to in the Config or Low-power state. Figure 3-28 shows the register bit assignments.

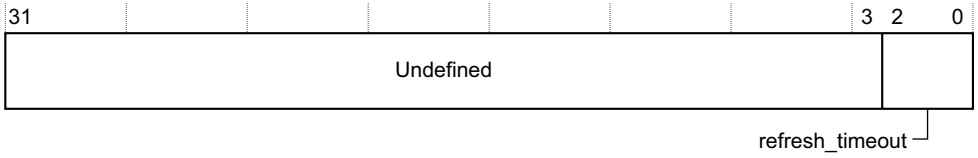


Figure 3-28 memory_cfg3 Register bit assignments

Table 3-22 lists the register bit assignments.

Table 3-22 memory_cfg3 Register bit assignments

Bits	Name	Function
[31:3]	-	Read undefined, write as zero.
[2:0]	refresh_timeout	Determines the number of acceptable outstanding refreshes on a chip before a refresh timeout occurs and refreshes are raised to the highest priority in the queue.

3.3.22 t_faw Register

The t_faw Register sets four bank activate time in memory clock cycles. It can only be read from and written to in the Config or Low-power state. Figure 3-29 shows the register bit assignments.

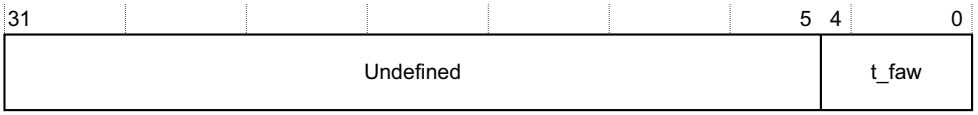


Figure 3-29 t_faw Register bit assignments

Table 3-23 lists the register bit assignments.

Table 3-23 t_faw Register bit assignments

Bits	Name	Function
[31:13]	-	Read undefined, write as zero.
[12:8]	schedule_faw	t_faw in clock cycles, minus 3. Used as a scheduler delay.
[7:5]	-	Read undefined, write as zero.
[4:0]	t_faw	Four-bank activate period in clock cycles.

3.3.23 id_<n>_cfg Registers

The id_<n>_cfg Registers are 16 registers that set the QoS and span address locations `0x100-0x13C`. The registers can only be read from and written to in the Config or Low-power states. Figure 3-30 shows the register bit assignments.

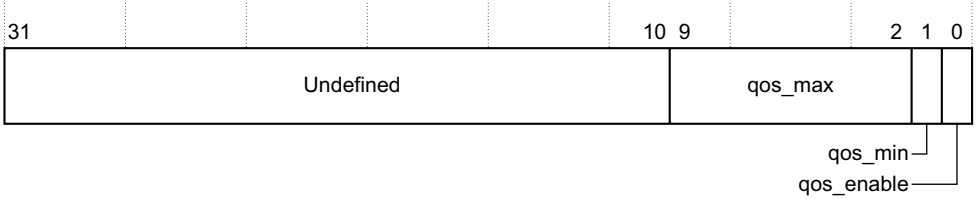


Figure 3-30 id_<n>_cfg Registers bit assignments

Table 3-24 lists the register bit assignments.

Table 3-24 id_<n>_cfg Registers bit assignments

Bits	Name	Function
[31:10]	-	Read undefined, write as zero
[9:2]	qos_max	Sets a maximum QoS
[1]	qos_min	Sets a minimum QoS
[0]	qoa_enable	Enables a QoS value to be applied to memory reads from address ID n

3.3.24 chip_<n>_cfg Registers

The chip_<n>_cfg Registers are registers that set up the external memory device configuration. The number of external chips supported, and therefore the number of these registers, depends on your configuration.

They span address locations `0x200-0x20C`. There is one register per memory device. The registers configure the base address and address decoding method. The registers can only be read from and written to in the Config or Low-power states.

Figure 3-31 shows the register bit assignments.

