PrimeCell® AHB DDR and SRAM/NOR Memory Controller (PL245)

Revision: r0p1

Technical Reference Manual

ARM®

ARM DDI 0393B

PrimeCell AHB DDR and SRAM/NOR Memory Controller (PL245)
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
11 May 2006	A	Non-Confidential	First release for r0p0
20 December 2006	B	Non-Confidential	Updated for r0p1

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 AHB DDR and SRAM/NOR Memory Controller (PL245) Technical Reference Manual

List of Tables
PrimeCell AHB DDR and SRAM/NOR Memory Controller (PL245) Technical Reference Manual

List of Figures
PrimeCell AHB DDR and SRAM/NOR Memory Controller (PL245) Technical Reference Manual

Preface

This preface introduces the *PrimeCell AHB DDR and SRAM/NOR Memory Controller (MC) (PL245) Technical Reference Manual*. It contains the following sections:

About this manual

This is the *Technical Reference Manual* (TRM) for the *PrimeCell AHB DDR and SRAM/NOR Memory Controller*.

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 system designers, system integrators, and verification engineers who are designing a *System-on-Chip* (SoC) device that uses the AHB MC. The manual describes the external functionality of the AHB MC.

Using this manual

This manual is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for a high-level view of the AHB MC.

Chapter 2 *Functional Overview*

Read this chapter for a description of the major components of the AHB MC and how they operate.

Chapter 3 *Programmer's Model*

Read this chapter for a description of the AHB MC registers.

Chapter 4 *Programmer's Model for Test*

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

Chapter 5 *Device Driver Requirements*

Read this chapter for a description of device driver requirements for the *Dynamic Memory Controller* (DMC) and *Static Memory Controller* (SMC).

Appendix A *Signal Descriptions*

Read this appendix for a description of the AHB MC input and output 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 xvi
- *Signals* on page xvi
- *Numbering* on page xvii.

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 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.

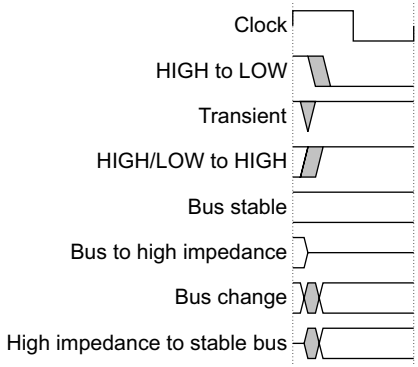
Note

Angle brackets can also enclose a permitted range of values. The example, <0-3>, shows that in name extensions, only one of the values 0, 1, 2, or 3 is valid.

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 Verilog 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 Limited, and by third parties.

ARM Limited 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 AHB MC. See the following documents for other relevant information:

- *PrimeCell AHB DDR and SRAM/NOR Memory Controller (PL245) Integration Manual* (ARM DII 0155)
- *PrimeCell AHB DDR and SRAM/NOR Memory Controller (PL245) Implementation Guide* (ARM DII 0150)
- *AMBA™ Specification* (Rev 2.0) ARM IHI 0011)
- *AMBA 3 APB Protocol v1.0 Specification* (ARM IHI 0024).

Feedback

ARM Limited welcomes feedback on the AHB MC and its documentation.

Feedback on this product

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 Limited also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the AHB MC. It contains the following sections:

1.1 About the AHB MC

The AHB MC is an *Advanced Microcontroller Bus Architecture* (AMBA) compliant *System-on-Chip* (SoC) peripheral. It is developed, tested, and licensed by ARM Limited.

The AHB MC takes advantage of the newly developed *Dynamic Memory Controller* (DMC) and *Static Memory Controller* (SMC). The AHB MC has six AHB ports with access to the external memory. Each AHB port has a bridge interface to the memory controllers. There is a separate AHB port to configure the memory controllers. Specific configurations of the SMC and DMC are instantiated to target specific memory devices. Figure 1-1 shows the AHB MC (PL245) configuration.

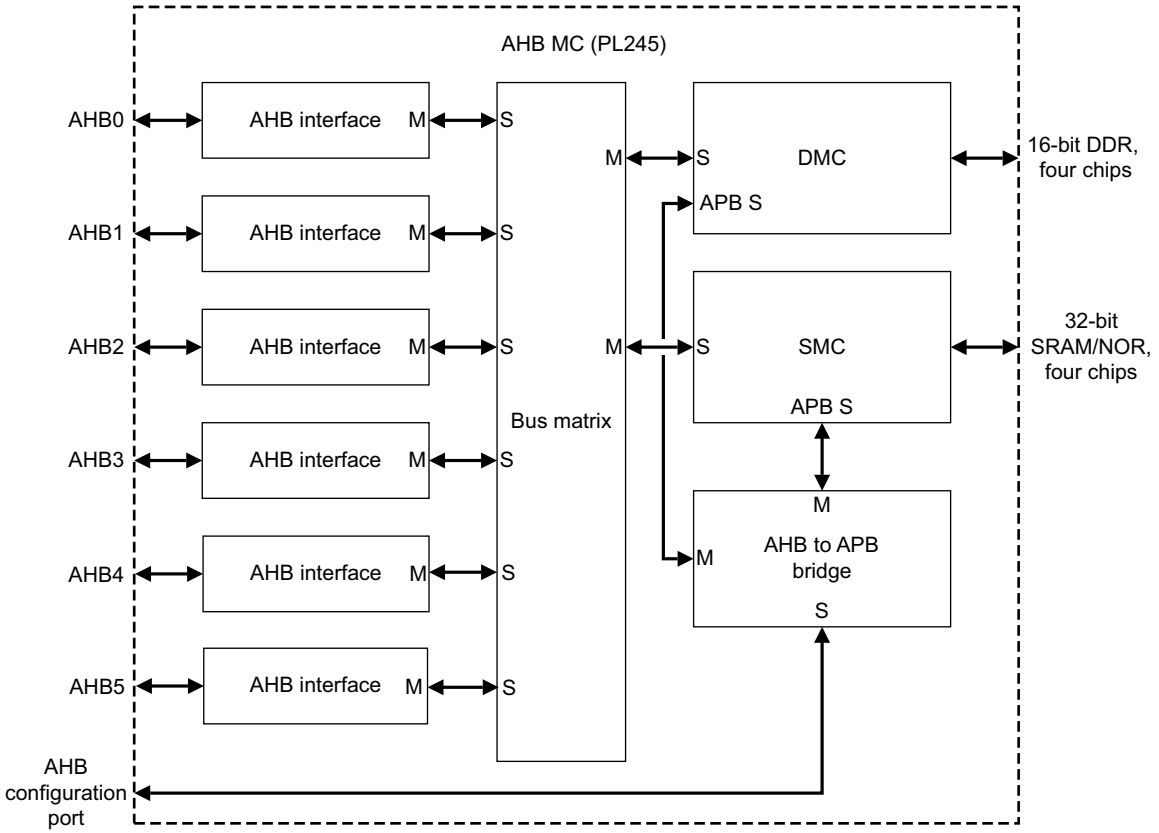


Figure 1-1 AHB MC (PL245) configuration

This section describes:

- *AHB interface*
- *AHB to APB bridge* on page 1-4
- *Bus matrix* on page 1-4
- *DMC* on page 1-4
- *SMC* on page 1-5
- *Clock domains* on page 1-5
- *Low-power interfaces* on page 1-5.

1.1.1 AHB interface

The interface converts the incoming AHB transfers to the protocol used internally by the AHB MC.

The interface has the following features:

- all AHB fixed length burst types are directly translated to fixed length bursts
- all undefined length INCR bursts are converted to INCR4 bursts
- broken bursts are supported
- the bufferable bit of the **HPROT** signal determines if the interface must wait for a write transfer to complete internally
- a *Read After Write* (RAW) hazard detection buffer avoids RAW hazards
- AHB response signals are registered to improve timing
- locked transfers are supported within a 512MB region
- **HWDATA** is registered to improve internal timing paths
- a big-endian 32-bit mode option is implemented
- AHB error response logic is removed because no internal components generate errors.

This interface is a fully validated component. This ensures that it obeys both the AHB protocol and the internal protocol that the interconnect uses.

See Chapter 2 *Functional Overview* for more information.

1.1.2 AHB to APB bridge

This bridge converts AHB transfers from the configuration port to the APB transfers that the internal memory controllers require.

See Chapter 2 *Functional Overview* for more information.

1.1.3 Bus matrix

The bus matrix enables each AHB port to access the two memory controllers.

See Chapter 2 *Functional Overview* for more information.

1.1.4 DMC

The DMC is optimized for 16-bit DDR SDRAM. The DMC supports four chip selects that you can individually program.

This DMC product is pre-configured and validated for:

- the SDRAM memory type
- the number of SDRAM memory devices
- the maximum SDRAM memory width.

The DMC block offers the following features:

- synchronous or asynchronous operation between AHB and the external memory bus
- active and precharge power down supported in the SDRAM
- quality of service features for low latency transfers
- support for the PL220 *External Bus Interface* (EBI) PrimeCell, enabling sharing of external address and data bus pins between memory controller interfaces
- optimized utilization of external memory bus
- tie-off pin to select external memory widths
- multiple outstanding addresses.

See Chapter 2 *Functional Overview* for more information.

1.1.5 SMC

The SMC is a high performance, area-optimized SRAM memory controller.

The SMC is pre-configured and validated for:

- the SRAM memory type
- the number of SRAM memory devices
- the maximum SRAM memory width.

The SRAM memory interface type is defined as supporting:

- synchronous or asynchronous SRAM
- *Pseudo Static Random Access Memory* (PSRAM)
- NOR flash
- NAND flash devices with an SRAM interface.

The SMC block offers the following features:

- it is configured to support the maximum SRAM memory data width of 32-bit
- programmable cycle timings and memory width per chip select
- atomic switching of memory device and controller operating modes
- support for the PL220 *External Bus Interface* (EBI) PrimeCell, enabling sharing of external address and data bus pins between memory controller interfaces
- support for a low-power interface
- support for a remap signal
- support for clock domains to be synchronous or asynchronous.

See Chapter 2 *Functional Overview* for more information.

1.1.6 Clock domains

The memory controller has three clock domains:

- AHB clock domain
- dynamic memory clock domain
- static memory clock domain.

See Chapter 2 *Functional Overview* for more information.

1.1.7 Low-power interfaces

The memory controller has three low-power interfaces, one for each clock domain.

See Chapter 2 *Functional Overview* for more information.

1.2 Supported devices

The DMC supports DDR SDRAM, see *DMC* on page 1-4. The *Release Note* provides a specific list of memory devices tested with each configuration.

The SMC supports SRAM/NOR, see *SMC* on page 1-5. The *Release Note* provides a specific list of memory devices tested with each configuration.

Some memory devices or series of memory devices have specific requirements:

Intel W18 series NOR FLASH, for example 28f128W18td

These devices, when in synchronous operation, use a **WAIT** pin. However, non-array operations when in synchronous mode do not use the **WAIT** pin and it is always asserted. The controller cannot differentiate between array and non-array accesses and therefore cannot support these non-array accesses.

Therefore, W18 devices can only carry out non-array operations such as `Read Status` in asynchronous modes of operation.

Cellular RAM 1.0, 64MB PSRAM, for example mt45w4mw16bfb_701_1us

You can program these devices using a **CRE** pin or by software access. Whenever you program these devices through software access, using a sequence of two reads followed by two writes, ensure that the third access, that is, the first write is a `CE#` controlled write.

SMC only does `WE#` controlled writes. This is to simplify the design of the SMC by having fewer timing registers and simpler timing controls.

Therefore, you can program these devices using the **CRE** pin method of access.

Note

Because the memory controller maps INCR transfers into INCR4 transfers, it does not support memory mapped FIFO components.

Chapter 2
Functional Overview

This chapter describes the major components of the AHB MC and how they operate. It contains the following sections:

2.1 Functional description

Figure 2-1 shows the AHB MC (PL245) configuration.

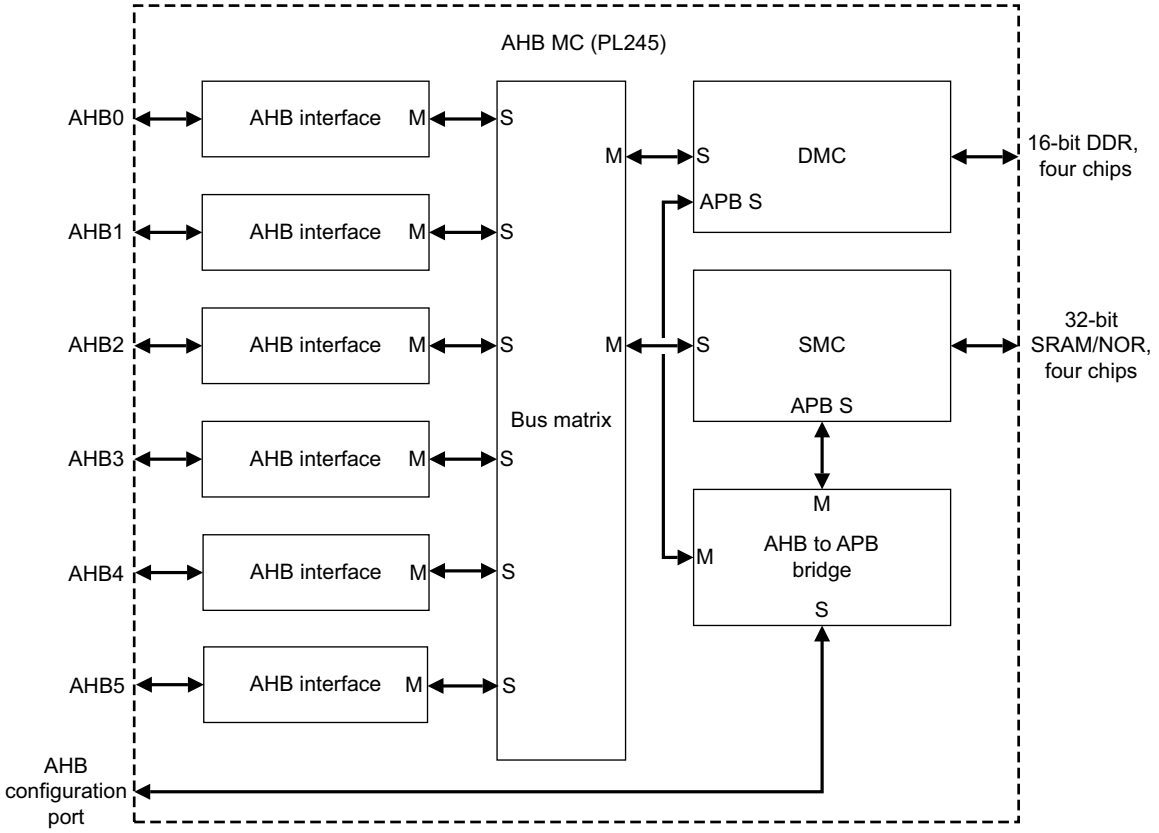


Figure 2-1 AHB MC (PL245) configuration

This section is divided into:

- *AHB interface* on page 2-3
- *AHB to APB bridge* on page 2-3
- *Bus matrix* on page 2-3
- *Clock domains* on page 2-3
- *Low-power interface* on page 2-4
- *DMC* on page 2-5
- *SMC* on page 2-10.

2.1.1 AHB interface

The AHB MC fully supports the AMBA AHB 2.0 specification. This interface component converts the incoming AHB transfers to the required transfers of the internal interconnect protocol. Because of the design of the internal interconnect, some optimizations are made in the interface to improve performance.

See *AHB interface operation* on page 2-13 for more information.

2.1.2 AHB to APB bridge

The internal memory controllers of the AHB MC use the AMBA3 APB protocol for their configuration ports. To enable the AHB MC to externally function as an AHB device, the APB configuration ports are connected to an AHB to APB bridge. The bridge converts incoming AHB transfers from the configuration port to the APB transfers that the internal memory controllers require. This bridge is part of the PrimeCell infrastructure components, part BP127.

See *AHB to APB bridge operation* on page 2-16 for more information.

2.1.3 Bus matrix

The bus matrix enables the AHB ports to access either of the internal memory controllers. The matrix uses round robin arbitration enabling each AHB port to have equal priority.

See *Bus matrix operation* on page 2-17 for more information.

2.1.4 Clock domains

The memory controller has three clock domains:

AHB clock domain

This is clocked by **hclk**, **dmc_alk**, and **smc_aclk** and is reset by **hresetn**.

Dynamic memory clock domain

This is clocked by **dmc_mclk**, **dmc_mclkn**, **dmc_mclkx2**, **dmc_mclkx2n** and reset by **dmc_mresetn**.

Static memory clock domain

This is clocked by **smc_mclk0**, and **smc_mclk0n** and is reset by **smc_mreset0n**.

Figure 2-2 on page 2-4 shows the three clock domains.

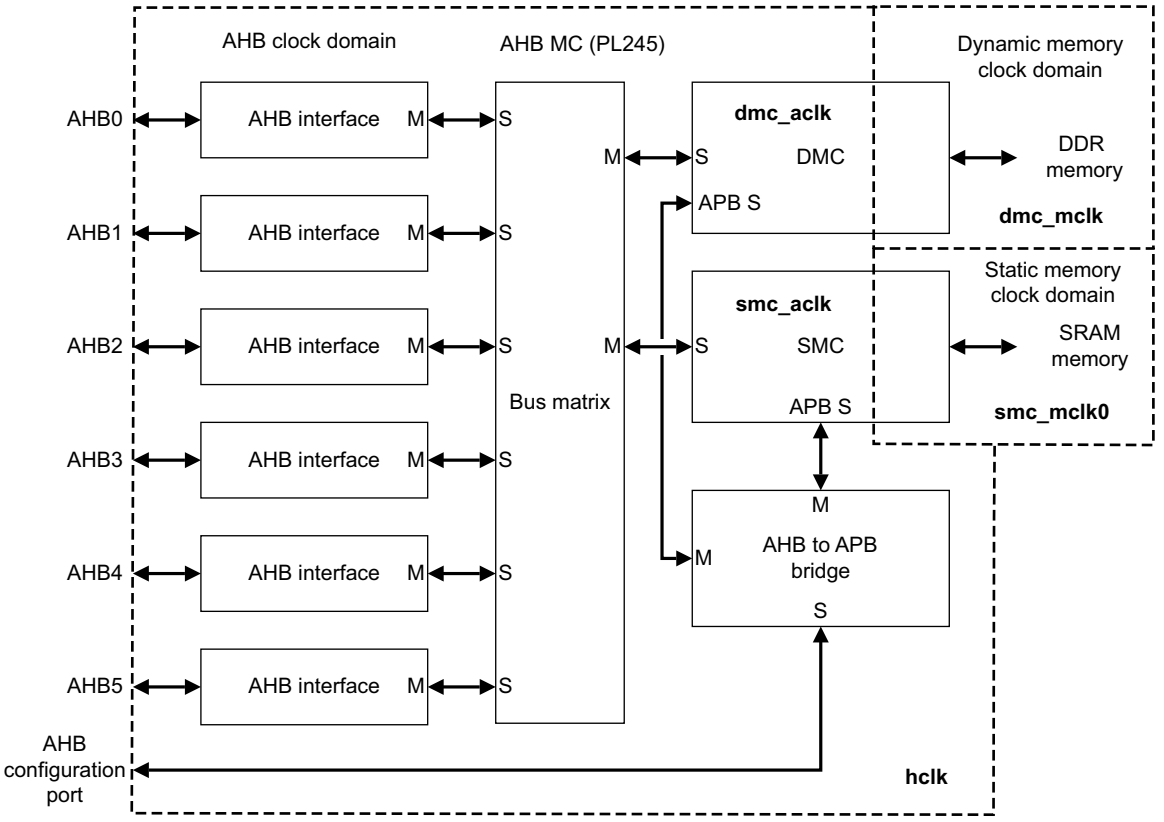


Figure 2-2 AHB MC (PL245) clock domains

The memory controller supports many different options for clocking the different domains. The DMC and SMC clock domains are completely independent and each have clocking options.

See *Clock domain operation* on page 2-19 for more information.

2.1.5 Low-power interface

The memory controller has three low-power interfaces, one for each clock domain. These operate with a simple three signal protocol. It is expected that a system clock controller drives these interfaces and associated clocks. Each domain has individual control that enables one memory clock domain or all clock domains to be powered down.

See *Low-power interface operation* on page 2-21 for more information.

2.2 DMC

Figure 2-3 shows a block diagram of the DMC.

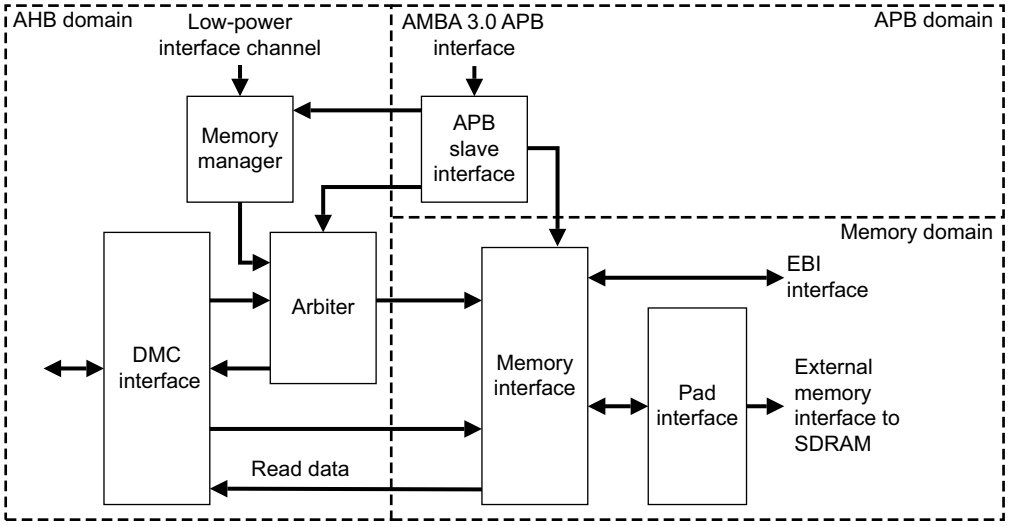


Figure 2-3 DMC block diagram

The DMC interface processes the incoming transfers from the AHB ports. It can only add a read or write to the queue at any one time. It uses round robin arbitration to decide between a simultaneous read or write.

The DMC uses a scheduling algorithm to determine the item in the queue to process next to maximize data bandwidth. The algorithm uses information including:

- the memory rows that are currently open
- the order of transfers received from each AHB port
- timing of refresh commands from the manager
- the *Quality Of Service* (QoS) required by each port.

The QoS is set by a register within the DMC on a port by port basis. The QoS value indicates a required read maximum latency. A QoS timeout causes the transaction to be raised to a higher priority. You can also set the QoS to minimum for a specific port so that its transfers are serviced with a higher priority. This impacts the overall memory bandwidth because it limits the options of the scheduling algorithm.

The main blocks of the DMC are:

- *Arbiter* on page 2-6
- *Memory manager* on page 2-6
- *APB slave interface* on page 2-7

- *Memory interface* on page 2-7
- *Pad interface* on page 2-8.

2.2.1 Arbiter

The arbiter receives memory access commands from the DMC 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 DMC interface.

See *Arbiter operation* on page 2-27 for more information.

2.2.2 Memory manager

The memory manager tracks and controls the current state of the DMC using a *Finite State Machine* (FSM) as Figure 2-4 shows.

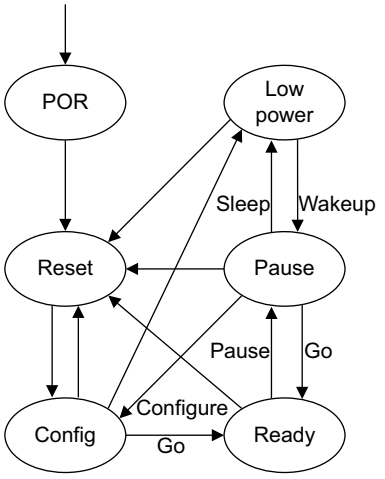


Figure 2-4 dmc_aclk domain FSM diagram

In Figure 2-4, non-state moving transitions are omitted for clarity. See Table 2-4 on page 2-42 for valid system states.

APB commands to the Direct Command Register or the low-power interface control the state of the DMC. Figure 2-5 shows the low-power interface channel signals.

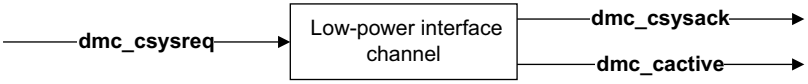


Figure 2-5 Low-power interface channel signals

The 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.

See *Memory manager operation* on page 2-31 for more information.

2.2.3 APB slave interface

The APB interface implements the register file as Chapter 3 *Programmer's Model* describes.

Note

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

See *APB slave interface operation* on page 2-32 for more information.

2.2.4 Memory interface

The memory interface provides a clean and defined interface between the pad interface and the arbiter ensuring 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 only external I/Os to this block are **dmc_mclk** and **dmc_mresetn**.

The memory interface tracks and controls the state of the external memories using an FSM. See Figure 2-6 on page 2-8.

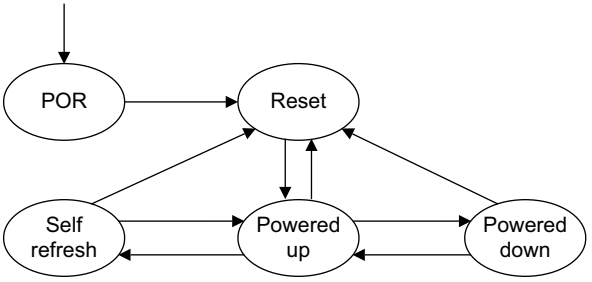


Figure 2-6 dmc_mclk domain FSM diagram

See Table 2-4 on page 2-42 for valid system states.

See *Memory interface operation* on page 2-33 for more information.

2.2.5 Pad interface

The pad interface provides a flip-flop for each external signal.

Figure 2-7 shows the pad interface external connections.

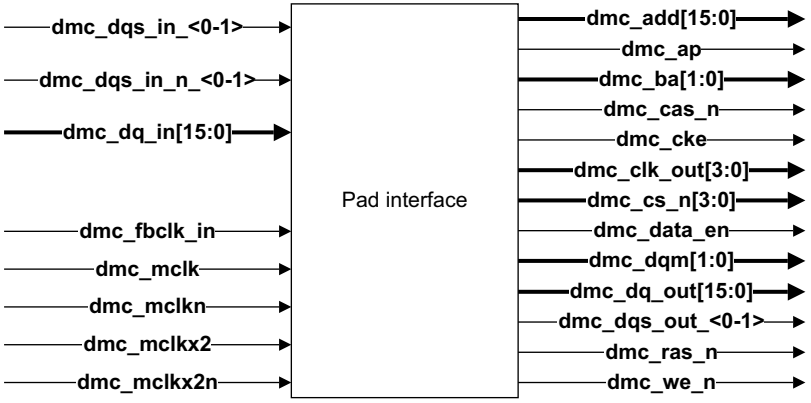


Figure 2-7 DMC Pad interface external connections

Pad interface to external memory devices

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

To support a PrimeCell EBI (PL220) the **dmc_ap** precharge bit signal is also an external signal to the DMC. Having **dmc_ap** separate to the address bus means that PRECHARGEALL commands to dynamic memory can be issued when the EBI has granted the external memory interface to another memory controller.

It is expected that a *Delay-Locked Loop* (DLL) is required to delay the **dmc_dqs** signals coming back from the memories with respect to the **dmc_dq** data bus. The standard delay for the **dmc_dqs** signals is a quarter clock period of **dmc_mclk**. The DLL is not included in the DMC.

See *Pad interface operation* on page 2-38 for more information.

2.3 SMC

Figure 2-8 shows a block diagram of the SMC.