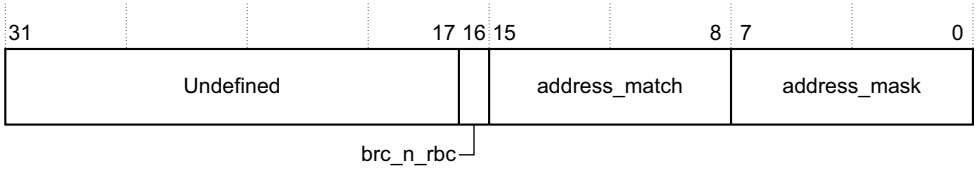


Figure 3-31 chip_<n>_cfg Registers bit assignments

Table 3-25 lists the register bit assignments.

Table 3-25 chip_<n>_cfg Registers bit assignments

Bits	Name	Function
[31:17]	-	Read undefined, write as zero.
[16]	brc_n_rbc	Selects the memory organization as decoded from the AXI address: 1'b0 = Row, bank, column organization 1'b1 = Bank, row, column organization.
[15:8]	address_match	Comparison value for AXI address bits [31:24] to determine the chip that is selected.
[7:0]	address_mask	The mask for AXI address bits [31:24] to determine the chip that is selected: 1 = corresponding address bit is to be used for comparison.

3.3.25 user_status Register

This register returns the state of the **user_status[7:0]** primary inputs.

Table 3-26 lists the register bit assignments.

Table 3-26 user_status Registers bit assignments

Bits	Name	Function
[7:0]	user_status	Value of primary input pins

3.3.26 user_config0 Register

This register sets the value of the **user_config0[7:0]** primary outputs and can be written to in all states.

Table 3-27 lists the register bit assignments.

Table 3-27 user_config0 Registers bit assignments

Bits	Name	Function
[7:0]	user_config0	Sets primary output pins

3.3.27 user_config1 Register

This register sets the value of the **user_config1[7:0]** primary outputs and can be written to in all states.

Table 3-28 lists the register bit assignments.

Table 3-28 user_config1 Registers bit assignments

Bits	Name	Function
[7:0]	user_config1	Sets primary output pins

3.3.28 feature_ctrl Register

The feature_ctrl register can only be read from and written to in the Config and Low-power states. Figure 3-32 shows the register bit assignments.

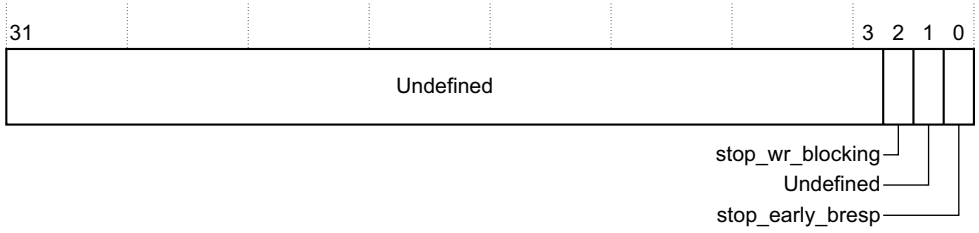


Figure 3-32 feature_ctrl Register bit assignments

Table 3-29 lists the register bit assignments.

Table 3-29 feature_ctrl Registers bit assignments

Bits	Name	Function
[31:3]	-	Undefined, write as zero.
[2]	stop_wr_blocking	Controls the write merge up to memory burst: 1'b0 = write transfers initiated when a full write memory burst is ready 1'b1 = write transfers initiated when write data is available.
[1]	-	Undefined, write as zero.
[0]	stop_early_bresp	Controls the early write response feature: 1'b0 = early write responses enabled 1'b1 = early write responses disabled.

3.3.29 Peripheral Identification Registers 0-3

The periph_id Registers are four eight-bit read-only registers, that span address locations `0xFE0-0xFEC`. The registers can conceptually be treated as a single register that holds a 32-bit peripheral ID value. An external master reads them to determine the version of the device.

Table 3-30 lists the register bit assignments.

Table 3-30 periph_id Register bit assignments

Bits	Name	Description
[31:24]	integration_cfg	Configuration options are peripheral-specific. See *Peripheral Identification Register 3* on page 3-29.
[23:20]	-	The peripheral revision number is revision-dependent.
[19:12]	designer	Designer's ID number. This is `0x41` for ARM.
[11:0]	part_number	Identifies the peripheral. The part number for PL341 is `0x341`.

Figure 3-33 shows the correspondence between bits of the periph_id registers and the conceptual 32-bit Peripheral ID Register.

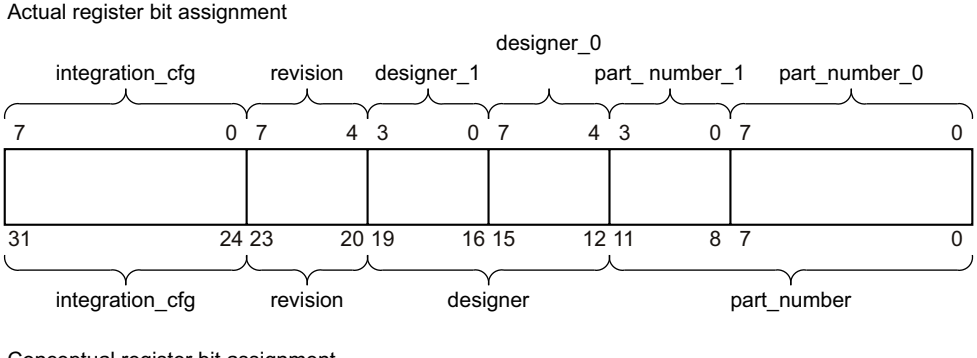


Figure 3-33 periph_id Register bit assignments

The following sections describe the periph_id Registers:

- *Peripheral Identification Register 0* on page 3-28
- *Peripheral Identification Register 1* on page 3-28
- *Peripheral Identification Register 2* on page 3-28
- *Peripheral Identification Register 3* on page 3-29.

Peripheral Identification Register 0

The periph_id_0 Register is hard-coded and the fields within the register determine the reset value. Table 3-31 lists the register bit assignments.

Table 3-31 periph_id_0 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:0]	part_number_0	These bits read back as `0x41`

Peripheral Identification Register 1

The periph_id_1 Register is hard-coded and the fields within the register determine the reset value. Table 3-32 lists the register bit assignments.

Table 3-32 periph_id_1 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:4]	designer_0	These bits read back as `0x1`
[3:0]	part_number_1	These bits read back as `0x3`

Peripheral Identification Register 2

The periph_id_2 Register is hard-coded and the fields within the register determine the reset value. Table 3-33 lists the register bit assignments.

Table 3-33 periph_id_2 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined.
[7:4]	revision	These bits read back as the revision number. This can be between 0 and 15 and is `0x0` for r0p0.
[3:0]	designer_1	These bits read back as `0x4`.

Peripheral Identification Register 3

The periph_id_3 register is hard-coded and the fields within the register determine the reset value. Table 3-34 lists the register bit assignments.

Table 3-34 periph_id_3 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:4]	-	Reserved for future use, read undefined
[3:0]	Customer Modified	Customer modified number, 0 from ARM

3.3.30 PrimeCell Identification Registers 0-3

The pcell_id Registers are four 8-bit wide read-only registers, that span address locations `0xFF0-0xFFC`. The registers can be treated conceptually as a single register that holds a 32-bit PrimeCell identification value. You can use the register for automatic BIOS configuration. The pcell_id Register is set to `0xB105F00D`.

You can access the register with one wait state.

Figure 3-34 shows the register bit assignments.

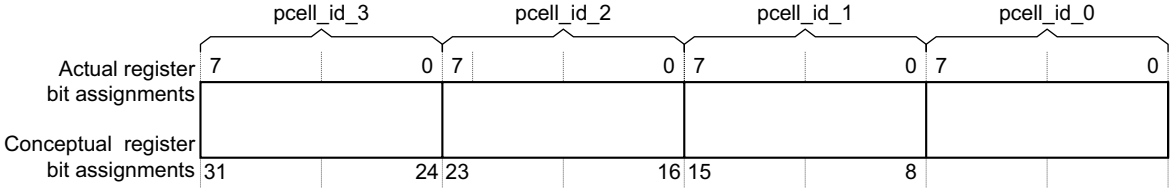


Figure 3-34 pcell_id Register bit assignments

Table 3-35 lists the register bit assignments.