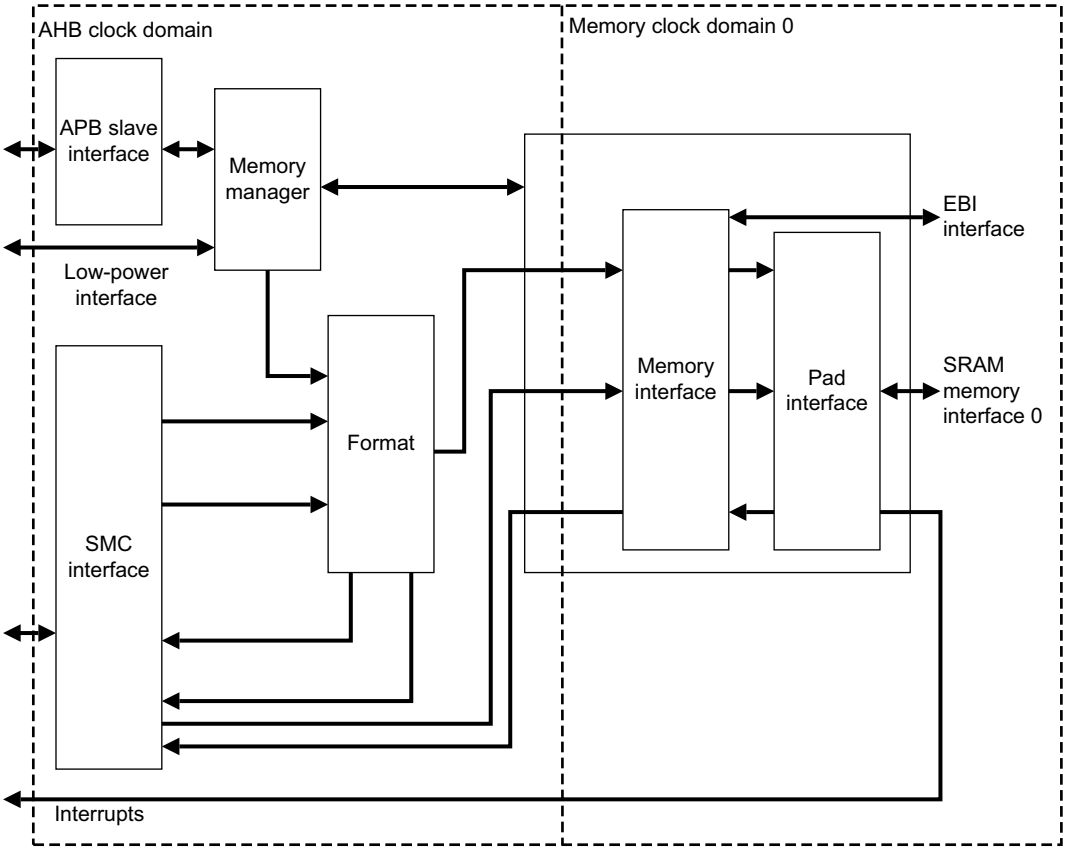


Figure 2-8 SMC block diagram

The main blocks of the SMC are:

- *SMC interface* on page 2-11
- *APB slave interface* on page 2-11
- *Format* on page 2-11
- *Memory manager* on page 2-11
- *Memory interface* on page 2-11
- *Pad interface* on page 2-12
- *Interrupts* on page 2-12.

2.3.1 SMC interface

The SMC interface processes the incoming AHB transfers and sends them to the command format block.

2.3.2 APB slave interface

The SMC has 4KB of memory allocated to it.

The APB slave interface accesses the SMC registers to program the memory system configuration parameters and to provide status information. See Chapter 3 *Programmer's Model* and *APB slave interface operation* on page 2-52 for more information.

2.3.3 Format

The format block receives memory accesses from the SMC interface and the memory manager. Read and write requests are arbitrated on a round robin basis. Requests from the manager have the highest priority. The format block also maps AHB memory transfers onto appropriate memory transfers and passes these to the memory interface through the command FIFO.

See *Format block* on page 2-52 for more information.

2.3.4 Memory manager

The memory manager tracks and controls the current state of **smc_aclk** domain logic. The block is responsible for:

- updating timing registers and controlling direct commands issued to memory
- controlling entry-to and exit-from low-power mode through the APB interface
- the low-power interface.

See *Memory manager operation* on page 2-54 for more information.

2.3.5 Memory interface

The SRAM memory interface consists of command, read data and write data FIFOs, plus a control FSM. To support an EBI, the memory interface also contains an EBI FSM. This controls interaction with the EBI and prevents the memory interface FSM from issuing commands until it has been granted the external bus.

See *Memory interface operation* on page 2-60 for more information.

2.3.6 Pad interface

The pad interface module provides a registered I/O interface for data and control signals. It also contains interrupt generation logic.

Figure 2-9 shows the SMC SRAM pad interface external connections. Clock and reset signals are omitted.

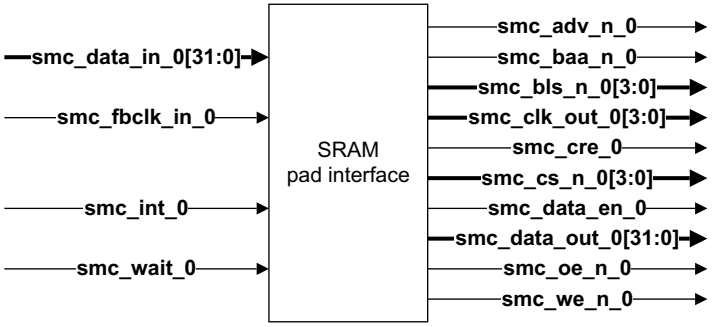


Figure 2-9 SMC SRAM pad interface external connections

2.3.7 Interrupts

The SRAM memory interface support interrupts. The interrupt is triggered on the rising edge of the **smc_int_0** input for the SRAM memory interface.

See *Interrupts operation* on page 2-60 for more information.

2.4 Functional operation

This section is divided into:

- *AHB interface operation*
- *AHB to APB bridge operation* on page 2-16
- *Bus matrix operation* on page 2-17
- *Clock domain operation* on page 2-19
- *Low-power interface operation* on page 2-21
- *DMC functional operation* on page 2-24
- *SMC functional operation* on page 2-48.

2.4.1 AHB interface operation

This section describes:

- *AHB fixed burst types*
- *Undefined length INCR bursts* on page 2-14
- *Broken bursts* on page 2-14
- *Bufferable bit of the HPROT signal* on page 2-14
- *Read after write hazard detection buffer* on page 2-15
- *AHB response signals* on page 2-15
- *Locked transfers* on page 2-15
- *Registered HWDATA* on page 2-16
- *Big-endian 32-bit mode* on page 2-16
- *Removal of AHB error response logic* on page 2-16.

AHB fixed burst types

All AHB fixed length bursts directly map to burst types that the internal interconnect uses. The internal interconnect and the memory controllers are based on transferring bursts of data. The larger the burst size, the more efficient the transfer and overall performance. The standard AHB fixed length burst types are directly mapped to the internal protocol.

Burst operation has performance benefits because when the first beat of a burst is accepted, it contains data about the remaining beats. For example, from the first beat of a read burst, all the data required to complete the transfer can be read from memory. This first transfer has some delay before data is returned. Subsequent beats of the burst can have less delay because the data they require might have already been read from the memory.

Undefined length INCR bursts

All undefined length INCR bursts are converted to INCR bursts of length four. Many AHB masters rely on using undefined length INCR bursts to access data. If each INCR transfer is processed as a single transfer by the internal protocol then the performance is significantly degraded.

The bridge converts the incoming INCR transfers to INCR transfers of length four, INCR4. This mean that the bridge speculatively requests data from the internal interconnect, before it knows it is going to require it. If the AHB master continues the burst, then the data can be returned quickly because it has already been requested. When the INCR burst finishes, the bridge disregards any data requested from the internal interconnect that is not required.

Any INCR burst of less than four beats results in a broken INCR4. Undefined length INCR bursts of more than four beats are split into an appropriate number of INCR4s plus a broken INCR4, if required.

Broken bursts

To fully support the AMBA AHB 2.0 specification, the bridge supports all broken AHB bursts. Although bursts cannot be broken by an AHB master, if the AHB system has multiple masters then the AHB system arbitration can break a burst. Also, because the bridge converts INCR to INCR4, broken INCR4s occur when undefined length INCRs of a length not equal to a multiple of four are performed.

To support broken bursts, the bridge must keep track of how many beats of a burst have been performed and ensure it obeys the protocol of the interconnect. For read bursts, this means draining the interconnect of any requested data that is not required. For write bursts this means artificially extending write data with enough beats to obey the protocol. The interconnect uses write strobes to indicate the bytes of the data bus that are valid. When extending broken bursts, these strobes are deasserted so that the artificial data does not corrupt the actual memory.

Bufferable bit of the HPROT signal

The bufferable bit of the **HPROT** signal determines whether the bridge must wait for a write transfer to complete internally. The AHB protection control bits support the concept of bufferable data accesses. The **HPROT[2]** signal determines this. The internal interconnect supports the concept of a write response to indicate when data has actually been written to memory. The bridge exploits these features by not waiting for the write response if the access is described as bufferable. This enables numerous bufferable writes to occur with minimum latency. These are accepted by the interconnect and queued in the memory controller.

If transfers are described as non-bufferable then the bridge must wait for the write response to indicate that the transfer has been completed to memory. If numerous bufferable writes are performed, followed by a non-bufferable write, then the bridge must wait until it receives the write response associated with the final write.

Read after write hazard detection buffer

A RAW hazard detection buffer avoids potential RAW hazards. The protocol used internally to AHB MC does not perform memory coherency checks to catch *Write After Read* (WAR) or RAW hazards.

Because of the nature of the AHB protocol, WAR hazards never occur because the read must have completed before the write can be accepted.

Because the bridge permits writes to be buffered internally, there is a potential for a RAW hazard to occur. If you perform a bufferable write then it might not complete immediately. If a read to that same memory location is performed then both transfers can be in the queue and the internal memory controllers can reorder these transactions for performance reasons so that the read occurs before the write. This means that the data read can be the value before the most recent write. The bridge has to detect these potential cases and stall the read transfer until any buffered writes that might cause a RAW hazard have been completed.

The bridge contains logic to monitor up to four outstanding write addresses. If an incoming read occurs to a 4KB region that has been written to, then it is stalled. If four bufferable writes occur then the AHB is stalled until a response is seen for the first of the four writes in the buffer.

AHB response signals

The interconnect used within the AHB MC contains many combinatorial paths that link different AHB input ports. To improve the synthesis timing, the AHB responses are registered to limit these paths to within the design.

Locked transfers

AHB MC supports locked transfers, within a 512MB region. This is because of the way the interconnect processes locked transfers. There is a significant performance penalty in using locked transfers. Transfers that are locked together wait for all other ports to complete any outstanding transfers to that region before they can begin. While a locked sequence occurs to a specific 512MB memory region, all other access to that region is stalled. All locked writes are processed as non-bufferable writes and so have to wait for the appropriate write response before indicating their completion.

You can lock the DMC and SMC channels independently. Therefore you can have:

- DMC locked accesses, SMC unlocked accesses
- DMC unlocked accesses, SMC locked accesses
- DMC and SMC locked accesses.

Registered HWDATA

The interconnect used within the AHB MC contains combinatorial paths for the write data. To improve the synthesis timing, **HWDATA** is registered and makes these paths internal to the design.

Big-endian 32-bit mode

The AHB MC supports the option of storing data to memory in big-endian 32-bit mode. Each bridge contains the logic to implement this data mapping depending on the **big_endian** input tie-off. Figure 2-10 shows that if the tie off is asserted then the data buses are reordered.

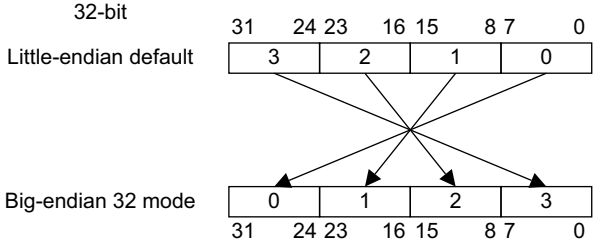


Figure 2-10 Big-endian implementation

Removal of AHB error response logic

The internal protocol used within AHB MC supports the concept of errors. However none of the components used ever generate errors. This means that the bridge does not require any logic to generate AHB errors because there are no circumstances under which errors can be generated.

2.4.2 AHB to APB bridge operation

The internal memory controllers each have an APB configuration port. The AHB configuration port is mapped to these controllers using an AHB to APB bridge. Figure 2-11 on page 2-17 shows that each internal memory controller configuration port has a 4KB address space.

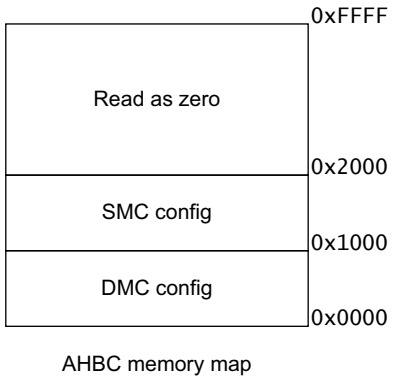


Figure 2-11 AHBC memory map

The other fourteen 4KB regions are read as zero. The lower 16 bits of the AHB address decode the memory controller that is being used. An external AHB decoder determines where in the system memory map, this 64KB region is located. See *About the programmer's model* on page 3-2 for information on the internal memory controller configuration registers. The configuration ports of the internal memory controllers are APB, so only word reads and writes are supported.

2.4.3 Bus matrix operation

The following sections explain the bus matrix configuration features and options:

- *Arbitration scheme*
- *Locked transfers*
- *Memory map* on page 2-18.

Arbitration scheme

The bus matrix supports a round robin arbitration scheme. This enables each port to have an equal opportunity to access memory.

Locked transfers

A locked sequential transfer must be within the same 512MB address region to guarantee that it is to a single memory controller. To maintain data integrity throughout a locked sequence, the interconnect only accepts the locked transfer after the target slave has completed all outstanding transactions. The slave then becomes locked and only the locking master can access it. The slave remains locked until the locking master completes an unlocked transfer to that slave. This unlocking transfer is added by the AHB interface at the end of a locked sequence.

Memory map

The bus matrix is configured with predefined 512 MB regions of memory allocated to the internal memory controllers. Figure 2-12 shows the memory map.

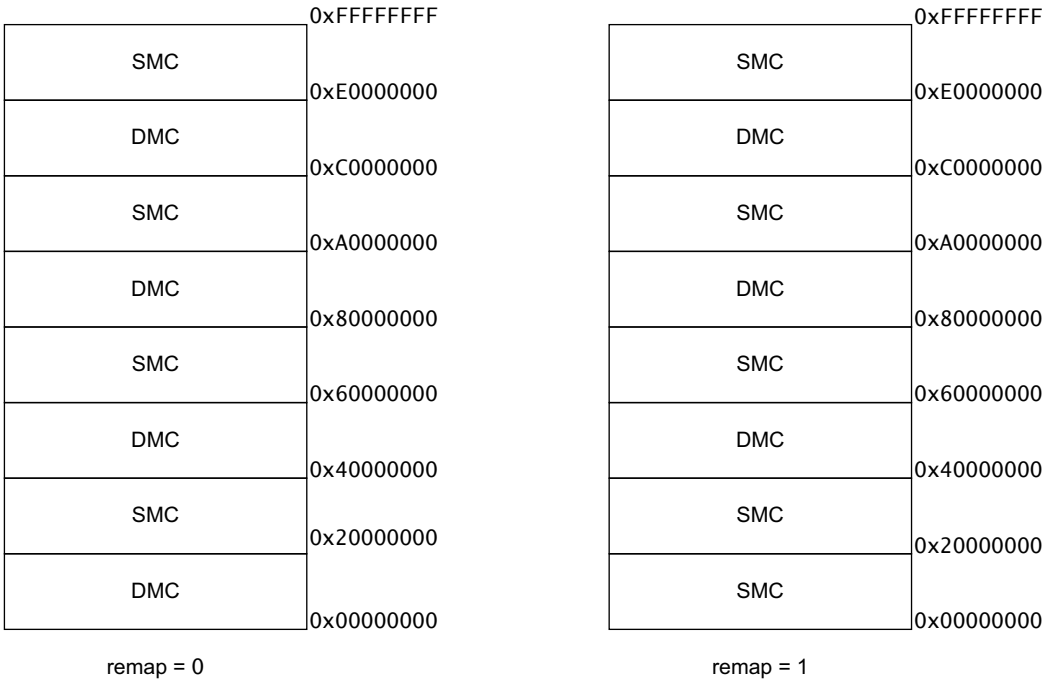


Figure 2-12 Memory map

This enables the system designer maximum flexibility with memory regions for a fixed bus matrix memory map. You must program the address match and mask fields of the static and dynamic memory controllers to determine the address decode for each chip select. See *DMC chip_<0-3>_cfg Registers at 0x0200* on page 3-22 and *SMC Opmode Registers <0-3> at 0x1104, 0x1124, 0x1144, 0x1164* on page 3-42. The selected values must be within the constraints of the bus matrix memory map shown in Figure 2-12. The **remap** pin enables either the static or the dynamic memory controller to be decoded to address `0x00000000`.

The SMC also contains some independent remap functionality. You can use the **smc_remap_0** signal to alias SMC chip select 0 to address `0x00000000` in addition to its address match and mask tie-off. If the **smc_remap_0** signal is HIGH, then the memory controller uses the **smc_sram_mw_0[1:0]** tie-off to determine the width of the external memory chip. The maximum timing values enable the system to boot from chip select 0 of the SMC.

2.4.4 Clock domain operation

The memory controller supports three clock domains.

- the AHB clock domain
- the dynamic memory clock domain
- the static memory clock domain.

The **hclk** input drives the AHB clock domain. This clock drives the AHB interfaces and bus matrix. Each of the internal memory controllers has a separate clock input in this domain. These are called **dmc_aclk** and **smc_aclk**. These signals are separated to enable the clock to be stopped on an individual memory controller for low-power operation, see *Low-power interface operation* on page 2-21. These three clocks must always be driven from the same clock source. The input signal **hresetn** resets the clock domain.

The dynamic memory clock domain controls the memory interface logic of the DMC. The input signal **dmc_mclk** and its inverse **dmc_mclkn** drive this domain. Each external dynamic memory chip is driven by a gated **dmc_mclk** signal, these are called **dmc_clk_out[3:0]**. A double speed memory clock is also required because the SDRAM type is DDR. This is driven by the input signal **dmc_mclkx2** and its inverse **dmc_mclkx2n**. Clocks are only driven out to chips that require them. The dynamic memory interface has a fed-back clock input, **dmc_fbclk_in**, to help with clock skews on the external pads.

The static memory clock domain controls the memory interface logic of the SMC. The input signal **smc_mclk0** and its inverse **smc_mclk0n** drive this domain. Each external static memory chip is driven by a gated **smc_mclk0** signal, these are called **smc_clk_out_0[3:0]**. Clocks are only driven out to chips that require them. The static memory interface has a fed back clock input, **smc_fbclk_in_0**, to help with clock skews on the external pads.

The memory controller supports many different options for clocking the different domains. The dynamic and static clock domains are completely independent and have clocking options:

- *Dynamic memory clocking options* on page 2-20
- *Static memory clocking options* on page 2-20.

Dynamic memory clocking options

Table 2-1 lists the dynamic memory clocking options.

Table 2-1 Dynamic memory clocking options

Options	Tie-off values
Fully synchronous	
hclk = dmc_mclk	**dmc_async** = **dmc_msync** = 1 **dmc_a_gt_m_sync** = 0
Synchronous multiples	
hclk = n x **dmc_mclk** where: n = integer value	**dmc_async** = **dmc_msync** = 1 **dmc_a_gt_m_sync** = 0
m x **hclk** = **dmc_mclk** where: m = integer value	**dmc_async** = **dmc_msync** = 1 **dmc_a_gt_m_sync** = 1
Asynchronous	
Extra registers are used to avoid metastability when crossing the asynchronous clock boundary.	**dmc_async = dmc_msync** = 0 **dmc_a_gt_m_sync** = 0

Static memory clocking options

Table 2-2 lists the static memory clocking options.

Table 2-2 Static memory clocking options

Options	Tie-off values
Fully synchronous	
hclk = smc_mclk0	**smc_async0** = **smc_msync0** = 1 **smc_a_gt_m0_sync** = 0
Synchronous multiples	
hclk = n x **smc_mclk0** where: n = integer value	**smc_async0** = **smc_msync0** = 1 **smc_a_gt_m0_sync** = 0

Table 2-2 Static memory clocking options (continued)

Options	Tie-off values
m x **hclk** = **smc_mclk0** where: m = integer value	**smc_async0** = **smc_msync0** = 1 **smc_a_gt_m0_sync** = 1
Asynchronous	
Extra registers are used to avoid metastability when crossing the asynchronous clock boundary.	**smc_async0 = smc_msync0** = 0 **smc_a_gt_m0_sync** = 0

2.4.5 Low-power interface operation

The memory controller has three low-power interfaces. These interfaces indicate whether the clock for a specific domain can be switched off to reduce power consumption. It is expected that these interfaces are controlled by a system clock controller. One interface controls each of the following domains:

- AHB clock domain
- dynamic memory clock domain
- static memory clock domain.

Each domain uses a simple three signal interface to indicate whether the clocks are required. The signals consist of:

a request input

<domain>_csyreq

an acknowledge output

<domain>_csysack

an active output

<domain>_cactive

Where:

<domain> is ahb, dmc or smc.

The following diagrams explain the protocol for the interface:

Figure 2-13 on page 2-22 shows a request to enter low-power mode.

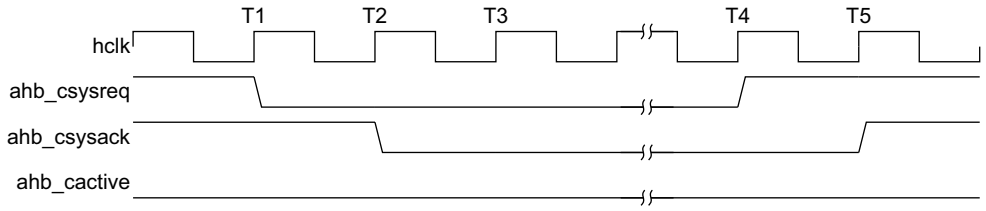


Figure 2-13 Request to enter low-power mode

The memory controller receives a request to enter low-power mode, indicated by **<domain>_csysreq** being driven LOW by the system clock controller, as shown at T1. The memory controller then has the chance to perform any required operations to prepare for the clock to be switched off. The memory controller acknowledges the request by asserting **<domain>_csysack** LOW, as shown at T2. At this point the **<domain>_cactive** signal is used to indicate whether the request has been accepted or denied. If the request is accepted, **<domain>_cactive** is LOW, as Figure 2-13 shows. If the request is denied, **<domain>_cactive** is HIGH. If the request is accepted, then the clock to that domain can be switched off. The peripheral is brought out of Low-power state by restarting the clock and driving **<domain>_csysreq** HIGH, as shown at T4. The memory controller completes the handshake by driving **<domain>_csysack** HIGH, as shown at T5. Figure 2-14 shows the AHB domain denying a low-power request.

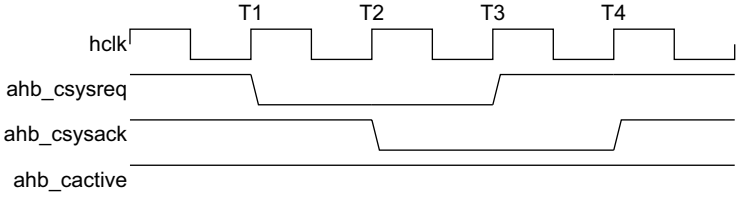


Figure 2-14 AHB domain denying a low-power request

When **ahb_csysack** is asserted LOW, the **ahb_cactive** signal is HIGH, as shown at T3, indicating the AHB domain is busy and the clock cannot be switched off. The handshake must be completed.

The AHB domain accepts or denies requests based on whether it is busy performing any transfers. Figure 2-15 on page 2-23 shows that dynamic and static memory controllers always accept requests after they have performed the required operations to prepare the external memory for the clock to be switched off.

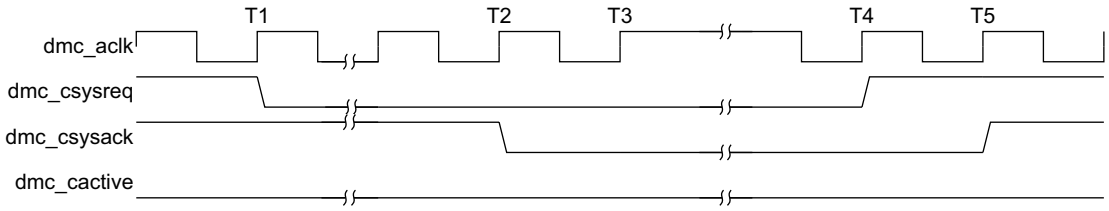


Figure 2-15 Accepting requests

The low-power request **dmc_csysreq** is driven LOW at time T1. When the memory controller is happy for the clock to be switched off, the **dmc_csysack** signal is driven LOW to acknowledge the request, as shown at T2. **dmc_cactive** is driven LOW, so the system clock controller knows the request has been accepted. When acknowledged, the system clock controller can disable both the **dmc_aclk** and **dmc_mclk** signals because the external memory is now in self refresh mode.

The three domains have separate interfaces to enable one or more domains to be switched off. The simple usage model is to switch off all domains. In this usage each individual low-power interface protocol must be observed before all the clocks can be disabled.

Another option is to put either the static or dynamic memory controller into low-power mode individually. This saves power on the memory controller not being used, while enabling the other memory controller to still perform data accesses. For example, if the system calculated that it did not require any data from dynamic memory for some considerable time it can use the DMC low-power interface to put the external memory into self refresh mode and then turn off the **dmc_aclk** and **dmc_mclk** signals. The memory controller can still access the static memory. In this situation, any access to the dynamic memory can cause the entire memory controller to stall as it waits for a response from the DMC that has no clock. You must take care to ensure that this scenario never arises.

2.5 DMC functional operation

This section describes:

- *Clocking and resets*
- *Miscellaneous signals* on page 2-26
- *DMC slave interface* on page 2-27
- *Arbiter operation* on page 2-27
- *Memory manager operation* on page 2-31
- *APB slave interface operation* on page 2-32
- *Memory interface operation* on page 2-33
- *Pad interface operation* on page 2-38
- *Initialization* on page 2-39
- *Power-down support and usage model* on page 2-42.

2.5.1 Clocking and resets

This section describes:

- *Clocking*
- *Resets* on page 2-26.

Clocking

The DMC has the following functional clock inputs:

- **dmc_aclk**
- **dmc_mclk**
- **dmc_mclkn**
- **dmc_mclkx2**
- **dmc_mclkx2n**
- **dmc_fbclk_in**
- **dmc_dqs_in_<0-1>**
- **dmc_dqs_in_n_<0-1>**.

These clocks can be grouped into two clock domains:

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

dmc_mclk domain All clocks except **dmc_aclk** are in this domain. The **dmc_mclk** signal must be clocked at the rate of the external memory clock. The **dmc_mclk** domain signals can only be stopped if the external memories are put in Self-refresh mode. The **dmc_mclkx2** is the double speed clock required for DDR.

You can tie off the DMC **dmc_async** and **dmc_msync** pins so that the two clock domains can operate synchronously or asynchronously with respect to each other:

Synchronous clocking

The benefit of synchronous clocking is that you can reduce the read and write latency by removing the synchronization registers between clock domains. However, because of the integer relationship of the clocks, you might not be able to get the maximum performance from the system because of constraints placed on the bus frequency by the external memory clock speed. In synchronous mode, the handshaking between the **dmc_aclk** and **dmc_mclk** domains enables synchronous operation of the two clocks at multiples of each other, that is ratios of n:1 and 1:m.

Asynchronous clocking

The main benefit of asynchronous clocking is that you can maximize the system performance, while running the memory interface at a fixed system frequency. Additionally, in sleep-mode situations when the system is not required to do much work, you can lower the frequency to reduce power consumption.

Output clocks

A clock output is provided for every external memory device. These are called **dmc_clk_out[3:0]**. These outputs are gated versions of the input **dmc_mclk**.

Data clocks

The data is clocked into and out of the external memory device using the **dmc_dqs** strobes. There is a strobe per external memory data byte. The strobes drive a tristate bus for write the memory controller drives out the strobes using **dmc_dqs_out_<0-1>**. For reads, the memory drives the strobes **dmc_dqs_in_<0-1>** and their inverse **dmc_dqs_in_n_<0-1>**.

Resets

The DMC has two reset inputs:

hresetn This is the reset signal for the **dmc_aclk** domain.

dmc_mresetn

This is the reset signal for the **dmc_mclk** domain.