Table 3-35 pcell_id Register bit assignments

Component ID register		CompID0-3 register		
Bits	**Reset value**	**Register**	**Bits**	**Description**
-	-	pcell_id_3	[31:8]	Read undefined
[31:24]	0xB1	pcell_id_3	[7:0]	These bits read back as 0xB1
-	-	pcell_id_2	[31:8]	Read undefined
[23:16]	0x05	pcell_id_2	[7:0]	These bits read back as 0x05
-	-	pcell_id_1	[31:8]	Read undefined
[15:8]	0xF0	pcell_id_1	[7:0]	These bits read back as 0xF0
-	-	pcell_id_0	[31:8]	Read undefined
[7:0]	0x0D	pcell_id_0	[7:0]	These bits read back as 0x0D

Chapter 4
Programmer's Model for Test

This chapter describes the additional logic for functional verification and production testing. It contains the following section:

- *Integration test registers* on page 4-2.

4.1 Integration test registers

Test registers are provided for integration testing.

Figure 4-1 shows the Integration Test Register map.

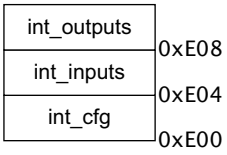


Figure 4-1 Integration Test Register map

Table 4-1 lists the integration test registers.

Table 4-1 Memory controller test register summary

Name	Base offset	Type	Reset value	Description
int_cfg	0xE00	RW	0x0	*Integration Configuration Register*
int_inputs	0xE04	RO	-[a]	*Integration Inputs Register* on page 4-3
int_outputs	0xE08	WO	-	*Integration Outputs Register* on page 4-4

a. Dependent on tie-off ports.

4.1.1 Integration Configuration Register

The read/write int_cfg Register selects the integration test registers. This register is only for test. It can only be read and written in Config state.

Figure 4-2 shows the register bit assignments.

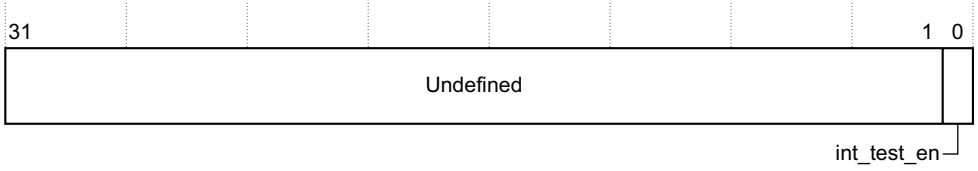


Figure 4-2 int_cfg Register bit assignments

Table 4-2 lists the register bit assignments.

Table 4-2 int_cfg Register bit assignments

Bits	Name	Function
[31:1]	Undefined	Read undefined, write as zero
[0]	int_test_en	When set, outputs are driven from the integration test registers, and input integration register values show external port states

4.1.2 Integration Inputs Register

The read-only int_inputs Register enables an external master to access the inputs of the device using the APB interface. This register is only for test.

Figure 4-3 shows the register bit assignments.

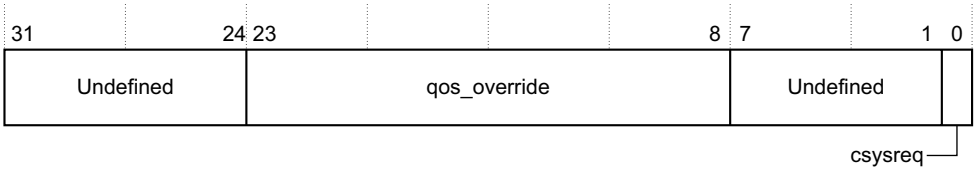


Figure 4-3 int_inputs Register bit assignments

Table 4-3 lists the register bit assignments.

Table 4-3 int_inputs Register bit assignments

Bits	Name	Function
[31:24]	-	Read undefined
[23:8]	qos_override	Returns the value of the qos_override external input
[7:1]	-	Read undefined
[0]	csysreq	Returns the value of the **csysreq** external input

4.1.3 Integration Outputs Register

The write-only int_outputs Register enables an external master to access the outputs of the device using the APB interface. It can only be read in Config state.

Figure 4-4 shows the register bit assignments.

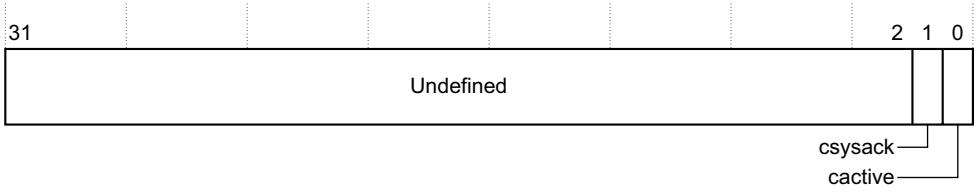


Figure 4-4 int_outputs Register bit assignments

Table 4-4 lists the register bit assignments.

Table 4-4 int_outputs Register bit assignments

Bits	Name	Function
[31:2]	-	Undefined, write as zero
[1]	csysack	Drives the **csysack** external output
[0]	cactive	Drives the **cactive** external output

Appendix A
Signal Descriptions

This chapter describes the AMBA AXI, AMBA APB, and module-specific non-AMBA signals used with the DDR2 DMC. It contains the following sections:

A.1 Clock and reset signals

Table A-1 lists the clock and reset signals.

Table A-1 Clock and reset signals

Signal	Type	Source/ destination	Description
mclk	Input	Clock source	Clock for **mclk** domain.
mclkn	Input	Clock source	Optional[a] clock for pad interface block.
mclkx2	Input	Clock source	Optional[a] clock for pad interface block.
mclkx2n	Input	Clock source	Optional[a] clock for pad interface block.
mresetn	Input	Reset source	Reset for **mclk** domain. This signal is active LOW.

a. These clocks are required if you use the delivered pad interface block.

A.2 Miscellaneous signals

Table A-2 lists the miscellaneous signals.

Table A-2 Miscellaneous signals

Signal	Type	Source	Description
sync	Input	Tie-off pin	When HIGH, indicates that **aclk** is synchronous to **mclk**. Otherwise, they are asynchronous.
cke_init	Input	Tie-off pin	The **cke** output port to the external memory resets to this value.
dqm_init	Input	Tie-off pin	The **dqm** output ports to the external memory reset to this value.
memory_width[1:0]	Input	Tie-off pin	These configure the external memory width. See Table 3-2 on page 3-7 for the definition of these two pins. **Note** You can use each PL341 configuration at half of its memory width, by setting the memory_width tie-off, provided that: • the new memory width is not less than 16 bits • the effective memory width is not less than half the AXI interface width.
bank_bits	Input	Tie-off pin	Encodes number of bits of AXI address that comprise the bank address. See the memory_banks field in Table 3-2 on page 3-7.
rst_bypass	Input	Tie-off pin	This signal is used for ATPG testing only.
se	Input	Scan test	Scan test enable.
dft_en_clk_out	Input	Scan test	Forces activation of **clk_out** for use in scan mode.
asetbok	Output	PL341 memif	Flags safe cycles to update the DLL control signals.
user_config0[7:0]	Output	External control logic	General purpose APB-accessible control pins. For example, you can use this for DLL control.
user_config1[7:0]	Output	External control logic	General purpose APB-accessible control pins. For example, you can use this for DLL control.
user_status[7:0]	Input	External control logic	General-purpose APB-accessible input pin.

A.3 Pad interface signals

Table A-3 lists the APB signal names.