Both reset signals can be changed asynchronously to their respective clock domain. Internally to the DMC, the deassertion of the **hresetn** signal is synchronized to **dmc_aclk**, and the deassertion of the **dmc_mresetn** signal is synchronized to the **dmc_mclk**, **dmc_mclkn**, **dmc_mckx2** and **dmc_mclkx2n** clock signals.

The clock inputs, **dmc_fbclk**, **dmc_dqs_in_<0-1>**, and **dmc_dqs_in_n_<0-1>** do not clock any resettable logic.

2.5.2 Miscellaneous signals

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

dmc_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 **dmc_aclk** signal, then external synchronization registers are required.

dmc_user_config[7:0]

General-purpose ports that are driven directly from a write-only APB register. If you do not require these ports leave them unconnected.

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

dmc_cke_init

The **dmc_cke** output port to the external memory resets to this value.

dmc_dqm_init

The **dmc_dqm** output ports to the external memory reset to this value.

dmc_memory_width[1:0]

This configures the external memory width. See *dmc_memc_status Register bit assignments* on page 3-8 for the definition of these two pins.

dmc_msync When HIGH, indicates **dmc_mclk** is synchronous to **dmc_aclk**. Otherwise they are asynchronous. Ensure that **dmc_msync** is tied to the same value as **dmc_async**.

dmc_async When HIGH, indicates **dmc_aclk** is synchronous to **dmc_mclk**. Otherwise they are asynchronous. Ensure that **dmc_async** is tied to the same value as **dmc_msync**.

dmc_a_gt_m_sync

When HIGH, indicates **dmc_aclk** is synchronous to **dmc_mclk** and **dmc_aclk** is greater than **dmc_mclk**.

dmc_use_ebi

When HIGH, indicates that the DMC must operate with a PrimeCell EBI. See *ARM PrimeCell External Bus Interface (PL220) Technical Reference Manual*.

dmc_rst_bypass

Use this signal for ATPG testing only. It must be tied LOW for normal operation.

dmc_dft_en_clk_out

Use this signal for ATPG testing only. It must be tied LOW for normal operation.

2.5.3 DMC slave interface

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

The base addresses of the external memory devices are programmable using the dmc_chip_cfg Registers, see *DMC chip_<0-3>_cfg Registers at 0x0200* on page 3-22.

2.5.4 Arbiter operation

This section describes:

- *Formatting from the DMC interface* on page 2-28
- *Formatting from the memory manager* on page 2-29
- *Arbiter access mux* on page 2-29
- *QoS* on page 2-29
- *Hazard detection* on page 2-30
- *Scheduler* on page 2-30

- *Arbitration algorithm* on page 2-30
- *Command formatting* on page 2-31.

Formatting from the DMC interface

Formatting is as follows:

Chip select decoding

Using the programmed values in the dmc_chip_<0-3>_cfg Registers defined in Chapter 3 *Programmer's Model*, an incoming address has the most significant eight address bits when compared with the address match bits. The later method uses 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 address using bits [5:3] of the dmc_memory_cfg Register, and also the brc_n_rbc bit for the selected chip defined in the dmc_chip_<0-3>_cfg Register.

Column select decoding

The column address is determined from the address using bits [2:0] of the dmc_memory_cfg Register.

Bank select decoding

The chip bank is determined using the brc_n_rbc bit for the selected chip defined in the dmc_chip_<0-3>_cfg Register.

Number of beats

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

Quality of Service (QoS) selection

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

There are two forms of quality of service tracking:

- `qos_max` time-out
- `qos_min` time-out.

Example 2-1 shows that the type of QoS and QoS value is determined by the dmc_id_<0-5>_cfg Register.

Example 2-1

If a transfer is received from port 1, the dmc_id_1_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 dmc_id_1_cfg Register. In addition to this, if the qos_override bit associated with this port, **dmc_qos_override[1]**, is HIGH when the transfer is accepted then the `qos_min` arbiter entry is forced HIGH irrespective of whether the qos_enable bit is HIGH.

See also *DMC id_<0-5>_cfg Registers at 0x0100* on page 3-21.

Formatting from the memory manager

The direct command bits [21:20] determines the memory chip to access.

The command to be carried out is either an auto refresh, or from the APB interface. The memory manager encodes it to match the format required by the arbiter.

Arbiter access mux

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

The selection between a DMC read access and a write access is made using a round-robin arbitration, unless a read access has a low latency QoS value. In this case it is arbitrated immediately.

QoS

For write accesses, no quality of service is provided.

If the QoS enable bit for the port 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.

A QoS is also provided for the auto-refresh commands from the memory manager. The arbiter keeps track of the number of auto-refresh commands in the arbiter queue with a simple increment-decrement counter. If the number of auto-refresh commands reaches a set limit of six, a refresh time-out is signalled to the arbiter queue. This forces all of the auto-refresh queue entries to have a time-out. 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

There are two types of hazard:

Read After Read (RAR)

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

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 *Read-After-Write* (RAW) or *Write-After-Rea*d (WAR) hazard checks in the DMC.

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

This is combinatorial logic that selects the read pointer from the current queue state.

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

1. read min-latency timeout
2. read max-latency timeout.

If the last command was an open-row access:

- next access to a bank, for which there are no pending open-row accesses, that is an open-row miss.

If the last open-row access was a read:

1. open-row read
2. open-row write.

Else:

1. open-row write
2. open-row read
3. open-row miss.

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.

2.5.5 Memory manager operation

The major functions of the memory manager are described in:

- *DMC tracking and control*
- *Issuing commands to memory* on page 2-32.

DMC tracking and control

The memory manager tracks and controls the state of operation of the DMC. This is controlled by the **dmc_aclk** domain FSM and it can be traversed by writing to the dmc_memc_cmd Register. The state of this FSM can only be traversed when the DMC is idle. For example, the Ready state can only be entered from the Config state when all direct commands are completed.

Issuing commands to memory

The commands that the memory manager issues to memory are:

Direct commands

These are received over the APB interface as a result of a write to the dmc_direct_cmd Register. See *DMC Direct Command Register at 0x0008* 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.

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 DMC to be scaled without affecting the refresh rate. You can program the refresh rate period using the dmc_refresh_prd Register. The value of this register is the count value in **dmc_mclk** cycles.

When the refresh counter wraps around zero, an individual auto-refresh sequence is requested for each external chip in turn.

Programmable options

Auto Refresh Request Period, also known as RefreshPrd, is the time period in **dmc_mclk** clock cycles during which 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 operation* on page 2-27.

2.5.6 APB slave interface operation

The APB interface is clocked by the same clock as the AHB domain clock, **dmc_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 DMC is not in the config or Low-power state are the Memory Controller Status Register, to read the current state, and the Memory Controller Command Register to change state.

To guarantee no missed auto-refresh commands it is recommended that any change of **dmc_mclk** period, and therefore update of the refresh period, is carried out when the DMC is in the Low-power state. This is because the refresh rate is dependent on the **dmc_mclk** period. It is recommended that direct commands to the external memories are only written when the DMC is in the Config state and not in the Low-power state.

2.5.7 Memory interface operation

The interface is separated from the arbiter using three configurable synchronous or asynchronous 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.

You can program all the timing parameters in Figure 2-16 on page 2-34 to Figure 2-27 on page 2-38 using the APB interface. See Chapter 3 *Programmer's Model*.

When the auto_power_down Register bit is set then the **dmc_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 2-24 on page 2-37. 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 DMC apart from the memory interface and pad interface the state of the memory is known. When the rest of the DMC 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 2-16 to Figure 2-27 on page 2-38 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 the data sheet of the SDRAM device. Figure 2-16 to Figure 2-27 on page 2-38 show how the data sheet timings map onto the DMC timing registers.

The times in Figure 2-16 to Figure 2-27 on page 2-38 are not necessarily the default timing values but are values that are small enough to show the entire delay in one figure.

Note

The **command_en**, **data_cntl_en**, and **read_en** signals are internal to the DMC.

Figure 2-16 shows the command control output timing.

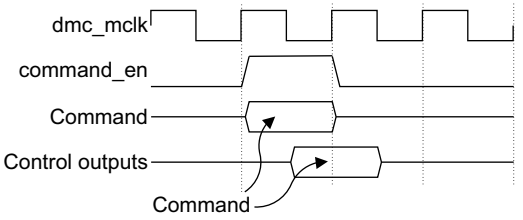


Figure 2-16 Command control output timing

Figure 2-17 shows the activate to read or write command timing.

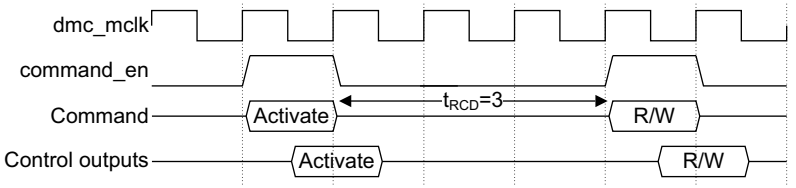


Figure 2-17 Activate to read or write command timing, t_{RCD}

Figure 2-18 shows the bank activate to bank activate or auto-refresh command timing.

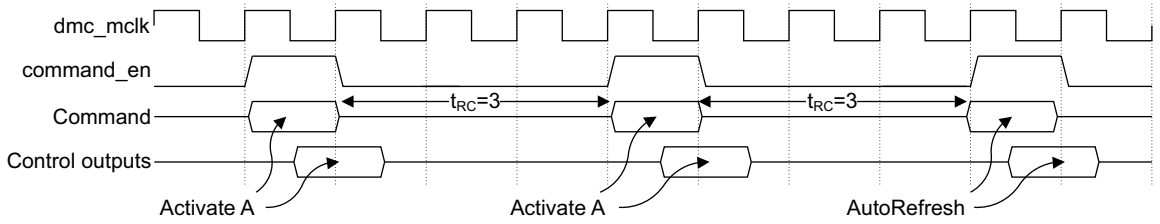


Figure 2-18 Bank activate to bank activate or auto-refresh command timing, t_{RC}

Figure 2-19 shows the bank activate to different bank activate for a memory timing.

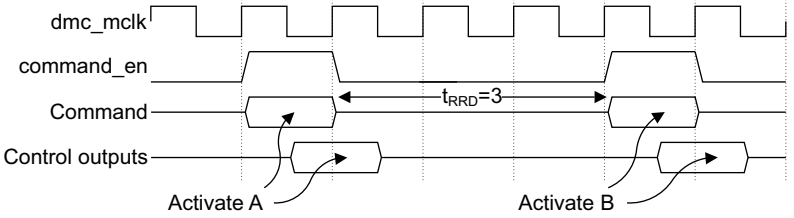


Figure 2-19 Bank activate to different bank activate for a memory timing, t_{RRD}

Figure 2-20 shows the precharge to command and auto-refresh timing.

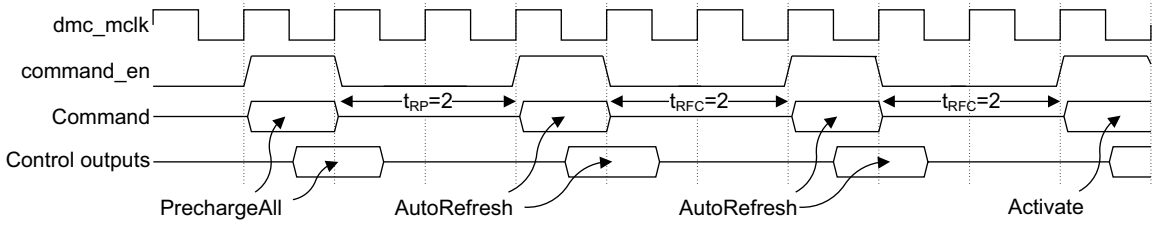


Figure 2-20 Precharge to command and auto-refresh timing, t_{RP} and t_{RFC}

Figure 2-21 shows activate to precharge, and precharge to precharge timing.

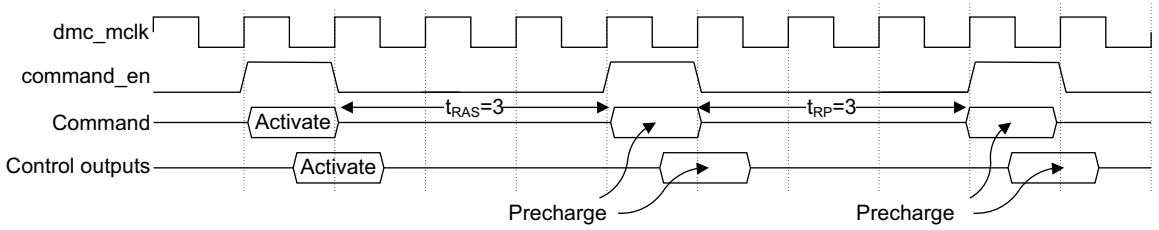


Figure 2-21 Activate to precharge, and precharge to precharge timing, t_{RAS} and t_{RP}

Figure 2-22 shows mode register write to command timing.

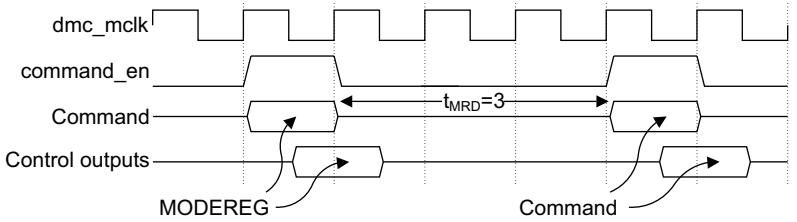


Figure 2-22 Mode register write to command timing, t_{MRD}

Figure 2-23 shows the self-refresh entry and exit timing.

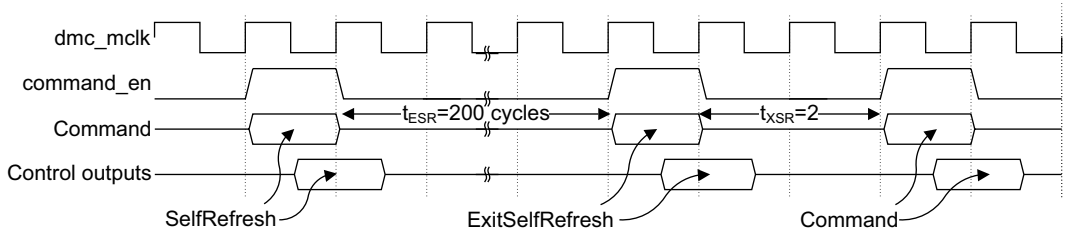


Figure 2-23 Self-refresh entry and exit timing, t_{ESR} and t_{XSR}

Figure 2-24 shows power-down entry and exit timing.

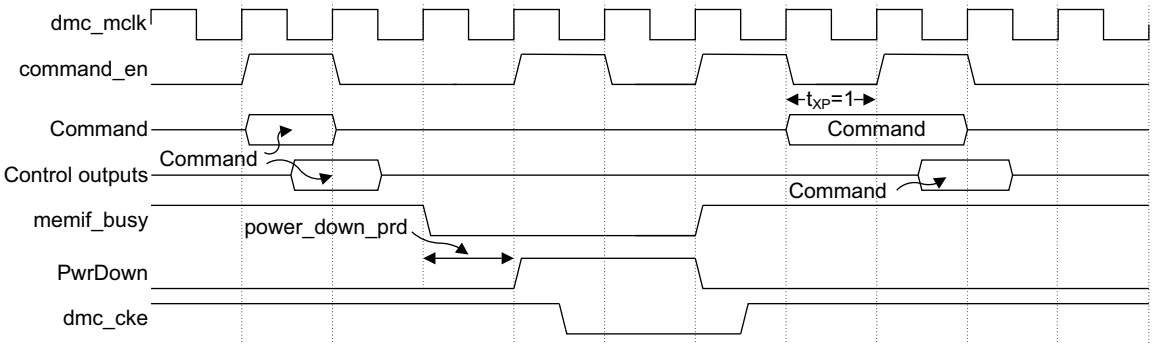


Figure 2-24 Power down entry and exit timing, t_{XP}

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. **dmc_cke** is asserted when the command FIFO is not empty.

Figure 2-25 shows the turnaround time, t_{WTR}, for the memory interface to output a Write command followed immediately by a Read command.

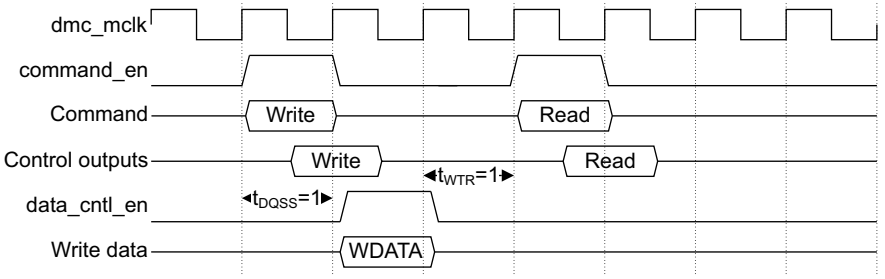


Figure 2-25 Data output timing, t_{WTR}

Figure 2-26 on page 2-38 shows the relationship between memory interface outputting the Write command and the WDATA when t_{DQSS} is set to 1. It also highlights the t_{WR} minimum time between a Write and a Precharge command.

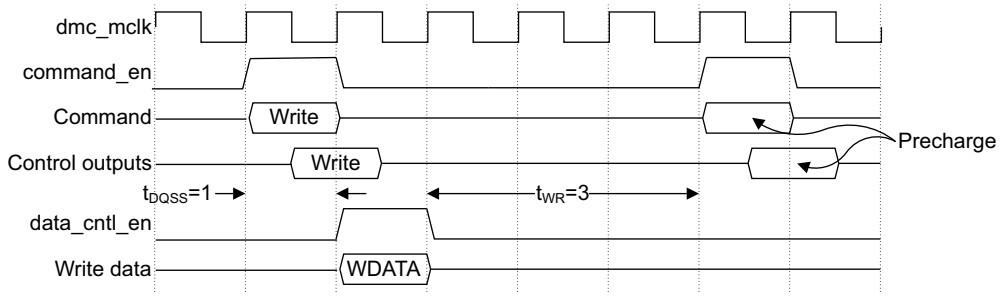


Figure 2-26 Data output timing, t_{DQSS} = 1

Figure 2-27 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.

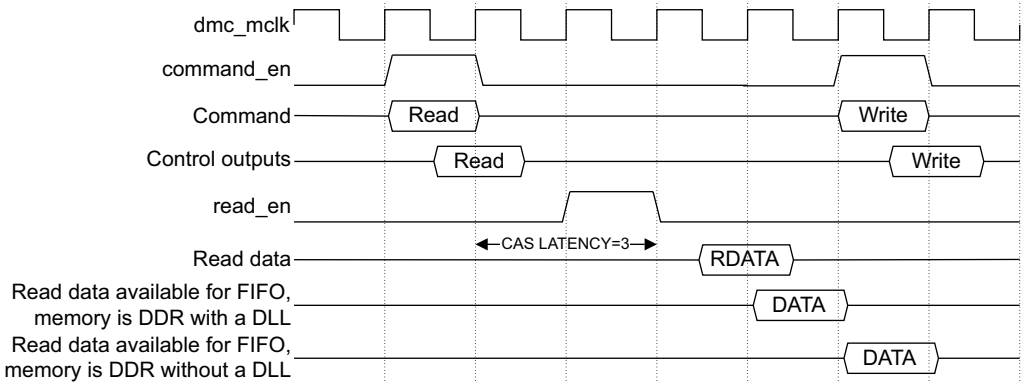


Figure 2-27 Data input timing

2.5.8 Pad interface operation

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 **dmc_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 **dmc_mclk** a set time from the command being clocked out. See Figure 2-27 on page 2-38. This delay is dependent on the current **cas_latency** value programmed and the delays in the pad interface.

2.5.9 Initialization

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

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

You might not have to configure all the DMC registers because some might power-up to the correct value, see Chapter 3 *Programmer's Model*. For completeness, all register values are included in Table 2-3.

Note

You might create a deadlock situation if the DMC interface is accessed before the DMC is configured to use the APB port.

Example setup

Table 2-3 shows an example DDR setup.

Table 2-3 Example DDR setup

Register address	Write data	Description
0x0014	0x00000004	Set cas_latency to 2
0x0018	0x00000001	Set t_dqss to 1
0x001C	0x00000002	Set t_mrd to 2
0x0020	0x00000007	Set t_ras to 7
0x0024	0x0000000B	Set t_rc to 11
0x0028	0x00000015	Set t_rcd to 5 and schedule_rcd to 2
0x002C	0x000001F2	Set t_rfc to 18 and schedule_rfc to 15
0x0030	0x00000015	Set t_rp to 5 and schedule_rp to 2
0x0034	0x00000002	Set t_rrd to 2

Table 2-3 Example DDR setup (continued)

Register address	Write data	Description
0x0038	0x00000003	Set t_wr to 3
0x003C	0x00000002	Set t_wtr to 2
0x0040	0x00000001	Set t_xp to 1
0x0044	0x0000000A	Set t_xsr to 10
0x0048	0x00000014	Set t_esr to 20
0x000C	0x00610009	Set memory configuration[a]
0x0010	0x00000640	Set auto refresh period to be every 1600 **dmc_mclk** periods
0x0200	0x000000FF	Set chip select for chip 0 to be 0x00XXXXXX, rbc configuration
0x0204	0x000040FF	Set chip select for chip 1 to be 0x40XXXXXX, rbc configuration
0x0208	0x000080FF	Set chip select for chip 2 to be 0x80XXXXXX, rbc configuration
0x020C	0x0000C0FF	Set chip select for chip 3 to be 0xC0XXXXXX, rbc configuration
0x0008	0x000C0000	Carry out chip0 Nop command
0x0008	0x00000000	Carry out chip0 Prechargeall command
0x0008	0x00090000	Extended mode register setup
0x0008	0x00080122	Mode register setup
0x0008	0x00000000	Precharge all
0x0008	0x00040000	Carry out chip0 Autorefresh command
0x0008	0x00040000	Carry out chip0 Autorefresh command
0x0008	0x00080032	Carry out chip0 Mode Reg command 0x32 mapped to low add bits
0x0008	0x001C0000	Carry out chip1 Nop command
0x0008	0x00100000	Carry out chip1 Prechargeall command
0x0008	0x00190000	Extended mode register setup
0x0008	0x00180122	Mode register setup
0x0008	0x00100000	Precharge all

Table 2-3 Example DDR setup (continued)

Register address	Write data	Description
0x0008	0x00140000	Carry out chip1 Autorefresh command
0x0008	0x00140000	Carry out chip1 Autorefresh command
0x0008	0x00180032	Carry out chip1 Mode Reg command 0x32 mapped to low add bits
0x0008	0x002C0000	Carry out chip2 Nop command
0x0008	0x00200000	Carry out chip2 Prechargeall command
0x0008	0x00290000	Extended mode register setup
0x0008	0x00280122	Mode register setup
0x0008	0x00200000	Precharge all
0x0008	0x00240000	Carry out chip2 Autorefresh command
0x0008	0x00240000	Carry out chip2 Autorefresh command
0x0008	0x00280032	Carry out chip2 Mode Reg command 0x32 mapped to low add bits
0x0008	0x003C0000	Carry out chip3 Nop command
0x0008	0x00300000	Carry out chip3 Prechargeall command
0x0008	0x00390000	Extended mode register setup
0x0008	0x00380122	Mode register setup
0x0008	0x00300000	Precharge all
0x0008	0x00340000	Carry out chip3 Autorefresh command
0x0008	0x00340000	Carry out chip3 Autorefresh command
0x0008	0x00380032	Carry out chip3 Mode Reg command 0x32 mapped to low add bits
0x0004	0x00000000	Change DMC state to ready

a. The memory is configured as follows:
- 9 column bits, 12 row bits
- precharge all bit is shared with A10
- power-down period of 0
- auto power down is disabled
- dynamic clock stopping is disabled
- memory burst size of 4.

2.5.10 Power-down support and usage model

The DMC provides architectural support for low-power operation in three ways:

- By using the dmc_memory_cfg Register at address offset `0x000C`. The DMC can automatically place the SDRAM into either the precharge power-down or active power-down states by deasserting **dmc_cke** when the SDRAM has been inactive for a number of clock cycles. This occurs with the DMC in its Ready state.
- By using the dmc_memc_cmd and dmc_memc_status Registers at address offsets `0x0004` and `0x0000`. The DMC can place the SDRAM into the Self-refresh state under software control.
- By using the DMC low-power interface, the DMC can place the SDRAM into the Self-refresh state under hardware control.

Additionally, the DMC microarchitecture provides additional power savings through extensive use of clock gating. This includes clock gating of the external memory clocks by selecting the stop_mem_clock bit in the dmc_memory_cfg Register.

It is possible to implement DMC with two power domains:

- **dmc_aclk**
- **dmc_mclk**.

See Figure 2-2 on page 2-4.

Table 2-4 shows the valid system states of the **dmc_aclk** domain FSM and the **dmc_mclk** domain FSM. It also shows the valid power, clock, and reset states in the **dmc_aclk** and **dmc_mclk** domains. Figure 2-28 on page 2-45 shows the valid transitions, and the text following it explains how to traverse the system states.

Table 2-4 Valid system states for FSMs

SDRAM		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
0	Null	>0	Running	No	Config	>0	Running	No	Powered_ up	4

Table 2-4 Valid system states for FSMs (continued)

SDRAM		DMC								System state
		aclk FSM				mclk FSM				
V_{DD}	State	V_{DD}	Clock	Reset	State	V_{DD}	Clock	Reset	State	
>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 and 13 are similar and the recommendation is to use state 12 from this pair. Table 2-5 shows a recommended set of power states.

Table 2-5 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 2-28 on page 2-45.

Note

Arcs are lines between states.

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

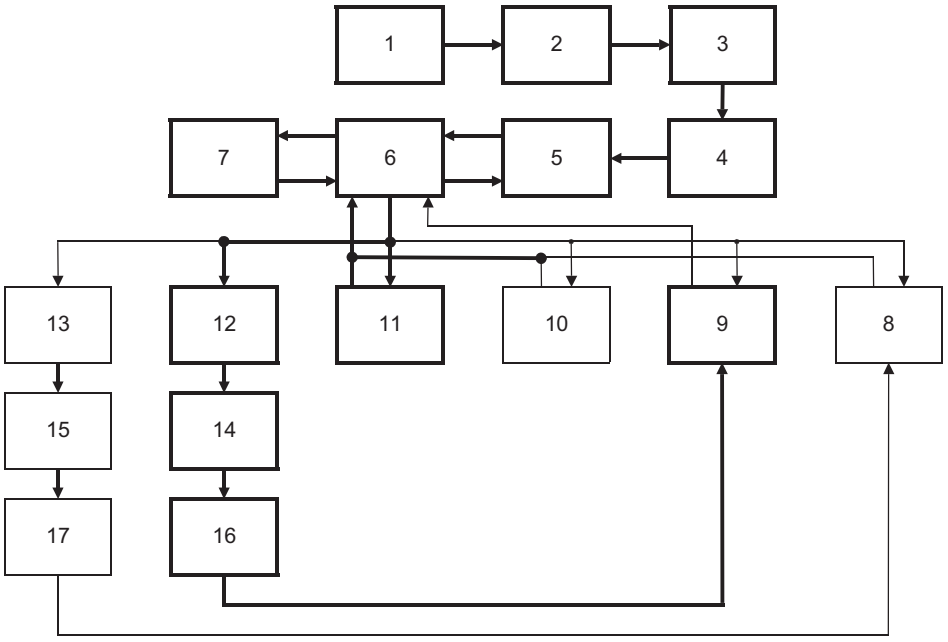


Figure 2-28 DMC system state transitions

State transitions are as follows:

Arc 1 to 2 Apply power to all DMC power domains, and ensure that **dmc_aclk** and **dmc_mclk** are running.

Arc 2 to 3 Assert reset in both the **dmc_aclk** reset domain and the **dmc_mclk** reset domain.

Arc 3 to 4 Deassert reset in both the **dmc_aclk** reset domain and the **dmc_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 `0x0014` to `0x0044`.
2. Write to the dmc_memory_cfg and dmc_refresh_prd Registers, address offsets `0x000C` and `0x0010`.

3. Initialize the memory, using the dmc_direct_cmd Register, offset `0x0008`, with the sequence of commands specified by the memory vendor. When you have sent these commands to the memory, you can write to the dmc_memc_cmd Register, offset `0x0004` with the GO command, `0x0`.
4. Poll the dmc_memc_status Register until the value of `0x1` is returned, READY, signifying that the DMC is ready to accept accesses to the SDRAM.

Arc 6 to 5 If you want to reconfigure either the DMC or SDRAM, you must first write to the dmc_memc_cmd Register, offset `0x0004`, with the Pause command, `0x3`, and poll the dmc_memc_status Register until the value of `0x2` is returned, Paused. Then you can write to the dmc_memc_cmd Register with the Configure command, `0x4` and poll the dmc_memc_status Register until the value of `0x0000` is returned, Config.

Arc 6 to 7 If auto_power_down is set in the dmc_memory_cfg Register (see *DMC Memory Configuration Register at 0x000C* on page 3-11) then this arc is automatically taken when the SDRAM has been idle for `power_down_prd` multiplied by **dmc_mclk** cycles, for example, 10 x **dmc_mclk** cycles.

Arc 7 to 6 When an SDRAM access command has been received in the **dmc_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 DMC 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 **dmc_aclk** domain clock.

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

Arc 6 to 12 The same as arc 6 to 8, but additionally stop the **dmc_mclk** domain clock and remove power from the **dmc_aclk** power domain. You can only do this if the DMC implementation has separate power domains for **dmc_aclk** and **dmc_mclk**.

Arc 6 to 13 The same as arc 6 to 8, but additionally remove power from the **dmc_aclk** power domain. You can only do this if the DMC implementation has separate power domains for **dmc_aclk** and **dmc_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 dmc_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 DMC low-power interface to bring the DMC out of a Low-power state.

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

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

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

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

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

Arc 16 to 9 Deassert reset to the **dmc_aclk** reset domain.

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

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

Arc 17 to 8 Deassert reset to the dmc_aclk reset domain.

2.6 SMC functional operation

This section describes:

- *Operating states*
- *Clocking and resets* on page 2-49
- *Miscellaneous signals* on page 2-51
- *APB slave interface operation* on page 2-52
- *Format block* on page 2-52
- *Memory manager operation* on page 2-54
- *Interrupts operation* on page 2-60
- *Memory interface operation* on page 2-60.

2.6.1 Operating states

The operation of the SMC is based on three operating states. In this section, each state is described. Figure 2-29 shows the state machine.

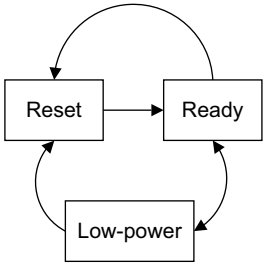


Figure 2-29 smc_aclk domain FSM diagram

The SMC states are as follows:

Reset Power is applied to the device, and **hresetn** is held LOW.

Ready Normal operation of the device. You can access the SMC register bank through the AHB configuration port and external memory devices accessed through the SMC interface.

Low-power The device does not accept new AHB transfers, and you can only access certain registers through the APB interface. You can stop the SMC clocks to reduce power consumption.

The state transitions are:

Ready to Reset

When reset is asserted to the **smc_aclk** domain, it enters the Reset state.

Reset to Ready

When reset is deasserted to the **smc_aclk** domain, it enters the Ready state.

Ready to Low-power

The Low-power state is entered when the SMC next becomes idle after either:

- The SMC receives a low-power request through the APB smc_memc_cfg_set Register
- The SMC receives a low-power request through the SMC low-power interface.

Low-power to Ready

The SMC exits the Low-power state back to Ready when either:

- The SMC low-power request bit is cleared in the APB smc_memc_cfg_clr Register
- The SMC low-power interface negates the low-power request.

Low-power to Reset

When Reset is asserted to the **smc_aclk** reset domain, it enters the Reset state.

2.6.2 Clocking and resets

This section describes:

- *Clocking*
- *Resets* on page 2-50.

Clocking

All configurations of the SMC support at least two clock domains, and have the following clock inputs:

- **smc_aclk**
- **smc_mclk0**
- **smc_mclk0n**.

These clocks can be grouped into two clock domains:

AHB domain **smc_aclk** is in this domain. You can only stop the **smc_aclk** domain signals when the SMC is in low-power mode.

Memory clock domain

The **smc_mclk0** and **smc_mclk0n** are in this domain. **smc_mclk0n** is an inverted version of **smc_mclk0**. **smc_mclk0** is used for timing and control signals.

You can tie off the **smc_async0** and **smc_msync0** pins so that the **smc_aclk** and **smc_mclk0** clock domains can operate synchronously or asynchronously with respect to each other.

Synchronous clocking

The benefit of synchronous clocking is that you can reduce the read and write latency by removing the synchronization registers between clock domains. However, because of the integer relationship of the clocks, you might not be able to get the maximum performance from the system because of constraints placed on the bus frequency by the external memory clock speed. In synchronous mode, the handshaking between the **smc_aclk** and **smc_mclk0** domains enables synchronous operation of the two clocks at multiples of each other, that is, ratios of n:1 and 1:m.

Asynchronous clocking

The main benefit of asynchronous clocking is that you can maximize the system performance, while running the memory interface at a fixed system frequency. Additionally, in sleep-mode situations when the system is not required to do much work, you can lower the frequency to reduce power consumption.

Output clocks

A clock output is provided for every external memory device on the SRAM memory interface type.

Resets

The SMC has two reset inputs:

hresetn This is the reset signal for the **smc_aclk** domain.

smc_mreset0n

This is the reset signal for the **smc_mclk0** domain.

You can change both reset signals asynchronously to their respective clock domain. Internally to the SMC the deassertion of the **hresetn** signal is synchronized to **smc_aclk**. The deassertion of **smc_mreset0n** is synchronized internally to **smc_mclk0** and **smc_mclk0n**.

2.6.3 Miscellaneous signals

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

smc_user_config[7:0]

General purpose output ports that are driven directly from the write-only APB register. If you do not require these ports leave them unconnected. See also the *SMC User Configuration Register at 0x1204* on page 3-45.

smc_user_status[7:0]

General purpose input ports that are readable from the APB interface through the smc_user_status Register. If you do not require these ports then 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 SMC **smc_aclk** signal, then you require external synchronization registers. See also the *SMC User Status Register at 0x1200* on page 3-44.

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

smc_a_gt_m0_sync

When HIGH, indicates that **smc_aclk** is greater than and synchronous to **smc_mclk0**.

smc_async0 When HIGH, indicates **smc_aclk** is synchronous to **smc_mclk0**. Otherwise they are asynchronous. Ensure that **smc_async0** is tied to the same value as **smc_msync0**.

smc_dft_en_clk_out

Use this signal for *Automatic Test Pattern Generator* (ATPG) testing only. Tie it LOW for normal operation.

smc_msync0

When HIGH, indicates **smc_mclk0** is synchronous to **smc_aclk**. Otherwise they are asynchronous. Ensure that **smc_msync0** is tied to the same value as **smc_async0**.

smc_rst_bypass

Use this signal for ATPG testing only. Tie it LOW for normal operation.

smc_use_ebi

When HIGH, indicates that the SMC must operate with a PrimeCell EBI. See the *ARM PrimeCell External Bus Interface (PL220) Technical Reference Manual*.

2.6.4 APB slave interface operation

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 during the first cycle of the access phase.

In two 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 an APB access is received and a previous direct command has not completed.

2.6.5 Format block

This section describes:

- *Hazard handling*
- *SRAM memory accesses* on page 2-53.

Hazard handling

There are four types of hazard:

- *Read After Read* (RAR)
- *Write After Write* (WAW)
- *Read After Write* (RAW)
- *Write After Read* (WAR).

The AHB interface deals with RAW hazards. WAR hazards do not occur in the AHB.

The SMC ensures the ordering of read transfers from a single port is maintained, RAR, and additionally that the ordering of write transfers from a single master is maintained, WAW.

SRAM memory accesses

This section describes:

- *Standard SRAM access*
- *Memory address shifting*
- *Memory burst alignment*
- *Memory burst length* on page 2-54
- *Booting using the SRAM* on page 2-54.

Standard SRAM access

The programmer's view is a flat area of memory.

The base addresses of external memory devices are defined by the **smc_address_match0_<0-3>[7:0]** and **smc_address_mask0_<0-3>[7:0]** tie-off pins. You can read the values of these tie-off pins through the opmode registers.

Memory address shifting

To produce the address presented to the memory device, the AHB address is aligned to the memory width. This is done because the AHB address is a byte-aligned address, while the memory address is a memory-width-aligned address.

Note

During initial configuration of a memory device, the memory mode register can be accessed with a sequence of transfers to specific addresses. You must take into consideration the shifting performance by the SMC when accessing memory mode registers.

Memory burst alignment

The SMC provides a programmable option for controlling the formatting of memory transfers with respect to memory burst boundaries, through the `burst_align` bit of the opmode registers.

When set, the `burst_align` bit causes memory bursts to be aligned to a memory burst boundary. This setting is intended for use with memories that use the concept of internal pages. This can be an asynchronous page mode memory, or a synchronous PSRAM. If a burst crosses a memory burst boundary, the SMC partitions the transfer into multiple

memory bursts, terminating a memory transfer at the burst boundary. Also ensure the page size is an integer multiple of the burst length, to avoid a memory burst crossing a page boundary.

When the `burst_align` bit is not set, the SMC ignores the memory burst boundary when mapping commands onto memory commands. This setting is intended for use with devices such as NOR flash. These devices have no concept of pages.

Memory burst length

The SMC enables you to program the memory burst length on an individual chip basis, from length 1 to 32 beats, or a continuous burst. However, the length of memory bursts are limited by the size of the read and write data FIFOs, and the programmed memory burst must not exceed this upper limit.

For read transfers, the maximum memory burst length is the depth of the read data FIFO, and that is four. For writes, the burst length is the depth of the write data FIFO, and that is four.

Booting using the SRAM

The SMC enables the lowest SRAM chip select, normally chip 0, to be bootable. To enable SRAM memory to be bootable, the SRAM interface does not require any special functionality, other than knowing the memory width of the memory concerned. This is indicated by a top-level tie-off. To enable the SMC to work with the slowest memories, the timing registers reset to the worst case values. When the **smc_remap_0** port signal is HIGH, the memory with the bootable chip select is set by the **smc_sram_mw_0[1:0]** tie-off port signal.

Additionally, while the SMC input **smc_remap_0** is HIGH, the bootable chip is aliased to base address `0x0`.

2.6.6 Memory manager operation

The memory manager module is responsible for controlling the state of the SMC and the updating of chip configuration registers.

This subsection describes:

- *Low-power operation* on page 2-55
- *Chip configuration registers* on page 2-55
- *Direct commands* on page 2-56.

Low-power operation

The SMC accepts requests to enter the Low-power state through either the SMC low-power interface or the APB register interface.

The SMC does not enter the Power-down state until it has received an idle indication from all areas of the peripheral, that is:

- there is no valid transfer held in the Format block
- there are no valid transfers held in the SMC interface
- all FIFOs are empty
- all memory interface blocks are IDLE.

When the Low-power state is entered, no new memory transfers are accepted until the SMC has been moved out of Low-power state. The SMC does not request to move out of Low-power state, and never refuses a power-down request.

Chip configuration registers

The SMC provides a mechanism for synchronizing the switching of operating modes of the SMC with that of the memory device.

The smc_set_cycles Register and smc_set_opmode Register act as holding registers for new operating parameters until the SMC detects the memory device has switched modes.

Figure 2-30 on page 2-56 shows the memory manager containing a bank of registers for each memory chip supported by the SMC. The manager register bank consists of all the timing parameters smc_sram_cycles0_<0-3> and smc_opmode0_<0-3>, that are required for the controller to correctly time any type of access to a supported memory type.

The APB registers smc_set_cycles and smc_set_opmode act as holding registers, the configuration registers within the manager are only updated if either:

- the smc_direct_cmd Register indicates only a register update is taking place
- the smc_direct_cmd Register indicates either a modereg operation or an memory access has taken place, and is complete.

The chip configuration registers are available as read-only registers in the address map of the APB interface.

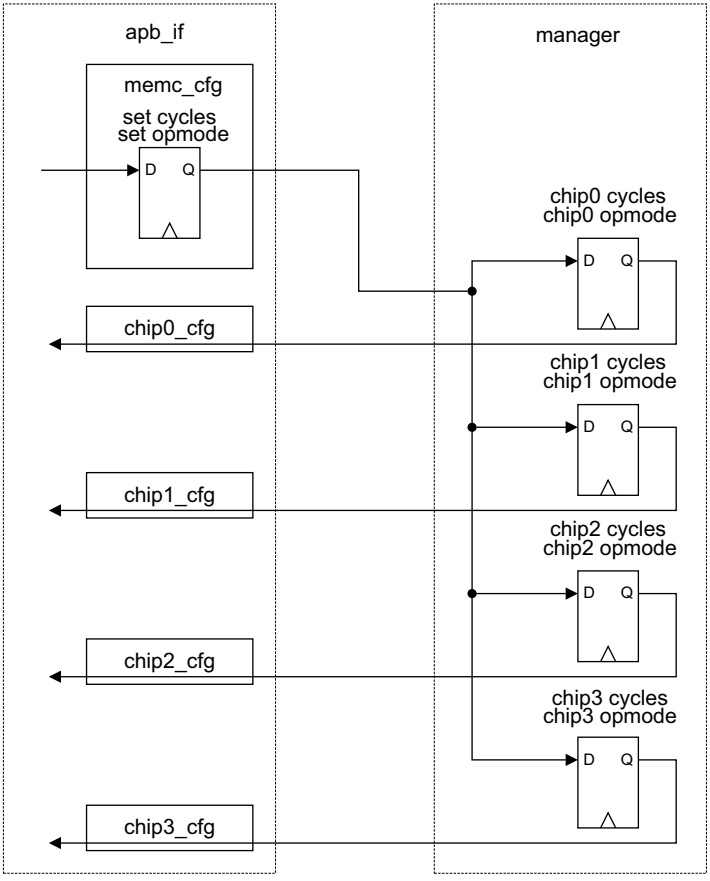


Figure 2-30 Chip configuration registers

Direct commands

The SMC enables code to be executed from the memory while simultaneously, from the software perspective, moving the same chip to a different operating mode. This is achieved by synchronizing the update of the chip configuration registers from the holding registers with the dispatch of the memory configuration register write.

The SMC provides two mechanisms for simultaneously updating the controller and memory configuration registers.

Device pin mechanism

For memories that use an input pin to indicate that a write is intended for the configuration register, for example some PSRAM devices, the write mechanism can be done through the APB direct command register. Figure 2-31 on page 2-58 shows the sequence of events.

Software mechanism

For memories that require a sequence of read and write commands, for example, most NOR Flash devices use the SMC interface, with the write data bus indicating when the last transfer has completed and when it is safe for the SMC to update the chip configuration registers. Figure 2-32 on page 2-59 shows the sequence of events.

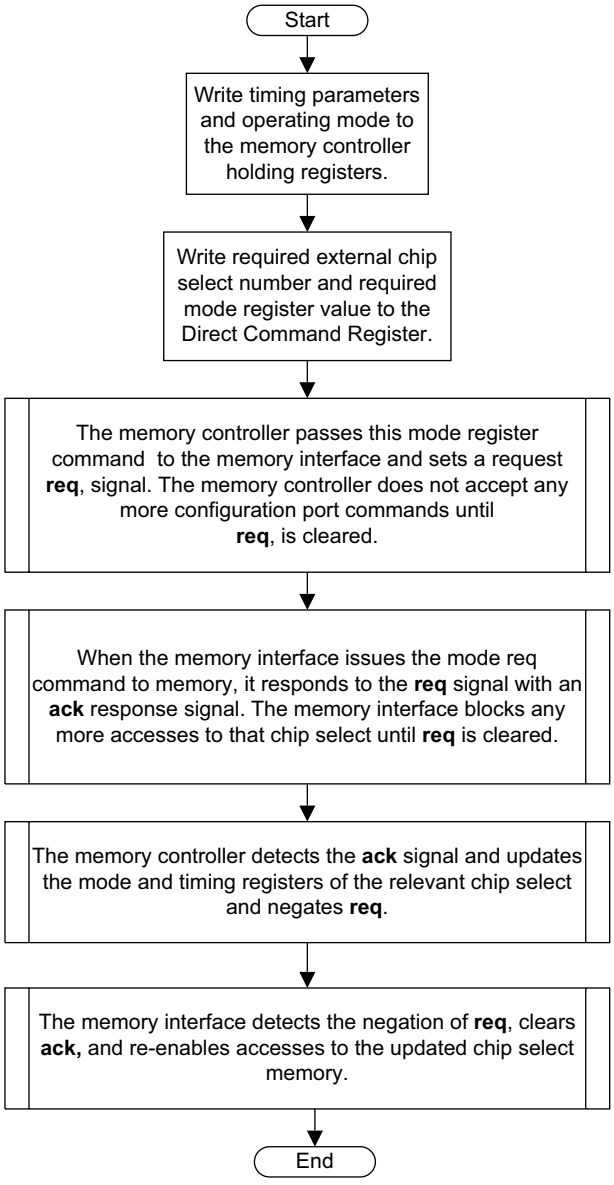


Figure 2-31 Device pin mechanism

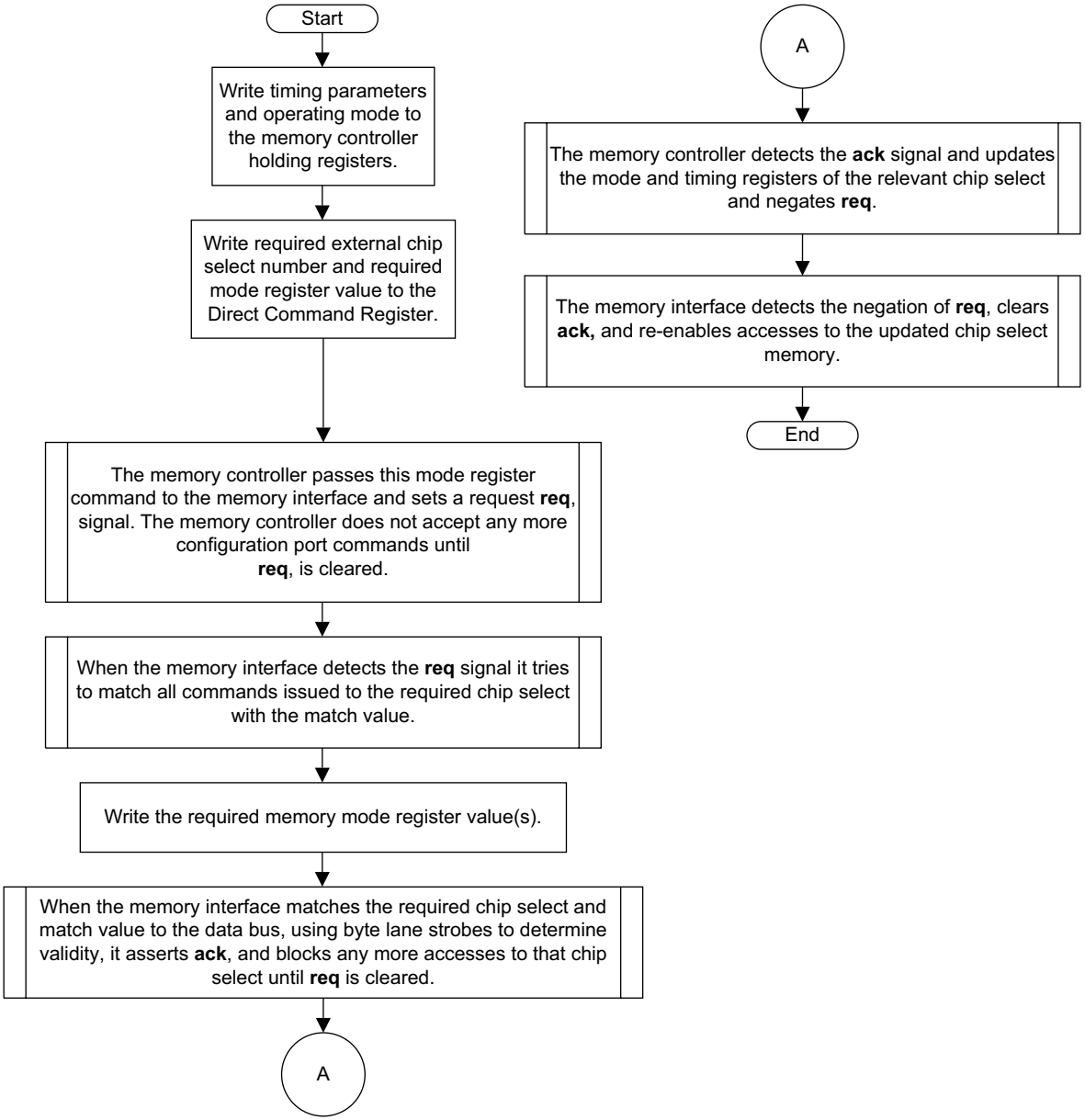


Figure 2-32 Software mechanism

2.6.7 Interrupts operation

The next read to any chip select on the appropriate memory interface clears the interrupt.

The interrupt outputs are generated through a combinational path from the relevant input pin. This enables you to place the SMC in Low-power state, and to stop the clocks, while waiting for an interrupt.

When interrupts are disabled, a synchronized version of the interrupt input is still readable through the APB interface.

2.6.8 Memory interface operation

The memory interface issues commands to the memory from the command FIFO, and controls the cycle timings of these commands. A new command is only issued when the previous command is complete and any turn-around times have been met. Additionally, a read command is not issued unless there is space for all the impending data in the read data FIFO.

Note

You must not set the `rd_bl` parameter in the smc_opmode Register to a value greater than the read data FIFO depth of four.

If enabled, the EBI can prevent commands being issued when the SMC is not granted the external bus.

Figure 2-33 on page 2-62 to Figure 2-42 on page 2-71 show the timing parameters. They are divided into *SRAM timing tables and diagrams*.

The internal signal **read_data** is included in the read transfer waveforms to indicate the clock edge on which data is registered by the SMC.

SRAM timing tables and diagrams

All address, control, and write data outputs of the SMC are registered on the rising edge of **smc_mclk0n**, equivalent to the falling edge of **smc_mclk0**, for both synchronous and asynchronous accesses. The clock output to memory, **smc_clk_out**, is driven directly by **smc_mclk0**, but gated to prevent toggling during asynchronous accesses, or when no transfers are occurring.

Read data output by the memory device is also registered on the rising edge of **smc_mclk0n**, equivalent to the falling edge of **smc_mclk0**, for asynchronous reads. For synchronous reads, read data is registered using the fed-back clock, **smc_fbclk_in**. For synchronous and asynchronous accesses, the data is then pushed onto the read data FIFO to be returned by the SMC interface.

This subsection describes:

- *Asynchronous read*
- *Asynchronous read in multiplexed-mode* on page 2-62
- *Asynchronous write* on page 2-63
- *Asynchronous write in multiplexed-mode* on page 2-64
- *Asynchronous page mode read* on page 2-64
- *Synchronous burst read* on page 2-65
- *Synchronous burst read in multiplexed-mode* on page 2-67
- *Synchronous burst write* on page 2-68
- *Synchronous burst write in multiplexed-mode* on page 2-69
- *Synchronous read and asynchronous write* on page 2-70.

Asynchronous read

Table 2-6 and Table 2-7 list the smc_opmode0_<0-3> and SRAM Register settings. See *SMC Register summary* on page 3-29.

Table 2-6 Asynchronous read opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	b0	b000	-	-	-	-	-	-

Table 2-7 Asynchronous read SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	b0011	-	b001	-	-	-

Figure 2-33 on page 2-62 shows a single asynchronous read transfer with an initial access time, t_{RC}, of three cycles and an output enable assertion delay, t_{CEOE}, of one cycle.

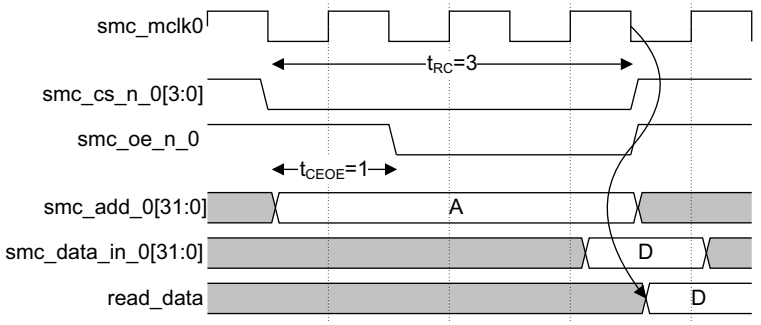


Figure 2-33 Asynchronous read

Asynchronous read in multiplexed-mode

Table 2-8 and Table 2-9 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-8 Asynchronous read in multiplexed-mode opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	b0	b000	-	-	-	b1	-	-

Table 2-9 Asynchronous read in multiplexed-mode SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	b0111	-	b101	-	-	-

Figure 2-34 shows a single asynchronous read transfer in multiplexed-SRAM mode, with t_{RC} = 7, and t_{CEOE}=5.

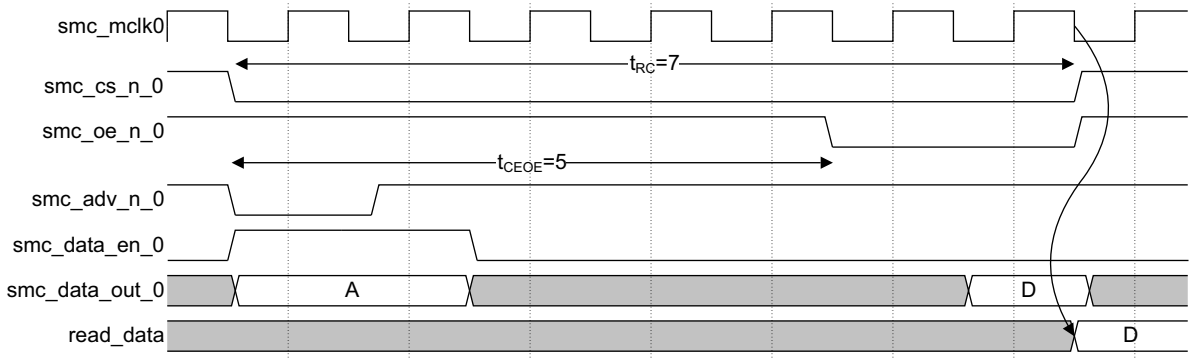


Figure 2-34 Asynchronous read in multiplexed-mode

Note

In multiplexed-mode, both address and data are output by the SMC on the **smc_data_out_0** output bus. Read data is accepted on the **smc_data_in_0** bus.

Asynchronous write

Table 2-10 and Table 2-11 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-10 Asynchronous write opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	-	-	b0	b000	-	-	-	-

Table 2-11 Asynchronous write SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	-	b0100	-	b010	-	-

Figure 2-35 shows an asynchronous write with a write cycle time t_{WC} of four cycles and a **smc_we_n_0** assertion duration, t_{WP}, of two cycles.

Note

The timing parameter t_{WC} is controlling the deassertion of **smc_we_n_0**. You can use it to vary the hold time of **smc_cs_n_0**, **smc_add_0** and **smc_data**. This differs from the read case where the timing parameter t_{CEOE} controls the delay in the assertion of **smc_oe_n_0**. Additionally, **smc_we_n_0** is always asserted one cycle after **smc_cs_n_0** to ensure the address bus is valid.

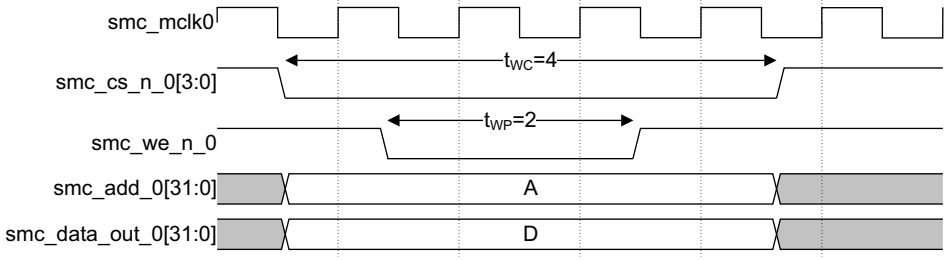


Figure 2-35 Asynchronous write

Asynchronous write in multiplexed-mode

Table 2-12 and Table 2-13 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-12 Asynchronous write in multiplexed-mode opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	-	-	b0	b000	b0	b0	-	-

Table 2-13 Asynchronous write in multiplexed-mode SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	-	b0111	-	b100	-	-

Figure 2-36 shows an asynchronous write in multiplexed-mode. t_{WC} is seven cycles. t_{WP} is four cycles, and is extended by two cycles for the address phase of the transaction.