Table A-3 Pad interface signals

Signal	Type	Source/ destination	Description
add	Output	External memory	Address to SDRAM
ap	Output	External memory	Auto precharge bit
ba[BANKWIDTH:0][a]	Output	External memory	Bank select
cas_n	Output	External memory	Column address strobe
cke	Output	External memory	Clock enable
clk_out[MEMORIES-1:0][b]	Output	External memory	Memory clock
cs_n[MEMORIES-1:0]	Output	External memory	Chip select
data_en	Output	External memory	Data direction enable
dq_in[MEMWIDTH-1:0][c]	Input	External memory	Data bus input
dq_out[MEMWIDTH-1:0]	Output	External memory	Data bus output
dqm[MEMBYTES-1:0][d]	Output	External memory	Data bus mask bits
dqs_in_<n>	Input	External memory	Data strobe in, DDR only
dqs_in_n_<n>	Input	External memory	Data strober bar, DDR only
dqs_out_<n>	Output	External memory	Data strobe out, DDR only
od	Output	External memory	Memory IO termination control
odt_read	Output	Output pad IO	Device IO termination control
ras_n	Output	External memory	Row address strobe
we_n	Output	External memory	Write enable

a. The BANKWIDTH value depends on the number of banks in the configuration.
b. MEMORIES is the number of chip selects.
c. MEMWIDTH is the data width of the external memory bus in bits.
d. MEMBYTES is the data width of the external memory bus in bytes.

Glossary

This glossary describes some of the terms used in this manual. Where terms can have several meanings, the meaning presented here is intended.

Abort

A mechanism that indicates to a core that it must halt execution of an attempted illegal memory access. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory. An abort is classified as a Prefetch Abort, a Data Abort, or an External Abort.

See also Data Abort, External Abort, and Prefetch Abort.

Advanced eXtensible Interface (AXI)

This is a bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure. The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

Advanced High-performance Bus (AHB)
The AMBA Advanced High-performance Bus system connects embedded processors such as an ARM core to high-performance peripherals, DMA controllers, on-chip memory, and interfaces. It is a high-speed, high-bandwidth bus that supports multi-master bus management to maximize system performance.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)
AMBA is the ARM open standard for multi-master on-chip buses, capable of running with multiple masters and slaves. It is an on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules. AHB conforms to this standard.

See also Advanced High-performance Bus and AHB-Lite.

Advanced Peripheral Bus (APB)
The AMBA Advanced Peripheral Bus is a simpler bus protocol than AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

See also Advanced High-performance Bus.

AHB *See* Advanced High-performance Bus.

Aligned Refers to data items stored so that their address is divisible by the highest power of two that divides their size. Aligned words and halfwords therefore have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore refer to addresses that are divisible by four and two respectively. The terms byte-aligned and doubleword-aligned are defined similarly.

AMBA *See* Advanced Microcontroller Bus Architecture.

APB *See* Advanced Peripheral Bus.

Architecture The organization of hardware and/or software that characterizes a processor and its attached components, and enables devices with similar characteristics to be grouped together when describing their behavior, for example, Harvard architecture, instruction set architecture, ARMv6 architecture.

AXI *See* Advanced eXtensible Interface.

Big-endian Memory organization where the least significant byte of a word is at a higher address than the most significant byte.

See also Little-endian and Endianness.

Central Processing Unit (CPU)
The part of a processor that contains the ALU, the registers, and the instruction decode logic and control circuitry. Also commonly known as the processor core.

Clock gating Gating a clock signal for a macrocell with a control signal, and using the modified clock that results to control the operating state of the macrocell.

Coprocessor A processor that supplements the main CPU. It carries out additional functions that the main CPU cannot perform. Usually used for floating-point math calculations, signal processing, or memory management.

CPU *See* Central Processing Unit.

Doubleword A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.

Endianness Byte ordering. The scheme that determines the order in which successive bytes of a data word are stored in memory.

See also Little-endian and Big-endian.

Little-endian Memory organization where the least significant byte of a word is at a lower address than the most significant byte.

See also Big-endian and Endianness.

Processor A contraction of microprocessor. A processor includes the CPU or core, plus additional components such as memory, and interfaces. These are combined as a single macrocell, that can be fabricated on an integrated circuit.

Register A temporary storage location used to hold binary data until it is ready to be used.

Reserved A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as zero and are read as zero.

Unaligned Memory accesses that are not appropriately word-aligned or halfword-aligned.

See also Aligned.

Unpredictable For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.