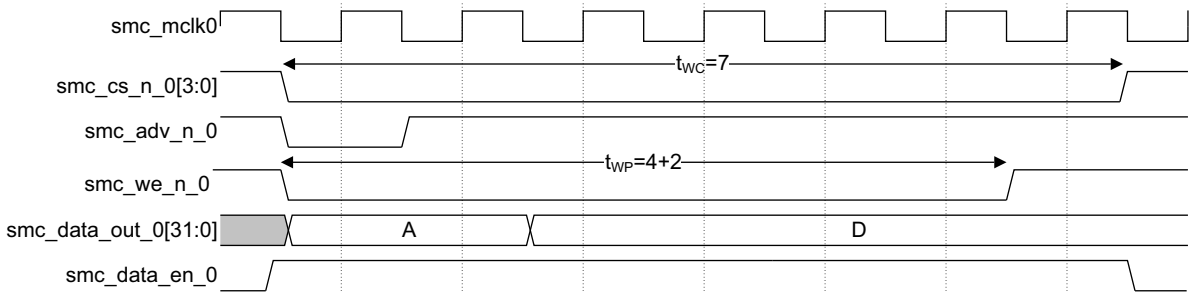


Figure 2-36 Asynchronous write in multiplexed-mode

Asynchronous page mode read

Table 2-14 and Table 2-15 on page 2-65 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-14 Page read opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	b0	<page length>	-	-	-	-	-	b1

Table 2-15 Page read SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	b0011	-	b010	-	b001	-

Figure 2-37 shows a page read access, with an initial access time, t_{RC}, of three cycles, an output enable assertion delay, t_{CEOE}, of two cycles and a page access time, t_{PC}, of one cycle.

Page mode is enabled in the SMC by setting the opmode Register for the relevant chip to asynchronous reads and the burst length to the page size.

Note

Multiplexed-mode page accesses are not supported.

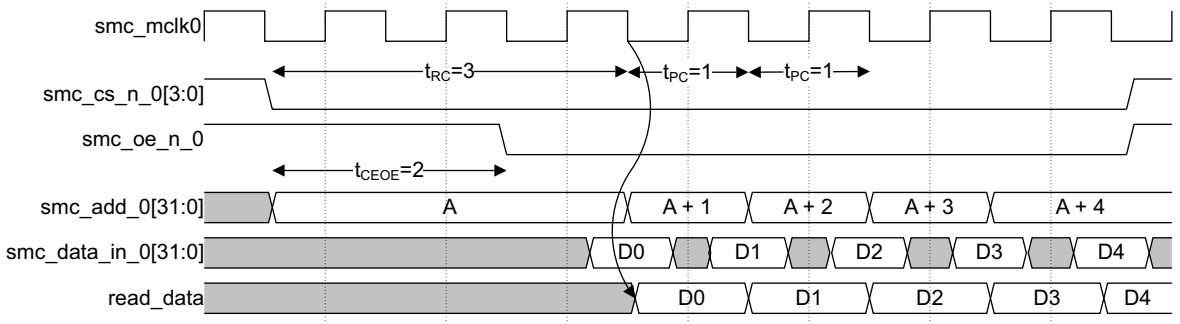


Figure 2-37 Page read

Synchronous burst read

Table 2-16 and Table 2-17 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-16 Synchronous burst read opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	b1	<burst length>	-	-	-	b1	-	-

Table 2-17 Synchronous burst read SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	b0100	-	b010	-	-	-

Figure 2-38 shows a burst read with the **smc_wait_0** output of the memory used to delay the transfer.

Note

- Synchronous memories have a configuration register enabling **smc_wait_0** to be asserted either on the same clock cycle as the delayed data or a cycle earlier. The SMC only supports **smc_wait_0** being asserted one cycle early, enabling **smc_wait_0** to be initially sampled with the fed-back clock and then with **smc_mclk0** before being used by the FSM. This enables the easiest timing closure. Additionally, you must configure the memory for **smc_wait_0** to be active LOW.
- In synchronous operation, the SMC relies on the **smc_wait_0** signal being deasserted HIGH to indicate that the memory can finish the transfer. When in synchronous mode some memories do not deassert the **smc_wait_0** signal during non-array read transfers. Non-array read transfers are typically status register reads. To avoid stalling the system with these memories, in synchronous mode you must not perform non-array read transfers with the memory and SMC.

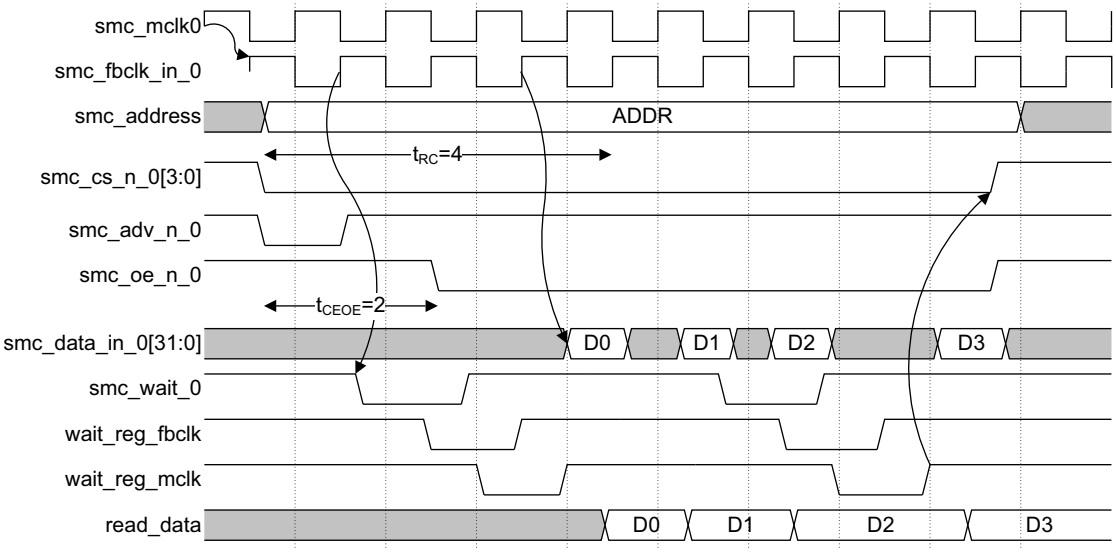


Figure 2-38 Synchronous burst read

Synchronous burst read in multiplexed-mode

Table 2-18 and Table 2-19 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-18 Synchronous burst read in multiplexed-mode opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	b1	<burst length>	-	-	-	-	-	-

Table 2-19 Synchronous burst read in multiplexed-mode read SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	b0100	-	b010	-	-	-

Figure 2-39 shows the same synchronous read burst transfer as Figure 2-38 on page 2-66, but in multiplexed-mode.

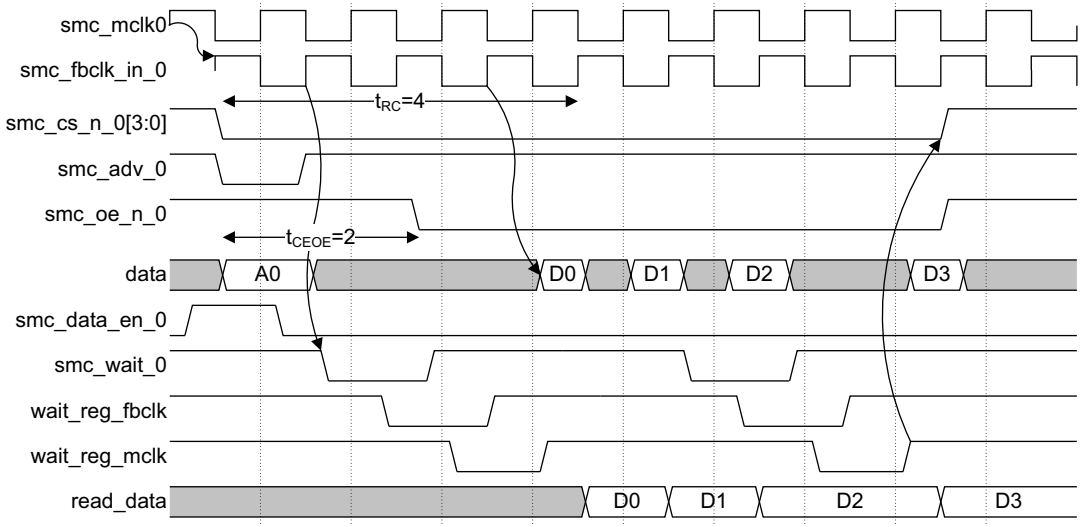


Figure 2-39 Synchronous burst read in multiplexed-mode

Synchronous burst write

Table 2-20 and Table 2-21 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-20 Synchronous burst write opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	-	-	b1	<burst length>	-	b1	-	-

Table 2-21 Synchronous burst write SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	-	b0100	-	b001	-	-

Figure 2-40 shows a synchronous burst write transfer that is delayed by the **smc_wait_0** signal. You must configure the memory to assert **smc_wait_0** one cycle early and with an active LOW priority. The **smc_wait_0** signal is again registered with the fed back clock and **smc_mclk0** before being used. The **smc_wait_0** signal is used in the **smc_mclk0** domain to the memory interface FSM.

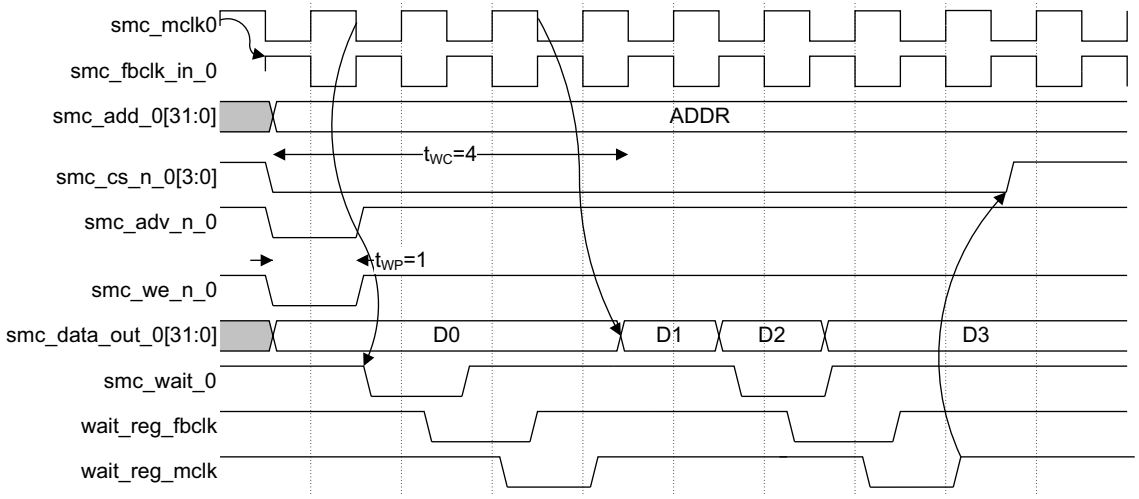


Figure 2-40 Synchronous burst write

Synchronous burst write in multiplexed-mode

Table 2-22 and Table 2-23 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-22 Synchronous burst write in multiplexed-mode opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	-	-	b1	<burst length>	-	b1	-	-

Table 2-23 Synchronous burst write in multiplexed-mode SRAM cycles register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	-	b0100	-	b001	-	-

Figure 2-41 shows the same synchronous burst write as Figure 2-40 on page 2-68, but in multiplexed-mode.

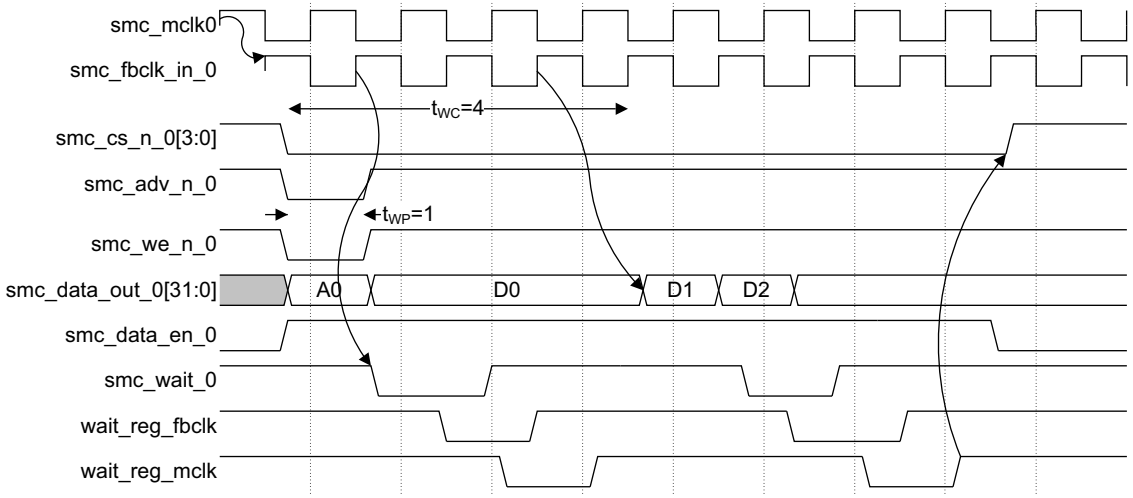


Figure 2-41 Synchronous burst write in multiplexed-mode

Synchronous read and asynchronous write

Table 2-24 and Table 2-25 list the smc_opmode0_<0-3> and SRAM Register settings.

Table 2-24 Synchronous read and asynchronous write opmode chip register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	b1	b001	b0	b000	b0	b1	b0	-

Table 2-25 Synchronous read and asynchronous write opmode chip register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	b0100	b0110	b010	b001	-	b011

Figure 2-42 on page 2-71 shows the turnaround time t_{TR}, enforced between synchronous read and asynchronous write. The turnaround time is enforced between:

- Reads followed by writes
- Writes followed by reads
- Read following a read from a different chip select.

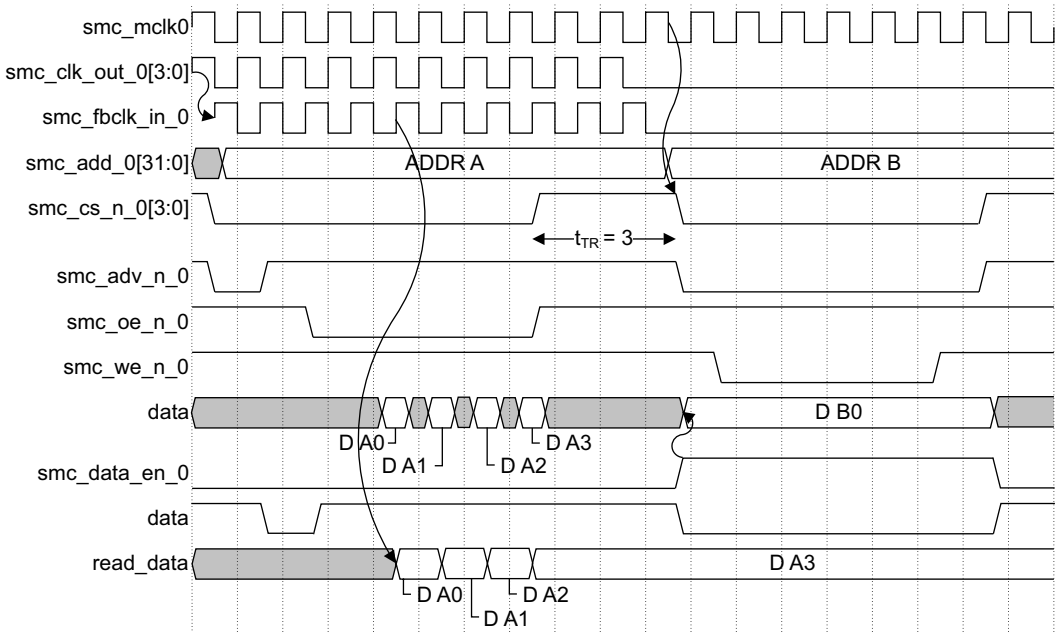


Figure 2-42 Synchronous read and asynchronous write

Programming t_{RC} and t_{WC} when the controller operates in synchronous mode

For t_{RC}:

- when using memory devices that are not wait-enabled, you must program t_{RC} to be the number of clock cycles required before valid data is available following the assertion of **cs_n**
- when using memory devices that are wait-enabled, you must program t_{RC} to be the number of clock cycles required before wait is active and stable, following the assertion of **cs_n**. That is:

  ```
  t_RC = 3 + t_CEOE
  ```

  Note

  t_CEOE is only required if **wait** is asserted when **oe_n** goes LOW.

For t_{WC}:

- when using memory devices that are not wait-enabled, you must program t_{WC} to be the number of clock cycles required before the first data is written, following the assertion of **cs_n**
- when using memory devices that are wait-enabled, you must program t_{WC} to be the number of clock cycles required before **wait** is active and stable, following the assertion of **cs_n**. That is:

  `t_WC = 3`

  Note

  If a memory device is configured so that there are two or less clock cycles between the assertion of **wait** and data being required then you must program t_{WC} as if the memory device is not wait-enabled.

Chapter 3
Programmer's Model

This chapter describes the registers of the DMC and SMC and provides information for programming the devices. It contains the following sections:

3.1 About the programmer's model

Both the DMC and SMC have 4KB of memory allocated to each of them. The DMC memory extends from a base address of `0x0000` to a maximum address of `0x0FFF`. The SMC memory extends from a base address of `0x1000` to a maximum address of `0x1FFF`. Figure 3-1 on page 3-3 shows that the register map address range is split into ten regions:

DMC and SMC configuration registers

Use these registers for the global configuration, and to control the operating state of the DMC and SMC.

DMC and SMC chip select configuration registers

These registers hold the operating parameters of each chip select.

DMC and SMC user configuration registers

These registers provide general purpose I/O for user specific applications.

DMC and SMC integration test registers

Use these registers to verify correct integration of the DMC and SMC within a system, by enabling non-AMBA ports to be set and read.

DMC and SMC PrimeCell Id registers

These registers enable the software to identify the system components.

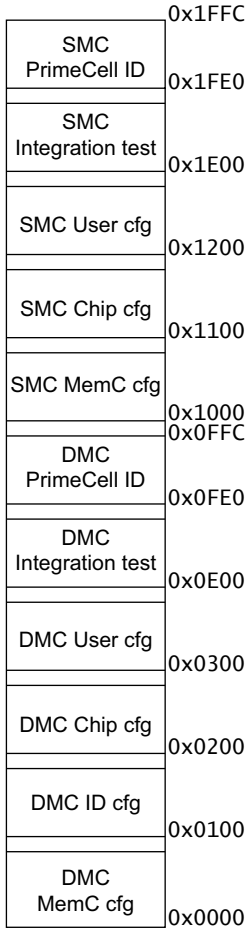


Figure 3-1 DMC and SMC register map

3.2 DMC Register summary

Figure 3-2 shows the DMC configuration register map.

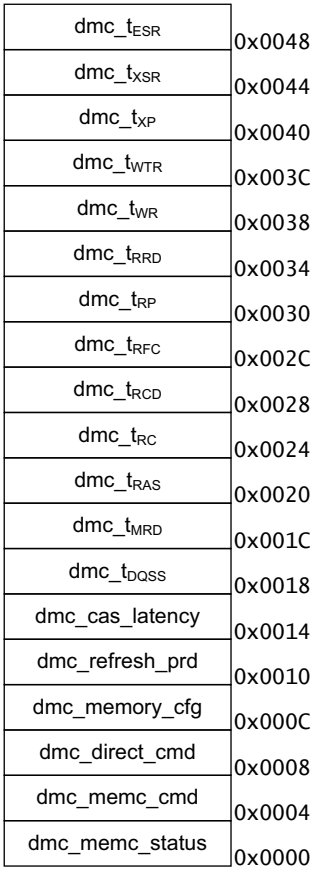


Figure 3-2 DMC configuration register map

Figure 3-3 on page 3-5 shows the DMC id configuration registers map.

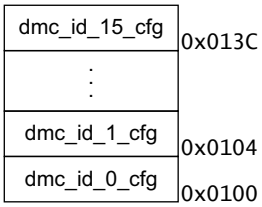


Figure 3-3 DMC id configuration register map

Figure 3-4 shows the DMC chip configuration register map.

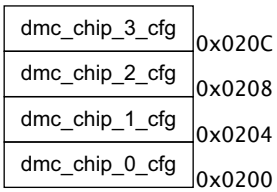


Figure 3-4 DMC chip configuration register map

Figure 3-5 shows the DMC peripheral and PrimeCell identification configuration register map.

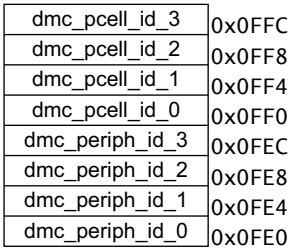


Figure 3-5 DMC peripheral and PrimeCell Identification configuration register map

Table 3-1 on page 3-6 lists the DMC registers.

Table 3-1 DMC register summary

Name	Base offset	Type	Reset value	Description
dmc_memc_status	0x000	RO	0x00000790	See *DMC Memory Controller Status Register at 0x0000* on page 3-8
dmc_memc_cmd	0x004	WO	-	See *DMC Memory Controller Command Register at 0x0004* on page 3-9
dmc_direct_cmd	0x008	WO	-	See *DMC Direct Command Register at 0x0008* on page 3-10
dmc_memory_cfg	0x00C	R/W	0x00010020	See *DMC Memory Configuration Register at 0x000C* on page 3-11
dmc_refresh_prd	0x010	R/W	0x00000A60	See *DMC Refresh Period Register at 0x0010* on page 3-13
dmc_cas_latency	0x014	R/W	0x00000006	See *DMC CAS Latency Register at 0x0014* on page 3-14
dmc_t_dqss	0x018	R/W	0x00000001	See *DMC t_dqss Register at 0x0018* on page 3-14
dmc_t_mrd	0x01C	R/W	0x00000002	See *DMC t_mrd Register at 0x001C* on page 3-15
dmc_t_ras	0x020	R/W	0x00000007	See *DMC t_ras Register at 0x0020* on page 3-15
dmc_t_rc	0x024	R/W	0x0000000B	See *DMC t_rc Register at 0x0024* on page 3-16
dmc_t_rcd	0x028	R/W	0x0000001D	See *DMC t_rcd Register at 0x0028* on page 3-16
dmc_t_rfc	0x02C	R/W	0x00000212	See *DMC t_rfc Register at 0x002C* on page 3-17
dmc_t_rp	0x030	R/W	0x0000001D	See *DMC t_rp Register at 0x0030* on page 3-17
dmc_t_rrd	0x034	R/W	0x00000002	See *DMC t_rrd Register at 0x0034* on page 3-18
dmc_t_wr	0x038	R/W	0x00000003	See *DMC t_wr Register at 0x0038* on page 3-19
dmc_t_wtr	0x03C	R/W	0x00000002	See *DMC t_wtr Register at 0x003C* on page 3-19
dmc_t_xp	0x040	R/W	0x00000001	See *DMC t_xp Register at 0x0040* on page 3-20
dmc_t_xsr	0x044	R/W	0x0000000A	See *DMC t_xsr Register at 0x0044* on page 3-20
dmc_t_esr	0x048	R/W	0x00000014	See *DMC t_esr Register at 0x0048* on page 3-21
dmc_id_<0-5>_cfg	0x100	R/W	0x00000000	See *DMC id_<0-5>_cfg Registers at 0x0100* on page 3-21

Table 3-1 DMC register summary (continued)

Name	Base offset	Type	Reset value	Description
dmc_chip_<0-3>_cfg	0x200	R/W	0x0000FF00	See *DMC chip_<0-3>_cfg Registers at 0x0200* on page 3-22
dmc_user_status	0x300	RO	-	See *DMC user_status Register at 0x0300* on page 3-23
dmc_user_config	0x304	WO	-	See *DMC user_config Register at 0x0304* on page 3-23
dmc_int_cfg	0x0E00	R/W	0x0	See *DMC Integration Configuration Register at 0x0E00* on page 4-3
dmc_int_inputs	0x0E04	RO	Tie-off dependent	See *DMC Integration Inputs Register at 0x0E04* on page 4-4
dmc_int_outputs	0x0E08	WO	-	See *DMC Integration Outputs Register at 0x0E08* on page 4-6
dmc_periph_id_<0-3>	0xFE0-0xFEC	RO	-	See *DMC Peripheral Identification Registers <0-3> at 0x0FE0-0x0FEC* on page 3-24
dmc_pcell_id_<0-3>	0xFF0-0xFFC	RO	-	See *DMC PrimeCell Identification Registers <0-3> at 0xFF0-0xFFC* on page 3-26

3.3 DMC Register descriptions

This section describes the DMC registers.

3.3.1 DMC Memory Controller Status Register at 0x0000

The read-only dmc_memc_status Register provides information on the configuration of the DMC and also the current state of the DMC. It cannot be read in either the Reset or POR states. Figure 3-6 shows the register bit assignments.

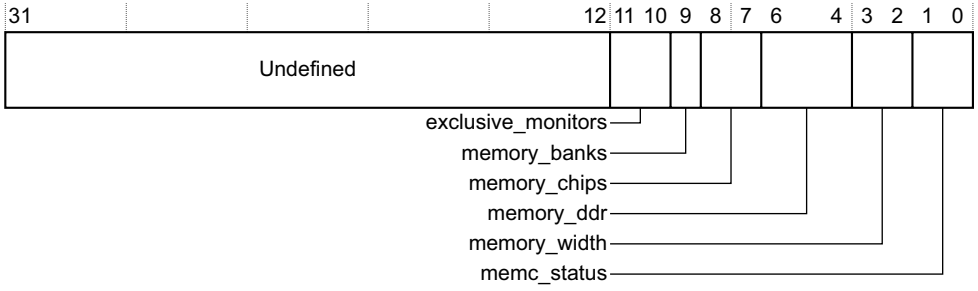


Figure 3-6 dmc_memc_status Register bit assignments

Table 3-2 lists the register bit assignments.

Table 3-2 dmc_memc_status Register bit assignments

Bits	Name	Function
[31:12]	-	Read undefined.
[11:10]	exclusive_monitors	Returns the number of exclusive access monitor resources implemented in the DMC: b00 = 0 monitors b01 = 1 monitor b10 = 2 monitors b11 = 4 monitors.
[9]	memory_banks	Returns the maximum number of banks that the DMC supports on each chip. This is fixed at 1'b1 = 4 banks.
[8:7]	memory_chips	Returns the number of different chip selects that the DMC supports: b00 = 1 chip b01 = 2 chips b10 = 3 chips b11 = 4 chips.

Table 3-2 dmc_memc_status Register bit assignments (continued)

Bits	Name	Function
[6:4]	memory_ddr	Returns the SDRAM that the DMC supports: b000 = Reserved b001 = DDR SDRAM b011 = Reserved b010 = Reserved b1xx = Reserved.
[3:2]	memory_width	Returns the width of the external memory: b00 = 16-bit b01 = 32-bit b10 = 64-bit b11 = Reserved.
[1:0]	memc_status	Returns the state of the memory controller: b00 = Config b01 = Ready b10 = Paused b11 = Low_power.

3.3.2 DMC Memory Controller Command Register at 0x0004

The write-only dmc_memc_cmd Register enables the DMC to be traversed. The command register controls the programmer's view FSM. By writing to this register the FSM can 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 3-7 shows the register bit assignments.

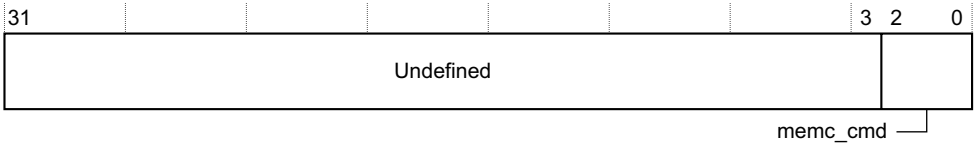


Figure 3-7 dmc_memc_cmd Register bit assignments

Table 3-3 lists the register bit assignments.

Table 3-3 dmc_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: b000 = Go b001 = Sleep b010 = Wakeup b011 = Pause b100 = Configure.

3.3.3 DMC Direct Command Register at 0x0008

The write-only dmc_direct_cmd Register passes commands to the external memory. The configuration of the dmc_direct_cmd 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 dmc_direct_cmd Register therefore enables any initialization sequence that an external memory device might require. The only timing information associated with the dmc_direct_cmd 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 in the Config or Low-power state. Figure 3-8 shows the register bit assignments.

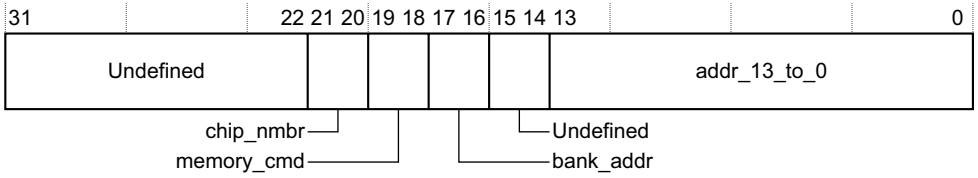


Figure 3-8 dmc_direct_cmd Register bit assignments

Table 3-4 lists the register bit assignments.

Table 3-4 dmc_direct_cmd Register bit assignments

Bits	Name	Function
[31:22]	-	Reserved. Write as zero.
[21:20]	chip_nmbr	Bits mapped to external memory chip address bits.
[19:18]	memory_cmd	Determines the command required: b00 = Prechargeall b01 = Autorefresh b10 = Modereg or Extended modereg access 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 DMC Memory Configuration Register at 0x000C

The read/write dmc_memory_cfg Register configures the memory. It can only be read and written in the Config or Low-power state. Figure 3-9 shows the register bit assignments.

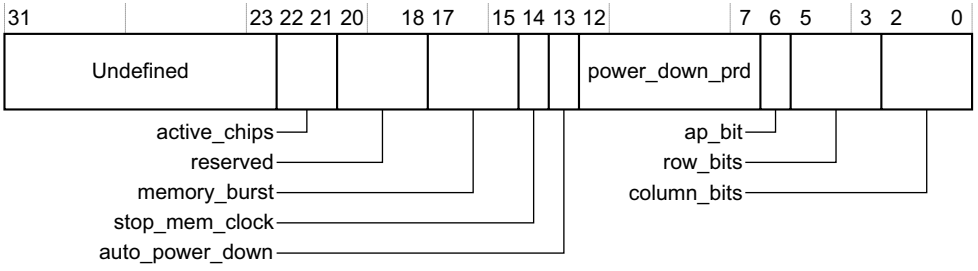


Figure 3-9 dmc_memory_cfg Register bit assignments

Table 3-5 lists the register bit assignments.

Table 3-5 dmc_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 defined in the memc_status Register: b00 = 1 chip b01 = 2 chips b10 = 3 chips b11 = 4 chips.
[20:18]	-	Reserved, read undefined. Write as zero.
[17:15]	memory_burst	Encodes the number of data accesses that are performed to the SDRAM for each read and write command: b000 = Reserved b001 = Burst 2 b010 = Burst 4 b011 = Burst 8 b100 = Burst 16. You must program this value into the SDRAM mode register using the direct_cmd Register at offset `0x8`, and it must match it.
[14]	stop_mem_clock	When enabled, the memory clock is dynamically stopped when not performing an access to the SDRAM.
[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 number of memory clock cycles.
[12:7]	power_down_prd	Number of memory clock cycles for auto power-down of the SDRAM. Minimum value = 1.
[6]	ap_bit	Encodes the position of the auto-precharge bit in the memory address: b0 = address bit 10 b1 = address bit 8.

Table 3-5 dmc_memory_cfg Register bit assignments (continued)

Bits	Name	Function
[5:3]	row_bits	Encodes the number of bits of the AHB address that comprise the row address: b000 = 11 bits b001 = 12 bits b010 = 13 bits b011 = 14 bits b100 = 15 bits b101 = 16 bits. The combination of row size, column size, BRC/RBC, and memory width must ensure that neither the MSB of the row address nor the MSB of the bank address exceeds address range [27:0].
[2:0]	column_bits	Encodes the number of bits of the AHB address that comprise the column address: b000 = 8 bits b001 = 9 bits b010 = 10 bits b011 = 11 bits b100 = 12 bits.

3.3.5 DMC Refresh Period Register at 0x0010

The read/write dmc_refresh_prd Register sets the memory refresh period. It can only be read and written in the Config or Low-power state. Figure 3-10 shows the register bit assignments.

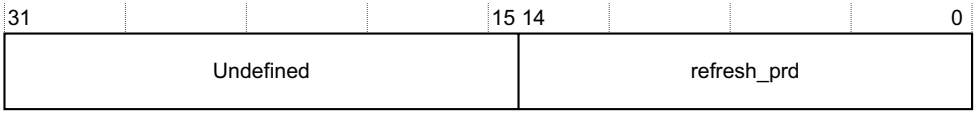


Figure 3-10 dmc_refresh_prd Register bit assignments

Table 3-6 lists the register bit assignments.

Table 3-6 dmc_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 DMC CAS Latency Register at 0x0014

The read/write dmc_cas_latency Register sets the CAS latency in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-11 shows the register bit assignments.

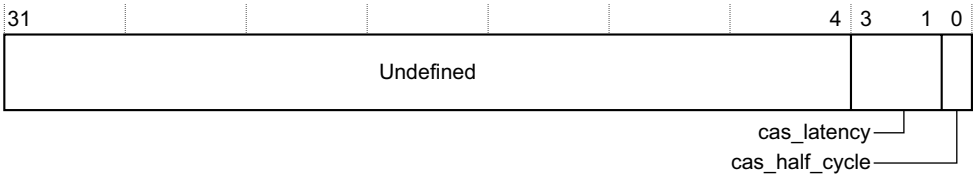


Figure 3-11 dmc_cas_latency Register bit assignments

Table 3-7 lists the register bit assignments.

Table 3-7 dmc_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.
[0]	cas_half_cycle	Encodes whether the CAS latency is half a memory clock cycle more than the value given in bits [3:1]: 1'b0 = Zero cycles offset to value in [3:1]. 1'b1 = Half cycle offset to value in [3:1].

3.3.7 DMC t_dqss Register at 0x0018

The read/write dmc_t_dqss Register writes to DQS in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-12 shows the register bit assignments.

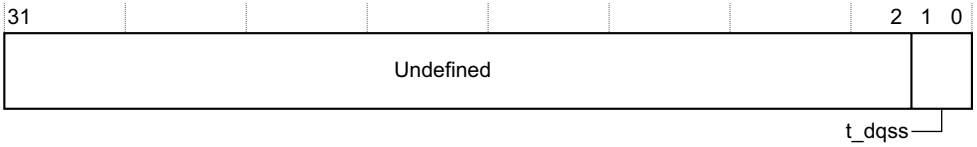


Figure 3-12 dmc_t_dqss Register bit assignments

Table 3-8 lists the register bit assignments.

Table 3-8 dmc_t_dqss Register bit assignments

Bits	Name	Function
[31:2]	-	Read undefined. Write as zero.
[1:0]	t_dqss	Write to DQS in memory clock cycles.

3.3.8 DMC t_mrd Register at 0x001C

The read/write dmc_t_mrd Register sets the mode register command time in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-13 shows the register bit assignments.

Figure 3-13 dmc_t_mrd Register bit assignments

Table 3-9 lists the register bit assignments.

Table 3-9 dmc_t_mrd Register bit assignments

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

3.3.9 DMC t_ras Register at 0x0020

The read/write dmc_t_ras Register sets the RAS to precharge delay in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-14 shows the register bit assignments.

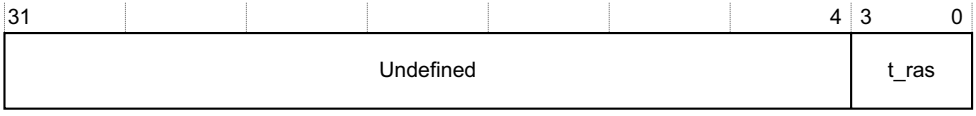


Figure 3-14 dmc_t_ras Register bit assignments

Table 3-10 lists the register bit assignments.

Table 3-10 dmc_t_ras Register bit assignments

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

3.3.10 DMC t_rc Register at 0x0024

The read/write dmc_ t_rc Register sets the Active bank x to Active bank x delay in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-15 shows the register bit assignments.

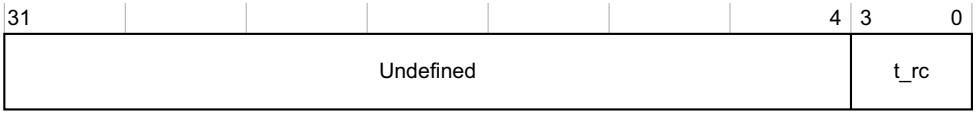


Figure 3-15 dmc_t_rc Register bit assignments

Table 3-11 lists the register bit assignments.

Table 3-11 dmc_t_rc Register bit assignments

Bits	Name	Function
[31:4]	-	Read undefined. Write as zero.
[3:0]	t_rc	Set Active bank x to Active bank x delay in memory clock cycles.

3.3.11 DMC t_rcd Register at 0x0028

The read/write dmc_ t_rcd Register sets the RAS to CAS minimum delay in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-16 shows the register bit assignments.

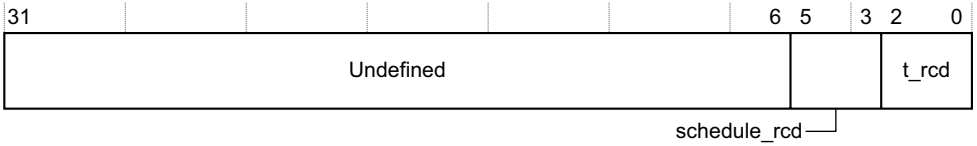


Figure 3-16 dmc_t_rcd Register bit assignments

Table 3-12 lists the register bit assignments.

Table 3-12 dmc_t_rcd Register bit assignments

Bits	Name	Function
[31:6]	-	Read undefined. Write as zero.
[5:3]	schedule_rcd	Set the RAS to CAS minimum delay in **dmc_aclk** cycles -3.
[2:0]	t_rcd	Set the RAS to CAS minimum delay in memory clock cycles.

3.3.12 DMC t_rfc Register at 0x002C

The read/write dmc_t_rfc Register sets the auto-refresh command time in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-17 shows the register bit assignments.

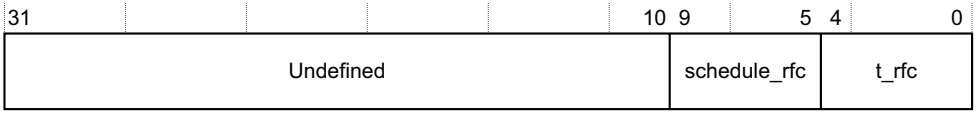


Figure 3-17 dmc_t_rfc Register bit assignments

Table 3-13 lists the register bit assignments.

Table 3-13 dmc_t_rfc Register bit assignments

Bits	Name	Function
[31:10]	-	Read undefined. Write as zero.
[9:5]	schedule_rfc	Set the autorefresh command time in **dmc_aclk** cycles -3.
[4:0]	t_rfc	Set the auto-refresh command time in memory clock cycles.

3.3.13 DMC t_rp Register at 0x0030

The read/write dmc_t_rp Register sets the precharge to RAS delay in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-18 on page 3-18 shows the register bit assignments.

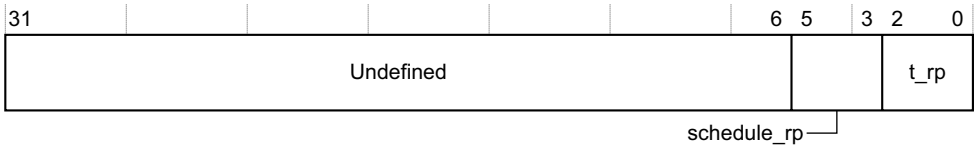


Figure 3-18 dmc_t_rp Register bit assignments

Table 3-14 lists the register bit assignments.

Table 3-14 dmc_t_rp Register bit assignments

Bits	Name	Function
[31:6]	-	Read undefined. Write as zero.
[5:3]	schedule_rp	Set the precharge to RAS delay in **dmc_aclk** cycles -3.
[2:0]	t_rp	Set the precharge to RAS delay in memory clock cycles.

3.3.14 DMC t_rrd Register at 0x0034

The read/write dmc_t_rrd Register sets the Active bank x to Active bank y delay in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-19 shows the register bit assignments.

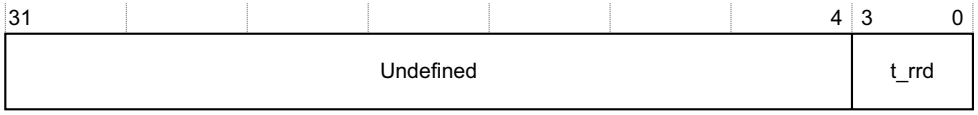


Figure 3-19 dmc_t_rrd Register bit assignments

Table 3-15 lists the register bit assignments.

Table 3-15 dmc_t_rrd Register bit assignments

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

3.3.15 DMC t_wr Register at 0x0038

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

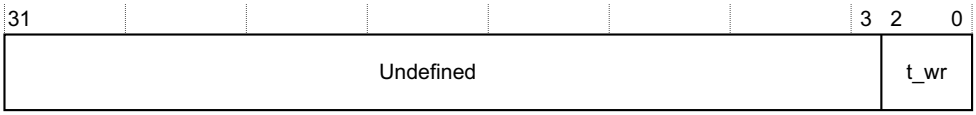


Figure 3-20 dmc_t_wr Register bit assignments

Table 3-16 lists the register bit assignments.

Table 3-16 dmc_t_wr Register bit assignments

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

3.3.16 DMC t_wtr Register at 0x003C

The read/write dmc_t_wtr Register sets the write to read delay in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-21 shows the register bit assignments.

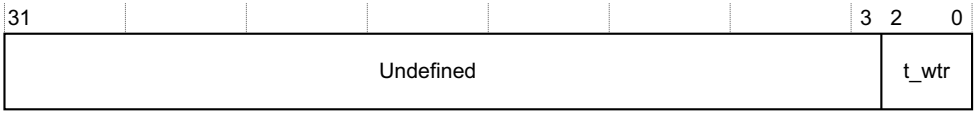


Figure 3-21 dmc_t_wtr Register bit assignments

Table 3-17 lists the register bit assignments.

Table 3-17 dmc_t_wtr Register bit assignments

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

3.3.17 DMC t_xp Register at 0x0040

The read/write dmc_t_xp Register sets exit power down command time in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-22 shows the register bit assignments.

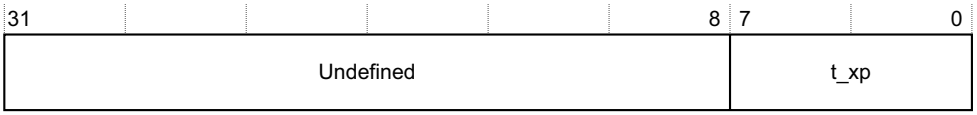


Figure 3-22 dmc_t_xp Register bit assignments

Table 3-18 lists the register bit assignments.

Table 3-18 dmc_t_xp Register bit assignments

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

3.3.18 DMC t_xsr Register at 0x0044

The read/write dmc_t_xsr Register sets exit self-refresh command time in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-23 shows the register bit assignments.

Figure 3-23 dmc_t_xsr Register bit assignments

Table 3-19 lists the register bit assignments.

Table 3-19 dmc_t_xsr Register bit assignments

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

3.3.19 DMC t_esr Register at 0x0048

The read/write dmc_t_esr Register sets self-refresh command time in memory clock cycles. It can only be read and written in the Config or Low-power state. Figure 3-24 shows the register bit assignments.

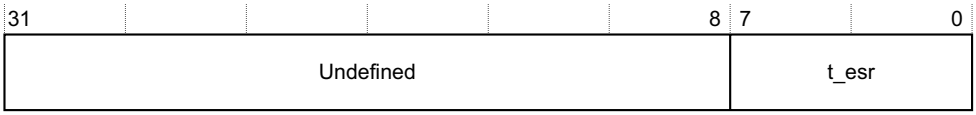


Figure 3-24 dmc_t_esr Register bit assignments

Table 3-20 lists the register bit assignments.

Table 3-20 dmc_t_esr Register bit assignments

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

3.3.20 DMC id_<0-5>_cfg Registers at 0x0100

The read/write dmc_id_<0-5>_cfg Registers are six registers that set the quality of service and span address locations `0x100-0x140`. The registers can only be read and written in the Config or Low-power states.

Figure 3-25 shows the register bit assignments.

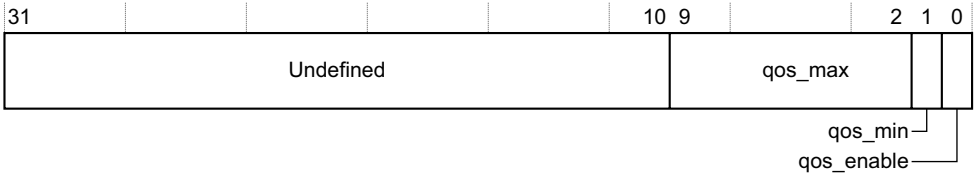


Figure 3-25 dmc_id_<0-5>_cfg Registers bit assignments

Table 3-21 lists the register bit assignments.

Table 3-21 dmc_id_<0-5>_cfg Registers bit assignments

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

3.3.21 DMC chip_<0-3>_cfg Registers at 0x0200

The read/write dmc_chip_<0-3>_cfg Registers set up the external memory device configuration. See the *Release Note* for the number of external chips supported, and therefore the number of these registers. They span address locations `0x200-0x300`. There is one register per memory device. The registers configure the base address and address decoding method. The registers can only be read and written in the Config or Low-power states.

Figure 3-26 shows the register bit assignments.

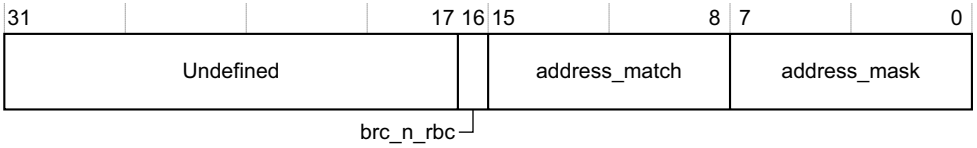


Figure 3-26 dmc_chip_<0-3>_cfg Registers bit assignments

Table 3-22 lists the register bit assignments.

Table 3-22 dmc_chip_<0-3>_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 AHB address: b0 = row, bank, column organization b1 = bank, row, column organization.
[15:8]	address_match	Comparison value for AHB address bits [31:24] to determine the chip that is selected.
[7:0]	address_mask	The mask for AHB address bits [31:24] to determine the chip that is selected: 1 = corresponding address bit is to be used for comparison.

3.3.22 DMC user_status Register at 0x0300

The dmc_user_status Register is a general purpose read-only register that returns the state on the **dmc_user_status[7:0]** primary inputs. The dmc_user_status Register can be read in all states. Figure 3-27 shows the register bit assignments.

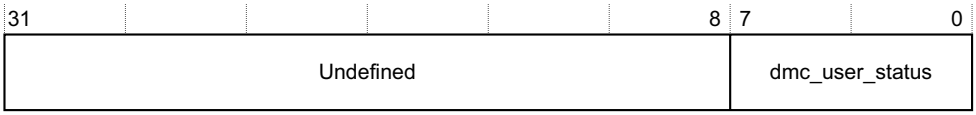


Figure 3-27 dmc_user_status Register bit assignments

Table 3-23 lists the register bit assignments.

Table 3-23 dmc_user_status Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	dmc_user_status	The value returns the state of the **dmc_user_status[7:0]** primary input pins

3.3.23 DMC user_config Register at 0x0304

The dmc_user_config Register is a general purpose write-only register. This register sets the value of the **dmc_user_config[7:0]** primary outputs. The dmc_user_config Register can be written in all states. Figure 3-28 shows the register bit assignments.

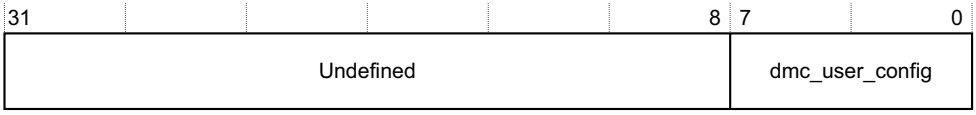


Figure 3-28 dmc_user_config Register bit assignments

Table 3-24 lists the register bit assignments.

Table 3-24 dmc_user_status Registers bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined. Write as zero.
[7:0]	dmc_user_config	This value sets the state of the **dmc_user_config[7:0]** primary output pins.

3.3.24 DMC Peripheral Identification Registers <0-3> at 0x0FE0-0x0FEC

The dmc_periph_id Registers are four 8-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. They are read by an external master to determine the device version of the DMC.

Table 3-25 lists the register bit assignments.

Table 3-25 dmc_periph_id Register bit assignments

Bits	Name	Description
[31:24]	integration_cfg	Configuration options are peripheral-specific. See the *DMC Peripheral Identification Register 3* on page 3-26.
[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 the DMC is `0x340`.

Figure 3-29 shows the correspondence between bits of the dmc_periph_id registers and the conceptual 32-bit Peripheral ID Register.

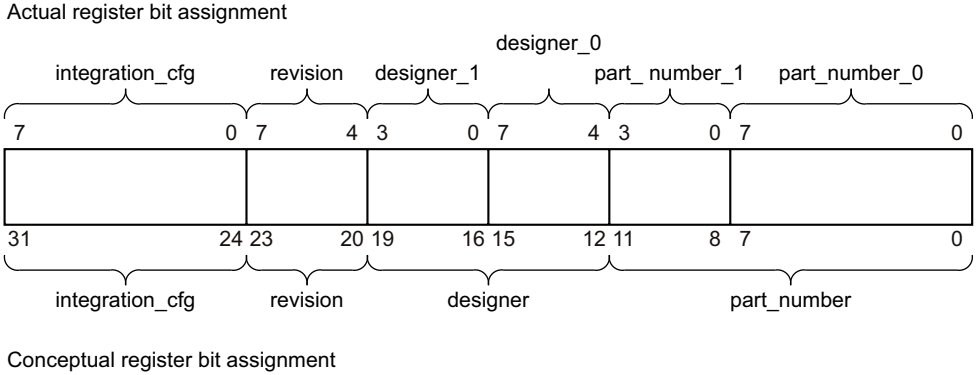


Figure 3-29 dmc_periph_id Register bit assignments

The following sections describe the dmc_periph_id Registers:

- *DMC Peripheral Identification Register 0* on page 3-25
- *DMC Peripheral Identification Register 1* on page 3-25
- *DMC Peripheral Identification Register 2* on page 3-25
- *DMC Peripheral Identification Register 3* on page 3-26.

DMC Peripheral Identification Register 0

The dmc_periph_id_0 Register is hard-coded and the fields within the register indicate the value. Table 3-26 lists the register bit assignments.

Table 3-26 dmc_periph_id_0 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:0]	part_number_0	These bits read back as `0x40`

DMC Peripheral Identification Register 1

The dmc_periph_id_1 Register is hard-coded and the fields within the register indicate the value. Table 3-27 lists the register bit assignments.

Table 3-27 dmc_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`

DMC Peripheral Identification Register 2

The dmc_periph_id_2 Register is hard-coded and the fields within the register indicate the value. Table 3-28 lists the register bit assignments.

Table 3-28 dmc_periph_id_2 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined.
[7:4]	revision	These bits read back as `0x1`
[3:0]	designer_1	These bits read back as `0x4`

DMC Peripheral Identification Register 3

The dmc_periph_id_3 Register is hard-coded and the fields within the register indicate the value of 0x0. Table 3-29 lists the register bit assignments.

Table 3-29 dmc_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.

3.3.25 DMC PrimeCell Identification Registers <0-3> at 0xFF0-0xFFC

The dmc_pcell_id Registers are four 8-bit wide registers, that span address locations 0xFF0-0xFFC. The registers can conceptually be treated as a single register that holds a 32-bit component ID value. The register can be used for automatic BIOS configuration. The pcell_id Register is set to 0xB105F00D. The register can be accessed with one wait state. Table 3-30 lists the register bit assignments.

Table 3-30 dmc_pcell_id Register bit assignments

Component ID register		CompID0-3 register		
Bits	**Value**	**Register**	**Bits**	**Description**
-	-	dmc_pcell_id_3	[31:8]	Read undefined
[31:24]	0xB1	dmc_pcell_id_3	[7:0]	These bits read back as 0xB1
-	-	dmc_pcell_id_2	[31:8]	Read undefined
[23:16]	0x05	dmc_pcell_id_2	[7:0]	These bits read back as 0x05
-	-	dmc_pcell_id_1	[31:8]	Read undefined
[15:8]	0xF0	dmc_pcell_id_1	[7:0]	These bits read back as 0xF0
-	-	dmc_pcell_id_0	[31:8]	Read undefined
[7:0]	0x0D	dmc_pcell_id_0	[7:0]	These bits read back as 0x0D

Figure 3-30 shows the register bit assignments.

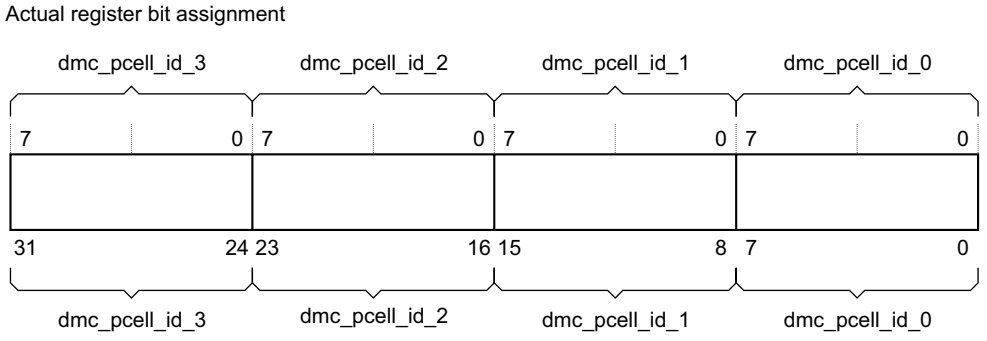


Figure 3-30 dmc_pcell_id Register bit assignments

The following sections describe the dmc_pcell_id Registers:

- *DMC PrimeCell Identification Register 0*
- *DMC PrimeCell Identification Register 1* on page 3-28
- *DMC PrimeCell Identification Register 2* on page 3-28
- *DMC PrimeCell Identification Register 3* on page 3-28.

Note

These registers cannot be read in the Reset state.

DMC PrimeCell Identification Register 0

The dmc_pcell_id_0 Register is hard-coded and the fields within the register indicate the value. Table 3-31 lists the register bit assignments.

Table 3-31 dmc_pcell_id_0 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	dmc_pcell_id_0	These bits read back as `0x0D`

DMC PrimeCell Identification Register 1

The dmc_pcell_id_1 Register is hard-coded and the fields within the register indicate the value. Table 3-32 lists the register bit assignments.

Table 3-32 dmc_pcell_id_1 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	dmc_pcell_id_1	These bits read back as `0xF0`

DMC PrimeCell Identification Register 2

The dmc_pcell_id_2 Register is hard-coded and the fields within the register indicate the value. Table 3-33 lists the register bit assignments.

Table 3-33 dmc_pcell_id_2 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	dmc_pcell_id_2	These bits read back as `0x5`

DMC PrimeCell Identification Register 3

The dmc_pcell_id_3 Register is hard-coded and the fields within the register indicate the value. Table 3-34 lists the register bit assignments.

Table 3-34 dmc_pcell_id_3 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	dmc_pcell_id_3	These bits read back as `0xB1`

3.4 SMC Register summary

Figure 3-31 shows the SMC configuration register map.

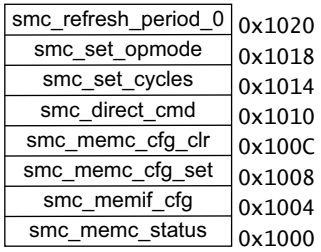


Figure 3-31 SMC configuration register map

Figure 3-32 shows the SMC chip<0-3> configuration register map:

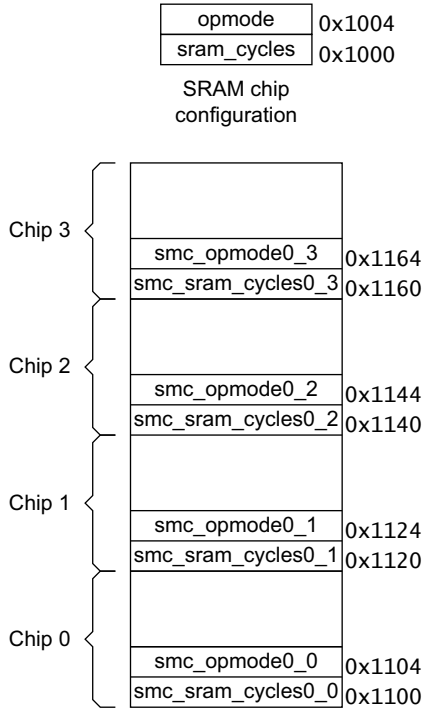


Figure 3-32 SMC chip configuration register map

Note

Figure 3-32 on page 3-29 shows the maximum number of supported chips. If you intend to use fewer, then the highest chip configuration blocks of the correct type are read back as zero.

Figure 3-33 shows the SMC user configuration memory register map.

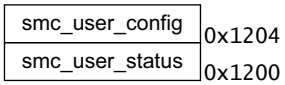


Figure 3-33 SMC user configuration register map

Figure 3-34 shows the SMC peripheral and PrimeCell identification configuration register map.

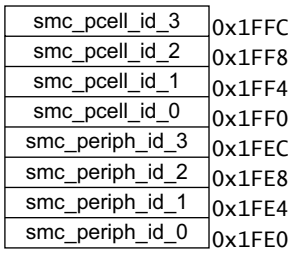


Figure 3-34 SMC peripheral and PrimeCell identification configuration register map

Table 3-35 lists the SMC Registers.

Table 3-35 Register summary

Name	Base offset	Type	Reset value	Description
smc_memc_status	0x1000	RO	0x00000000	See *SMC Memory Controller Status Register at 0x1000* on page 3-32.
smc_memif_cfg	0x1004	RO	0x0000002D	See *SMC Memory Interface Configuration Register at 0x1004* on page 3-33.
smc_memc_cfg_set	0x1008	WO	N/A	See *SMC Set Configuration Register at 0x1008* on page 3-34.
smc_memc_cfg_clr	0x100C	WO	N/A	See *SMC Clear Configuration Register at 0x100C* on page 3-35.

Table 3-35 Register summary (continued)

Name	Base offset	Type	Reset value	Description
smc_direct_cmd	0x1010	WO	N/A	See *SMC Direct Command Register at 0x1010* on page 3-36.
smc_set_cycles	0x1014	WO	N/A	See *SMC Set Cycles Register at 0x1014* on page 3-37.
smc_set_opmode	0x1018	WO	N/A	See *SMC Set Opmode Register at 0x1018* on page 3-38.
smc_refresh_period_0	0x1020	R/W	0x00000000	See *SMC Refresh Period 0 Register at 0x1020* on page 3-41.
smc_sram_cycles0_<0-3>	0x1000 + chip configuration base address	RO	0x0002B3CC	See *SMC SRAM Cycles Registers <0-3> at 0x1100, 0x1120, 0x1140, 0x1160* on page 3-41.
smc_opmode0_<0-3>	0x1004 + chip configuration base address	RO	0x00000802	See *SMC Opmode Registers <0-3> at 0x1104, 0x1124, 0x1144, 0x1164* on page 3-42.
smc_user_status	0x1200	RO	0x00000000	See *SMC User Status Register at 0x1200* on page 3-44.
smc_user_config	0x1204	WO	-	See *SMC User Configuration Register at 0x1204* on page 3-45.
smc_int_cfg	0x1E00	R/W	0x00000000	See *SMC Integration Configuration Register at 0x1E00* on page 4-6.
smc_int_inputs	0x1E04	RO	-	See *SMC Integration Inputs Register at 0x1E04* on page 4-7.
smc_int_outputs	0x1E08	WO	-	See *SMC Integration Outputs Register at 0x1E08* on page 4-8.
smc_periph_id_<0-3>	0x1FE0-0x1FEC	RO	See registers	See *SMC Peripheral Identification Registers <0-3> at 0x1FE0-0x1FEC* on page 3-45.
smc_pcell_id_<0-3>	0x1FF0-0x1FFC	RO	See registers	See *SMC PrimeCell Identification Registers <0-3> at 0x1FF0-0x1FFC* on page 3-48.

3.5 SMC Register descriptions

This section describes the SMC registers.

3.5.1 SMC Memory Controller Status Register at 0x1000

The read-only smc_memc_status Register provides information on the configuration of the SMC and also the current state of the SMC. This register cannot be read in the Reset state. Figure 3-35 shows the register bit assignments.

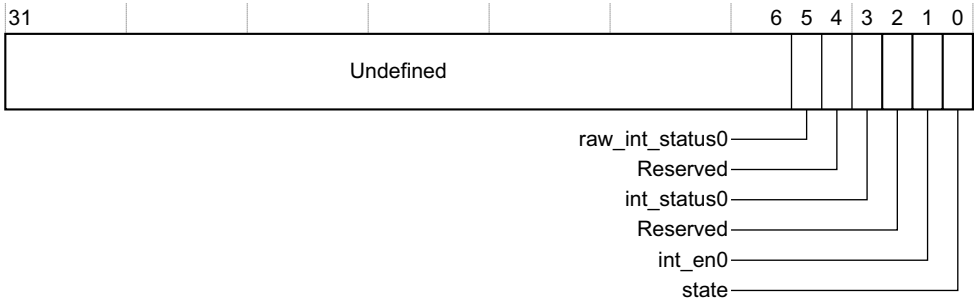


Figure 3-35 smc_memc_status Register bit assignments

Table 3-36 lists the register bit assignments.

Table 3-36 smc_memc_status Register bit assignments

Bits	Name	Function
[31:6]	-	Reserved, read undefined
[5]	raw_int_status0	Current raw interrupt status for interface 0
[4]	-	Reserved, read undefined
[3]	int_status0	Current interrupt status for interface 0
[2]	-	Reserved, read undefined
[1]	int_en0	Status of memory interface 0 interrupt enable
[0]	state	b0 indicates that the SMC is in Ready state b1 indicates that the SMC is in Low-power state.

3.5.2 SMC Memory Interface Configuration Register at 0x1004

The read-only smc_memif_cfg Register provides information on the configuration of the memory interface. This register cannot be read in the Reset state. Figure 3-36 shows the register bit assignments.

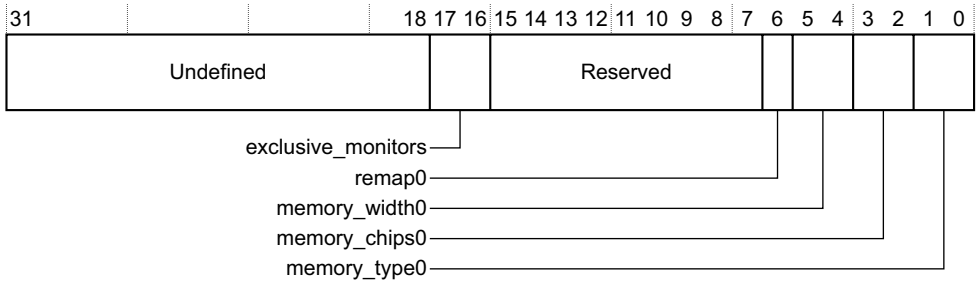


Figure 3-36 smc_memif_cfg Register bit assignments

Table 3-37 lists the register bit assignments.

Table 3-37 smc_memif_cfg Register bit assignments

Bits	Name	Function
[31:18]	-	Reserved, read undefined.
[17:16]	exclusive_monitors	Returns the number of exclusive access monitor resources that are implemented in the SMC: b00 = 0 monitors b01 = 1 monitors b10 = 2 monitors b11 = 4 monitors.
[15:7]	-	Reserved, read undefined.
[6]	remap0	Returns the value of the **smc_remap0** input.
[5:4]	memory_width0	Returns the maximum width of the SMC memory data bus for interface 0: b00 = 8 bits b01 = 16 bits b10 = 32 bits b11 = Reserved.

Table 3-37 smc_memif_cfg Register bit assignments (continued)

Bits	Name	Function
[3:2]	memory_chips0	Returns the number of different chip selects that the memory interface 0 supports: b00 = 1 chip b01 = 2 chips b10 = 3 chips b11 = 4 chips.
[1:0]	memory_type0	Returns the memory interface 0 type: b00 = reserved b01 = SRAM b10 = NAND b11 = reserved.

3.5.3 SMC Set Configuration Register at 0x1008

The write-only smc_memc_cfg_set Register enables the memory controller to be changed to Low-power state, and interrupts enabled. This register cannot be written to in the Reset state. Figure 3-37 shows the register bit assignments.

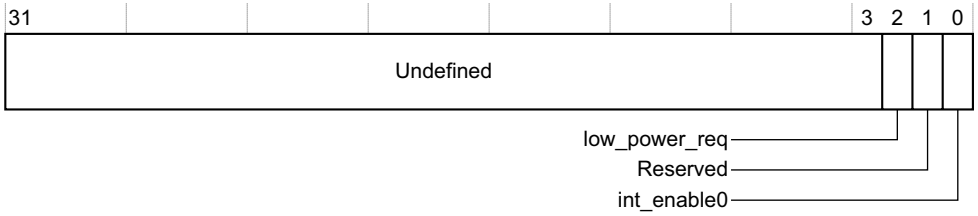


Figure 3-37 smc_memc_cfg_set Register bit assignments

Table 3-38 lists the register bit assignments.

Table 3-38 smc_memc_cfg_set Register bit assignments

Bits	Name	Function
[31:3]	-	Reserved, undefined. Write as zero.
[2]	low_power_req	b0 = no effect b1 = request the SMC to enter Low-power state when it next becomes idle.

Table 3-38 smc_memc_cfg_set Register bit assignments (continued)

Bits	Name	Function
[1]	-	Reserved, undefined. Write as zero.
[0]	int_enable0	b0 = no effect b1 = interrupt enable, memory interface 0.

3.5.4 SMC Clear Configuration Register at 0x100C

The write-only smc_memc_cfg_clr Register enables the memory controller to be moved out of the Low-power state, and the interrupts disabled. This register cannot be written to in the Reset state. Figure 3-38 shows the register bit assignments.

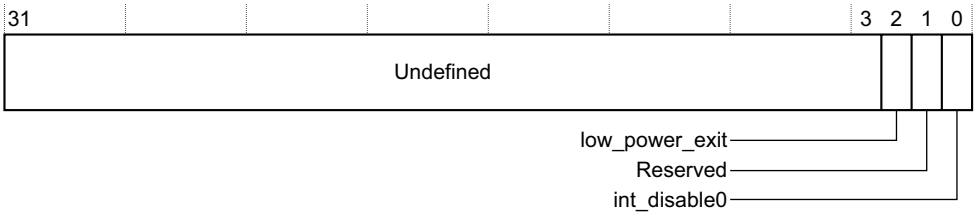


Figure 3-38 smc_memc_cfg_clr Register bit assignments

Table 3-39 lists the register bit assignments.

Table 3-39 smc_memc_cfg_clr Register bit assignments

Bits	Name	Function
[31:3]	-	Reserved, undefined. Write as zero.
[2]	low_power_exit	b0 = no effect b1= request the SMC to exit Low-power state.
[1]	-	Reserved, undefined, write as zero.
[0]	int_disable0	b0 = no effect b1= interrupt disable, memory interface 0.

3.5.5 SMC Direct Command Register at 0x1010

The write-only smc_direct_cmd Register passes commands to the external memory, and controls the updating of the chip configuration registers with values held in the set_opmode and set_cycles registers.

This register cannot be written to in either the Reset or Low-power states. Figure 3-39 shows the register bit assignments.

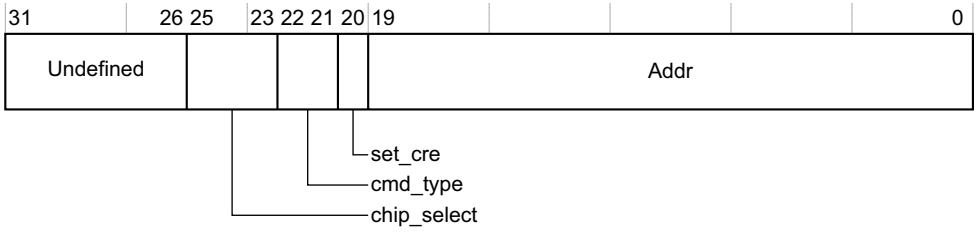


Figure 3-39 smc_direct_cmd Register bit assignments

Table 3-40 lists the register bit assignments.

Table 3-40 smc_directcmd Register bit assignments

Bits	Name	Function
[31:26]	-	Reserved, undefined, write as zero.
[25:23]	chip_select	Selects chip configuration register bank to update and enables chip mode register access depending on cmd_type. The encoding is: b000-b011 = chip selects 0-3 on interface 0 b100-b111 = reserved.
[22:21]	cmd_type	Determines the current command. The encoding is: b00 = UpdateRegs and AHB command b01 = ModeReg access b10 = UpdateRegs b11 = ModeReg and UpdateRegs.
[20]	set_cre	Maps to configuration register enable, **smc_cre**, output, when a ModeReg command is issued. The encoding is: b1 = **smc_cre** is HIGH when ModeReg write occurs b0 = **smc_cre** is LOW.
[19:0]	addr	Bits mapped to external memory address bits [19:0] when command is ModeReg access. Addr[15:0] matches **hwdata[15:0]** when the commands are UpdateRegs and AHB command access. Addr[19:16] are undefined.

3.5.6 SMC Set Cycles Register at 0x1014

This is the holding register for the smc_sram_cycles0_<0-3>. The write-only smc_set_cycles Register enables the time interval to be set for holding registers before being written to the memory manager specific registers. This register cannot be written to in either the Reset or Low-power states. Figure 3-40 shows the register bit assignments.

Note

Table 3-41 describes register holding, see *Memory manager operation* on page 2-54 for more information.

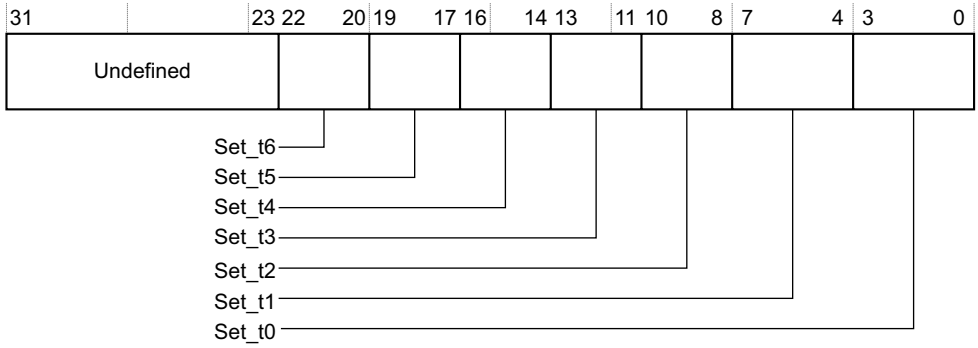


Figure 3-40 smc_set_cycles Register bit assignments

Table 3-41 lists the register bit assignments.

Table 3-41 smc_set_cycles Register bit assignments

Bits	Name	Function
[31:23]	-	Reserved, undefined. Write as zero.
[22:20]	Set_t6	Reserved.
[19:17]	Set_t5	Holding register for value to be written to the specific chip Register t_{TR} field.
[16:14]	Set_t4	Holding register for value to be written to the specific chip Register t_{PC} field.
[13:11]	Set_t3	Holding register for value to be written to the specific chip Register t_{WP} field.
[10:8]	Set_t2	Holding register for value to be written to the specific chip Register t_{CEOE} field.
[7:4]	Set_t1	Holding register for value to be written to the specific chip Register t_{WC} field.
[3:0]	Set_t0	Holding register for value to be written to the specific chip Register t_{RC} field.

3.5.7 SMC Set Opmode Register at 0x1018

This register is the holding register for the smc_opmode0_<0-3> working registers. The write-only smc_set_opmode Register cannot be written to in either the Reset or Low-power states. Figure 3-41 shows the register bit assignments.

Note

Table 3-42 on page 3-39 describes register holding, see *Memory manager operation* on page 2-54 for more information.

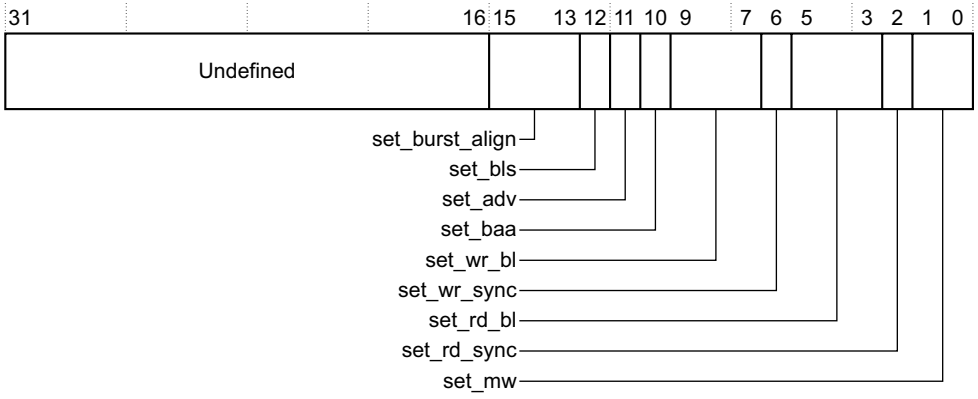


Figure 3-41 smc_set_opmode Register bit assignments

Table 3-42 lists the register bit assignments.

Table 3-42 smc_set_opmode Register bit assignments

Bits	Name	Function
[31:16]	-	Reserved, undefined, write as zero.
[15:13]	set_burst_align	Holding register for value to be written to the specific SRAM chip opmode Register burst_align field. These bits determine whether memory bursts are split on memory burst boundaries: 000 = bursts can cross any address boundary 001 = burst split on memory burst boundary, that is, 32 beats for continuous 010 = burst split on 64 beat boundary 011 = burst split on 128 beat boundary 100 = burst split on 256 beat boundary others = reserved. **Note** For asynchronous transfers: • the AHB MC always aligns read bursts to the memory burst boundary, when set_rd_sync = 0 • the AHB MC always aligns write bursts to the memory burst boundary, when set_wr_sync = 0.
[12]	set_bls	Holding register for value to be written to the specific SRAM chip smc_opmode Register *byte lane strobe* (bls) field. This bit affects the assertion of the byte-lane strobe outputs. b0 = bls timing equals chip select timing. This is the default setting. b1 = bls timing equals **smc_we_n_0** timing. This setting is used for eight bit wide memories that has no **smc_bls_n_0[3:0]** inputs. In this case, the smc_bls_n_0[3:0] output of the memory controller is connected to the **smc_we_n_0** memory input.
[11]	set_adv	Holding register for the value to be written to the specific SRAM chip smc_opmode Register *address valid* (adv) field. The memory uses the address advance signal **smc_adv_n_0** when set.
[10]	set_baa	Holding register for the value to be written to the specific SRAM chip smc_opmode Register *Burst Address Advance* (baa) field. The memory uses the **smc_baa_n_0** signal when set.

Table 3-42 smc_set_opmode Register bit assignments (continued)

Bits	Name	Function
[9:7]	set_wr_bl	Holding register for value to be written to the specific SRAM chip smc_opmode Register bls field. Encodes the memory burst length: b000 = 1 beat b001 = 4 beats b010 = 8 beats b011 = 16 beats b100 = 32 beats b101 = continuous b110-b111 = reserved.
[6]	set_wr_sync	Holding register for value to be written to the specific SRAM chip smc_opmode Register wr_sync field. The memory writes are synchronous when set. This bit is reserved for a NAND memory interface.
[5:3]	set_rd_bl	Holding register for value to be written to the specific SRAM chip smc_opmode Register bls field. Encodes the memory burst length: b000 = 1 beat b001 = 4 beats b010 = 8 beats b011 = 16 beats b100 = 32 beats b101 = continuous b110-b111 = reserved.
[2]	set_rd_sync	Holding register before being written to the specific SRAM chip smc_opmode Register rd_sync field. Memory in sync mode when set.
[1:0]	set_mw	Holding register for value to be written to the specific SRAM chip smc_opmode Register *memory width* (mw) field. Encodes the memory data bus width: b00 = 8 bits b01 = 16 bits b10 = 32 bits b11 = reserved You can program this to the configured width or half that width. See *SMC Memory Interface Configuration Register at 0x1004* on page 3-33.

3.5.8 SMC Refresh Period 0 Register at 0x1020

The read/write smc_refresh_period_0 Register enables the AHB MC to perform refresh cycles for PSRAM devices that you connect to memory interface 0. You cannot access this register in either the Reset or low-power states. Figure 3-42 shows the register bit assignments.

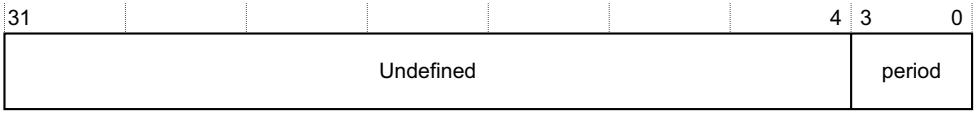


Figure 3-42 smc_refresh_period_0 Register bit assignments

Table 3-43 lists the register bit assignments.

Table 3-43 smc_refresh_period_0 Register bit assignments

Bits	Name	Function
[31:4]	-	Reserved, read undefined.
[3:0]	period	Sets the number of consecutive memory bursts that are permitted, prior to the AHB MC deasserting chip select to enable the PSRAM to initiate a refresh cycle. The options are: b0000 = disables the insertion of idle cycles between consecutive bursts b0001 = an idle cycle occurs after each burst b0010 = an idle cycle occurs after 2 consecutive bursts b0011 = an idle cycle occurs after 3 consecutive bursts b0100 = an idle cycle occurs after 4 consecutive bursts . . . b1111 = an idle cycle occurs after 15 consecutive bursts.

3.5.9 SMC SRAM Cycles Registers <0-3> at 0x1100, 0x1120, 0x1140, 0x1160

There is an instance of this register for each SRAM chip supported. The read-only smc_sram_cycles Register cannot be read in the Reset state. Figure 3-43 shows the register bit assignments.

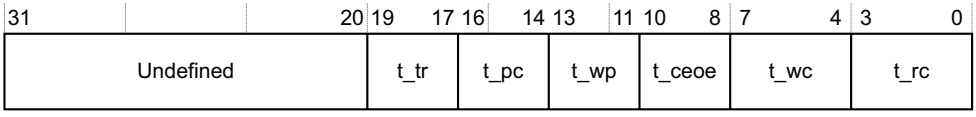


Figure 3-43 smc_sram_cycles Register bit assignments

Table 3-44 lists the register bit assignments.

Table 3-44 smc_sram_cycles Register bit assignments

Bits	Name	Function
[31:20]	-	Reserved, read undefined
[19:17]	t_tr	Turnaround time for SRAM chip configuration
[16:14]	t_pc	Page cycle time for SRAM chip configuration
[13:11]	t_wp	**smc_we_n_0** assertion delay
[10:8]	t_ceoe	**smc_oe_n_0** assertion delay for SRAM chip configuration
[7:4]	t_wc	Write cycle time
[3:0]	t_rc	Read cycle time

3.5.10 SMC Opmode Registers <0-3> at 0x1104, 0x1124, 0x1144, 0x1164

There is an instance of the smc_opmode Register for each chip supported. This register is read-only and cannot be read in the Reset state.

The reset values of these registers are configuration-dependent. You can set the memory width for the chip select 0 of each memory interface with a tie off to enable booting from that chip. The reset value of memory width for all other chip selects is the configured width. You must set all other bits to `0x0`, apart from address_match and address_mask. These are set by tie-offs at the top level.

Figure 3-44 shows the register bit assignments.

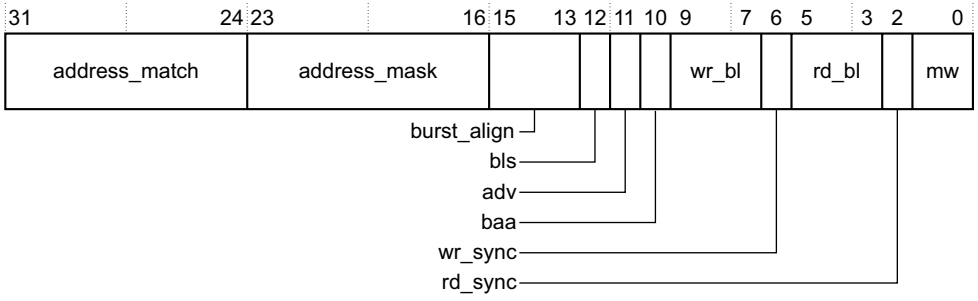


Figure 3-44 smc_opmode Register bit assignments

Table 3-45 lists the register bit assignments.

Table 3-45 smc_opmode Register bit assignments

Bits	Name	Function
[31:24]	address_match	Returns the value of this tie-off. This is the comparison value for address bits [31:24] to determine the chip that is selected.
[23:16]	address_mask	Returns the value of this tie-off. This is the mask for address bits[31:24] to determine the chip that must be selected. A logic 1 indicates the bit is used for comparison.
[15:13]	burst_align	These bits determine whether memory bursts are split on memory burst boundaries: 000 = bursts can cross any address boundary 001 = burst split on memory burst boundary, that is, 32 beats for continuous 010 = burst split on 64 beat boundary 011 = burst split on 128 beat boundary 100 = burst split on 256 beat boundary others = reserved. **Note** For asynchronous transfers: • the AHB MC always aligns read bursts to the memory burst boundary, when rd_sync = 0 • the AHB MC always aligns write bursts to the memory burst boundary, when wr_sync = 0.
[12]	bls	This bit affects the assertion of the byte-lane strobe outputs: b0 = bls timing equals chip select timing. This is the default setting. b1 = bls timing equals **smc_we_n_0** timing. This setting is used for 8-bit memories that has no Byte Lane Strobe inputs. In this case, the smc_bls_n_0[3:0] output of the memory controller is connected to the **smc_we_n_0** memory input.
[11]	adv	The memory uses the address advance signal **smc_adv_n_0**,when set.
[10]	baa	The memory uses the burst advance signal **smc_baa_n_0**, when set.
[9:7]	wr_bl	Determines the memory burst length for writes: b000 = 1 beat b001 = 4 beats b010 = 8 beats b011 = 16 beats b100 = 32 beats b101 = continuous b110-b111 = reserved.

Table 3-45 smc_opmode Register bit assignments (continued)

Bits	Name	Function
[6]	wr_sync	When set, the memory operates in write sync mode.
[5:3]	rd_bl	Determines the memory burst lengths for reads: b000 = 1 beat b001 = 4 beats b010 = 8 beats b011 = 16 beats b100 = 32 beats b101 = continuous b110-b111 = reserved.
[2]	rd_sync	When set, the memory operates in read sync mode.
[1:0]	mw	Determines the SMC memory data bus width: b00 = 8 bits b01 = 16 bits b10 = 32 bits b11 = reserved.

3.5.11 SMC User Status Register at 0x1200

The smc_user_status Register is a general purpose read-only register that returns the state on the **dmc_user_status[7:0]** primary inputs. The dmc_user_status Register can be read in all states. Figure 3-45 shows the register bit assignments.

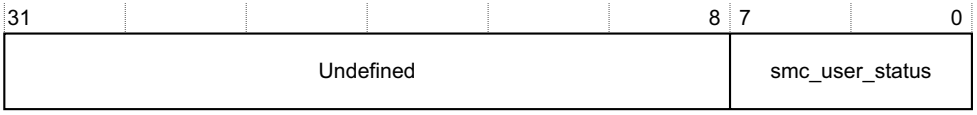


Figure 3-45 smc_user_status Register bit assignments

Table 3-46 lists the register bit assignments.

Table 3-46 smc_user_status Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	smc_user_status	The value returns the state of the **smc_user_status[7:0]** primary input pins

3.5.12 SMC User Configuration Register at 0x1204

The smc_user_config Register is a general purpose write-only register. This register sets the value of the **smc_user_config[7:0]** primary outputs. The smc_user_config Register can be written in all states. Figure 3-46 shows the register bit assignments.

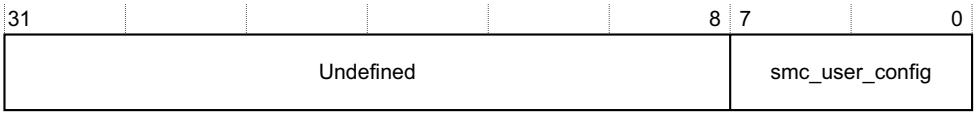


Figure 3-46 smc_user_config Register bit assignments

Table 3-47 lists the register bit assignments.

Table 3-47 smc_user_config Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, undefined. Write as zero.
[7:0]	smc_user_config	This value sets the state of the **dmc_user_config[7:0]** primary output pins.

3.5.13 SMC Peripheral Identification Registers <0-3> at 0x1FE0-0x1FEC

The smc_periph_id Registers are four 8-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. They are read by an external master to determine the SMC device version. None of these registers 0-3 can be read in the Reset state.

Table 3-48 lists the register bit assignments.

Table 3-48 smc_periph_id Register bit assignments

Bits	Name	Description
[31:25]	-	Reserved, read undefined.
[24]	integration_cfg	Configuration options are peripheral-specific. See *SMC Peripheral Identification Register 3* on page 3-47.
[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 the SMC is `0x352`.

Figure 3-47 shows the correspondence between bits of the smc_periph_id registers and the conceptual 32-bit Peripheral ID Register.

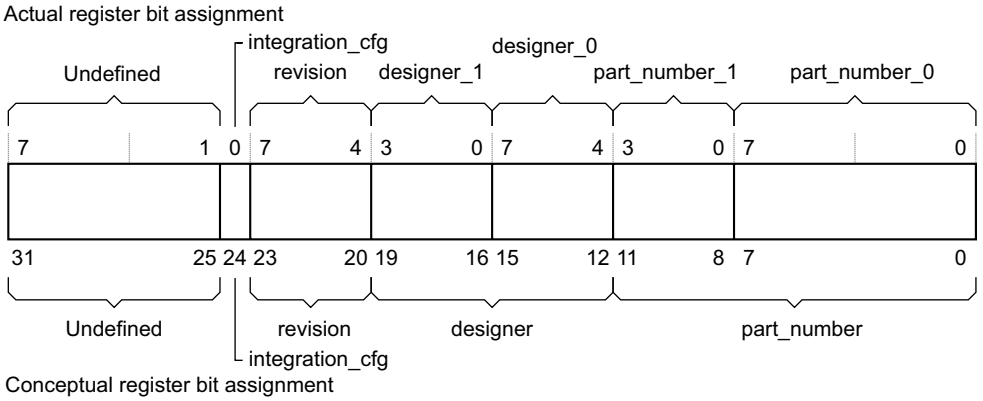


Figure 3-47 smc_periph_id Register bit assignments

The following sections describe the smc_periph_id Registers:

- *SMC Peripheral Identification Register 0*
- *SMC Peripheral Identification Register 1* on page 3-47
- *SMC Peripheral Identification Register 2* on page 3-47
- *SMC Peripheral Identification Register 3* on page 3-47.

SMC Peripheral Identification Register 0

The smc_periph_id_0 Register is hard-coded and the fields within the register indicate the value. Table 3-49 lists the register bit assignments.

Table 3-49 smc_periph_id_0 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	part_number_0	These bits read back as `0x52`

SMC Peripheral Identification Register 1

The smc_periph_id_1 Register is hard-coded and the fields within the register indicate the value. Table 3-50 lists the register bit assignments.

Table 3-50 smc_periph_id_1 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:4]	designer_0	These bits read back as 0x1
[3:0]	part_number_1	These bits read back as 0x3

SMC Peripheral Identification Register 2

The smc_periph_id_2 Register is hard-coded and the fields within the register indicate the value. Table 3-51 lists the register bit assignments.

Table 3-51 smc_periph_id_2 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:4]	revision	These bits read back as 0x3
[3:0]	designer_1	These bits read back as 0x4

SMC Peripheral Identification Register 3

The smc_periph_id_3 Register is hard-coded and the fields within the register indicate the value of 0x0. Table 3-52 lists the register bit assignments.

Table 3-52 smc_periph_id_3 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined.
[7:1]	-	Reserved for future use. Read undefined.
[0]	integration_cfg	When set, the integration test register map at address offset 0xE00 is present for reading and writing. If clear, the integration test registers have not been implemented.

3.5.14 SMC PrimeCell Identification Registers <0-3> at 0x1FF0-0x1FFC

The smc_pcell_id Registers are four 8-bit wide registers, that span address locations 0xFF0-0FFC. The registers can conceptually be treated as a single register that holds a 32-bit PrimeCell ID value. You can use the register for automatic BIOS configuration. The smc_pcell_id Register is set to 0xB105F00D. The register can be accessed with one wait state. Table 3-53 lists the register bit assignments.

Table 3-53 smc_pcell_id Register bit assignments

SMC pcell_id_0-3 register				
Bits	**Value**	**Register**	**Bits**	**Description**
-	-	smc_pcell_id_3	[31:8]	Read undefined
[31:24]	0xB1	smc_pcell_id_3	[7:0]	These bits read back as 0xB1
-	-	smc_pcell_id_2	[31:8]	Read undefined
[23:16]	0x05	smc_pcell_id_2	[7:0]	These bits read back as 0x05
-	-	smc_pcell_id_1	[31:8]	Read undefined
[15:8]	0xF0	smc_pcell_id_1	[7:0]	These bits read back as 0xF0
-	-	smc_pcell_id_0	[31:8]	Read undefined
[7:0]	0x0D	smc_pcell_id_0	[7:0]	These bits read back as 0x0D

Figure 3-48 shows the register bit assignments.

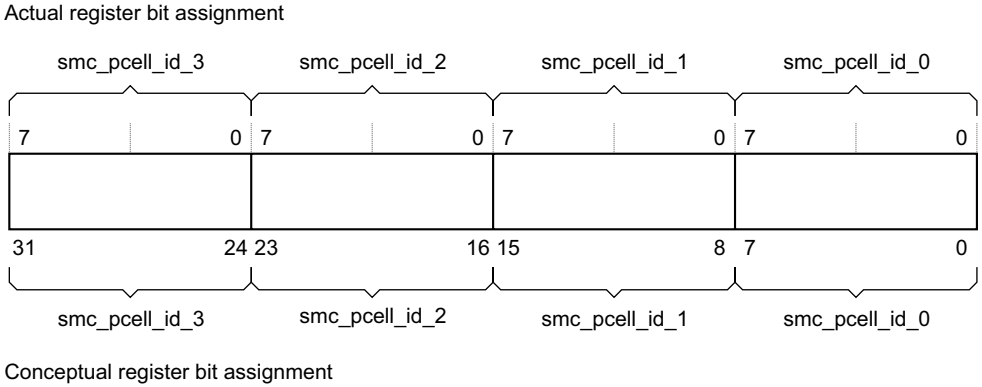


Figure 3-48 smc_pcell_id Register bit assignments

The following sections describe the smc_pcell_id Registers:

- *SMC Peripheral Identification Register 0* on page 3-46
- *SMC PrimeCell Identification Register 1*
- *SMC PrimeCell Identification Register 2* on page 3-50
- *SMC PrimeCell Identification Register 3* on page 3-50.

Note

These registers cannot be read in the Reset state.

SMC PrimeCell Identification Register 0

The smc_pcell_id_0 Register is hard-coded and the fields within the register indicate the value. Table 3-54 lists the register bit assignments.

Table 3-54 smc_pcell_id_0 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	smc_pcell_id_0	These bits read back as `0x0D`

SMC PrimeCell Identification Register 1

The smc_pcell_id_1 Register is hard-coded and the fields within the register indicate the value. Table 3-55 lists the register bit assignments.

Table 3-55 smc_pcell_id_1 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	smc_pcell_id_1	These bits read back as `0xF0`

SMC PrimeCell Identification Register 2

The smc_pcell_id_2 Register is hard-coded and the fields within the register indicate the value. Table 3-56 lists the register bit assignments.

Table 3-56 smc_pcell_id_2 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	smc_pcell_id_2	These bits read back as `0x5`

SMC PrimeCell Identification Register 3

The smc_pcell_id_3 Register is hard-coded and the fields within the register indicate the value. Table 3-57 lists the register bit assignments.

Table 3-57 smc_pcell_id_3 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	smc_pcell_id_3	These bits read back as `0xB1`

Chapter 4
Programmer's Model for Test

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

- *DMC and SMC integration test registers* on page 4-2.

4.1 DMC and SMC integration test registers

Figure 4-1 shows the DMC integration test register map.

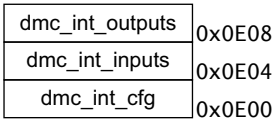


Figure 4-1 DMC integration test register map

Test registers are provided for integration testing. Table 4-1 lists the DMC integration test registers.

Table 4-1 DMC integration test register summary

Name	Base offset	Type	Reset value	Description
dmc_int_cfg	0x0E00	R/W	0x0	*DMC Integration Configuration Register at 0x0E00* on page 4-3
dmc_int_inputs	0x0E04	RO	Tie-off dependent	*DMC Integration Inputs Register at 0x0E04* on page 4-4
dmc_int_outputs	0x0E08	WO	-	*DMC Integration Outputs Register at 0x0E08* on page 4-6

Figure 4-2 shows the SMC integration Test Register map.

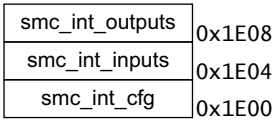


Figure 4-2 SMC integration test register map

Table 4-2 lists the SMC integration test registers.

Table 4-2 SMC integration test register summary

Name	Base offset	Type	Reset value	Description
smc_int_cfg	0x1E00	R/W	0x0	*SMC Integration Configuration Register at 0x1E00* on page 4-6
smc_int_inputs	0x1E04	RO	-	*SMC Integration Inputs Register at 0x1E04* on page 4-7
smc_int_outputs	0x1E08	WO	-	*SMC Integration Outputs Register at 0x1E08* on page 4-8

This section describes:

- *DMC Integration Configuration Register at 0x0E00*
- *DMC Integration Inputs Register at 0x0E04* on page 4-4
- *DMC Integration Outputs Register at 0x0E08* on page 4-6
- *SMC Integration Configuration Register at 0x1E00* on page 4-6
- *SMC Integration Inputs Register at 0x1E04* on page 4-7
- *SMC Integration Outputs Register at 0x1E08* on page 4-8.

4.1.1 DMC Integration Configuration Register at 0x0E00

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

Figure 4-3 shows the register bit assignments.

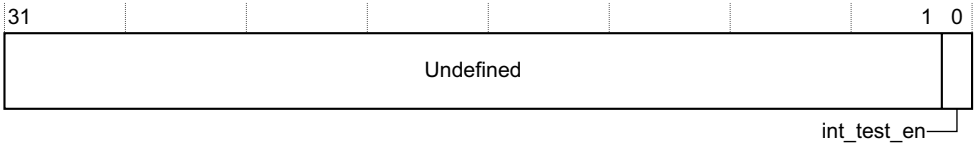


Figure 4-3 dmc_int_cfg Register bit assignments

Table 4-3 lists the register bit assignments.

Table 4-3 dmc_int_cfg Register bit assignments

Bits	Name	Function
[31:1]	Undefined	Reserved, 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 DMC Integration Inputs Register at 0x0E04

The read-only dmc_int_inputs Register enables an external master to access the inputs of the DMC using the APB interface. This register is only for test. It can only be read in the Config state.

Figure 4-4 shows the register bit assignments.

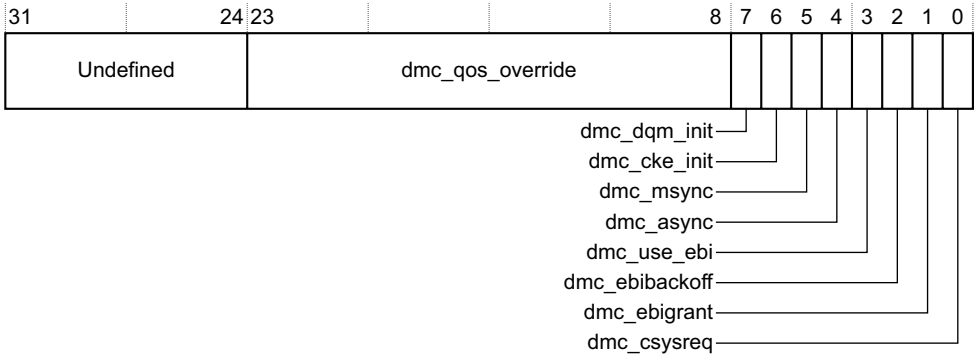


Figure 4-4 dmc_int_inputs Register bit assignments

Table 4-4 lists the register bit assignments.

Table 4-4 dmc_int_inputs Register bit assignments

Bits	Name	Function
[31:24]	-	Read undefined
[23:8]	dmc_qos_override	Returns the value of the **dmc_qos_override** external input
[7]	dmc_dqm_init	Returns the value of the **dmc_dqm_init** external input
[6]	dmc_cke_init	Returns the value of the **dmc_cke_init** external input

Table 4-4 dmc_int_inputs Register bit assignments (continued)

Bits	Name	Function
[5]	dmc_msync	Returns the value of the **dmc_msync** external input
[4]	dmc_async	Returns the value of the **dmc_async** external input
[3]	dmc_use_ebi	Returns the value of the **dmc_use_ebi** external input
[2]	dmc_ebibackoff	Returns the value of the **dmc_ebibackoff** external input
[1]	dmc_ebigrant	Returns the value of the **dmc_ebigrant** external input
[0]	dmc_csysreq	Returns the value of the **dmc_csysreq** external input

4.1.3 DMC Integration Outputs Register at 0x0E08

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

Figure 4-5 shows the register bit assignments.

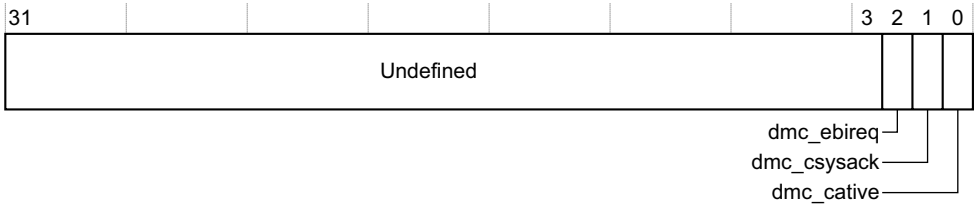


Figure 4-5 dmc_int_outputs Register bit assignments

Table 4-5 lists the register bit assignments.

Table 4-5 dmc_int_outputs Register bit assignments

Bits	Name	Function
[31:3]	-	Undefined. Write as zero.
[2]	dmc_ebireq	Drives the **dmc_ebireq** external output.
[1]	dmc_csysack	Drives the **dmc_csysack** external output.
[0]	dmc_cactive	Drives the **dmc_cactive** external output.

4.1.4 SMC Integration Configuration Register at 0x1E00

The read/write smc_int_cfg Register selects the integration test registers. This register is only for test. This register cannot be read or written to in the Reset state.

Figure 4-6 shows the register bit assignments.

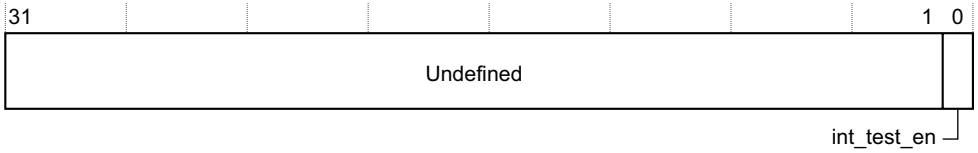


Figure 4-6 smc_int_cfg Register bit assignments

Table 4-6 lists the register bit assignments.

Table 4-6 smc_int_cfg Register bit assignments

Bits	Name	Function
[31:1]	Undefined	Reserved, read undefined. Write as zero.
[0]	int_test_en	When set, outputs are driven from the integration test registers and tied-off, and inputs can change for integration testing.

4.1.5 SMC Integration Inputs Register at 0x1E04

The read-only smc_int_inputs Register enables an external master to access the inputs of the SMC using the APB interface. This register is only for test. This register cannot be read in the Reset state.

Figure 4-7 shows the register bit assignments.

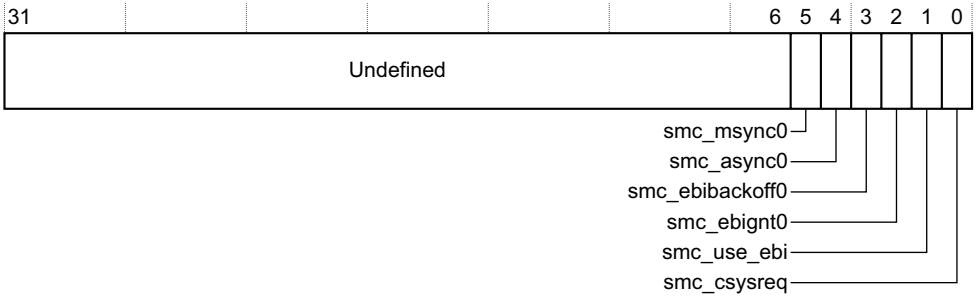


Figure 4-7 smc_int_inputs Register bit assignments

Table 4-7 lists the register bit assignments.

Table 4-7 smc_int_inputs Register bit assignments

Bits	Name	Function
[31:6]	-	Reserved, read undefined
[5]	smc_msync0	Returns the value of this top-level tie-off
[4]	smc_async0	Returns the value of this top-level tie-off
[3]	smc_ebibackoff0	Returns the value of the **smc_ebibackoff0** input

Table 4-7 smc_int_inputs Register bit assignments (continued)

Bits	Name	Function
[2]	smc_ebignt0	Returns the value of the **smc_ebigrant0** input
[1]	smc_use_ebi	Returns the value of the **smc_use_ebi** input
[0]	smc_csysreq	Returns the value of this external input

4.1.6 SMC Integration Outputs Register at 0x1E08

The write-only smc_int_outputs Register enables an external master to access the outputs of the SMC using the APB interface. This register cannot be read in the Reset state. Figure 4-8 shows the register bit assignments.

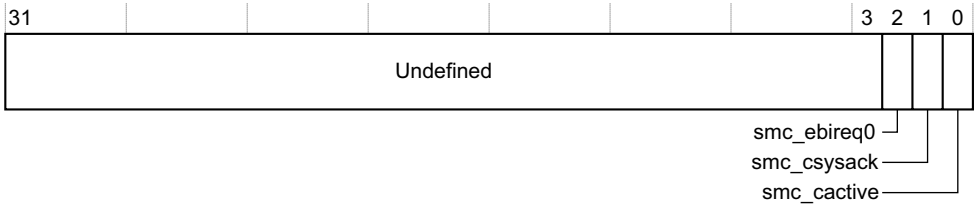


Figure 4-8 smc_int_outputs Register bit assignments

Table 4-8 lists the register bit assignments.

Table 4-8 smc_int_outputs Register bit assignments

Bits	Name	Function
[31:3]	-	Reserved, undefined. Write as zero.
[2]	smc_ebireq0	Sets the value of the **smc_ebireq0** output when in integration test mode.
[1]	smc_csysack	Sets the value of this external output when in integration test mode.
[0]	smc_cactive	This value is driven onto the external output when **int_test_en** is set HIGH.

Chapter 5
Device Driver Requirements

This chapter contains various flow diagrams to aid in the development of software drivers for the DMC and SMC. It contains the following sections:

5.1 DMC memory initialization

Figure 5-1 on page 5-3 and Figure 5-2 on page 5-4 show the sequence of events that a device driver must carry out to initialize the memory controller and a memory device to ensure the configuration of both is synchronized.

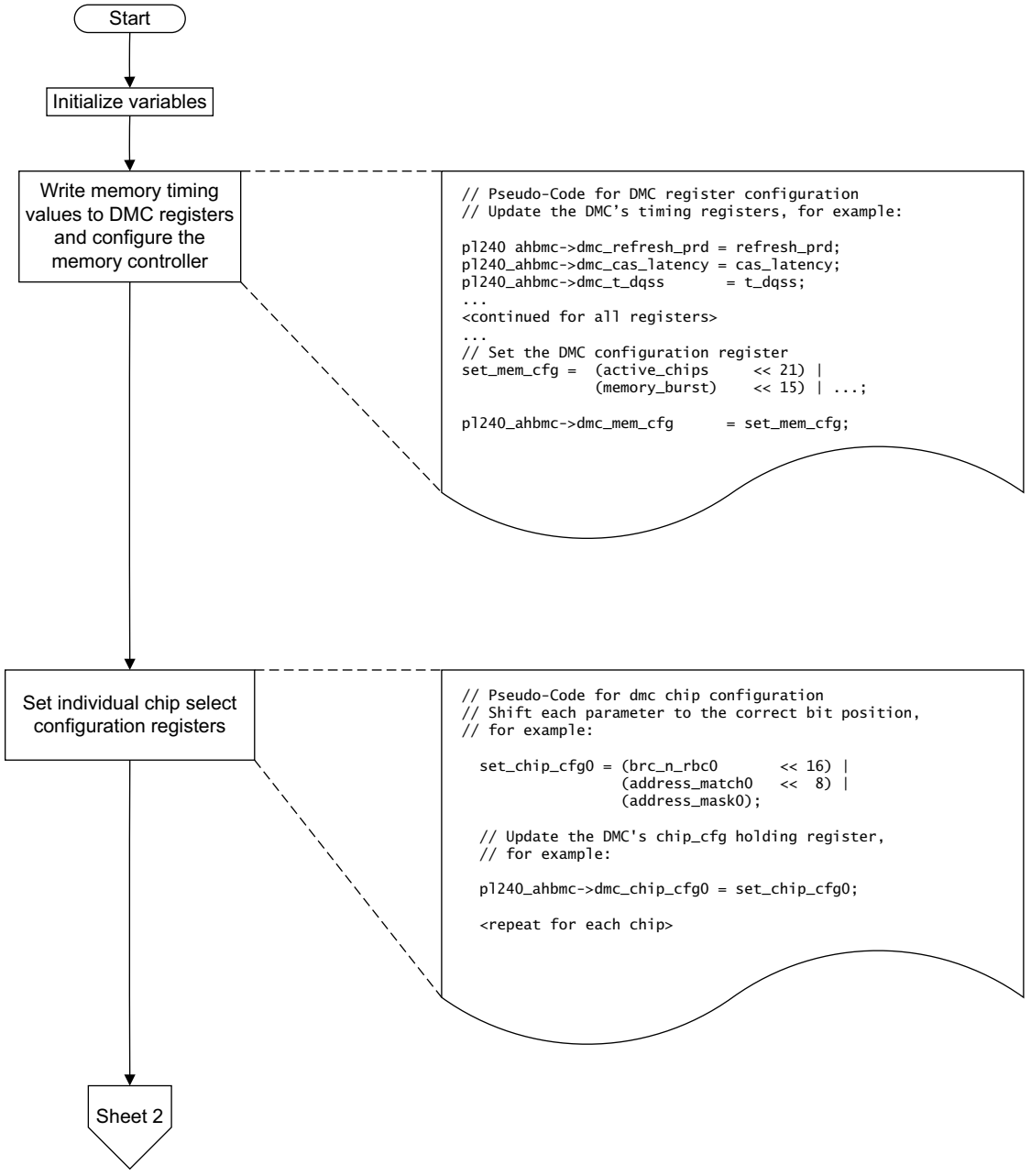


Figure 5-1 DMC and memory initialization sheet 1 of 2

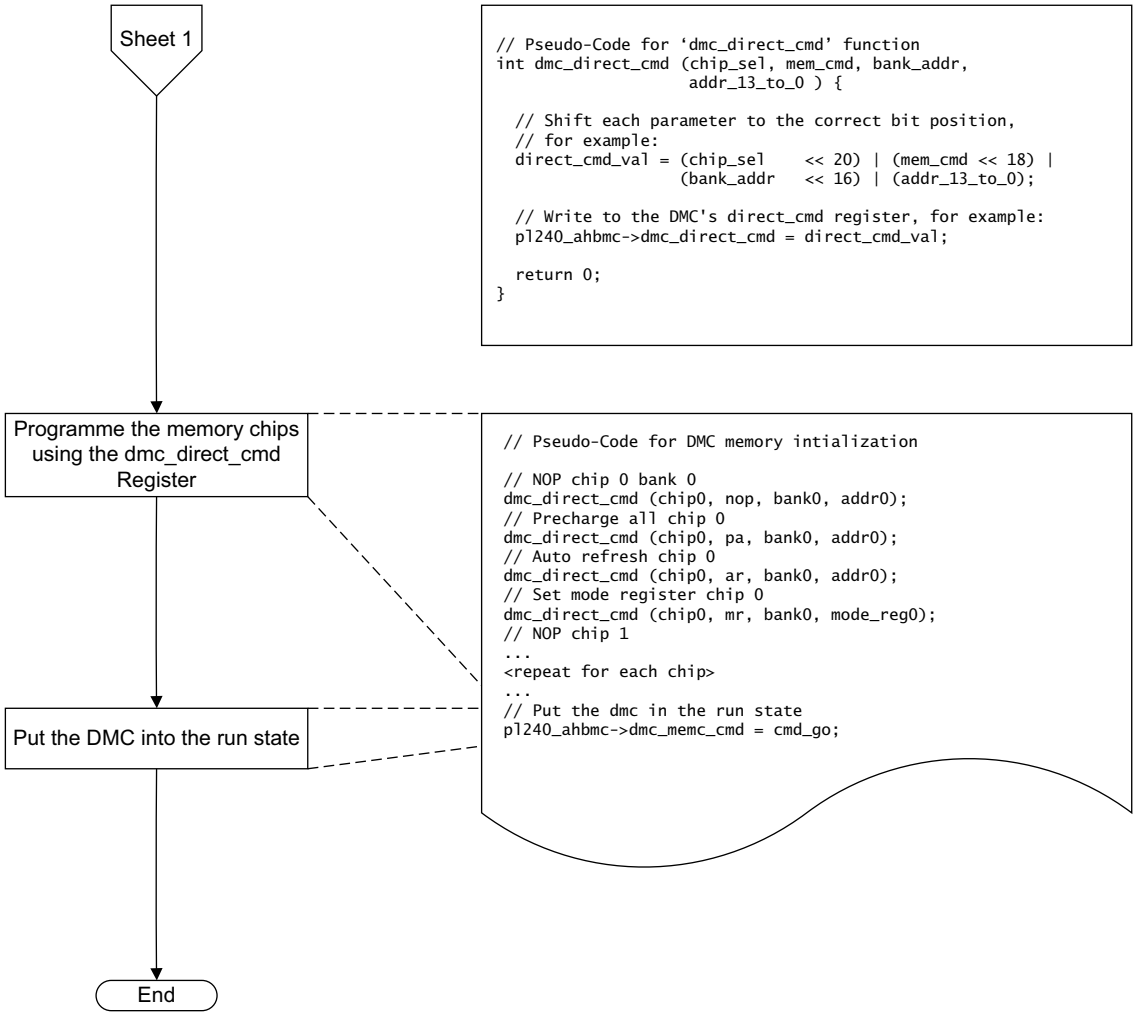


Figure 5-2 DMC and memory initialization sheet 2 of 2

5.2 SMC memory initialization

Figure 5-3 on page 5-6 to Figure 5-5 on page 5-8 show the sequence of events that a device driver must carry out to initialize the memory controller and a memory device to ensure the configuration of both is synchronized.

Typically, PSRAM devices can have the mode register programmed using the address bus only. NOR flash memory devices are examples of memory that require mode register accesses to be carried out using a sequence of accesses using the address and data buses. Check the data sheet for the specific memory device you are configuring to determine the configuration method.

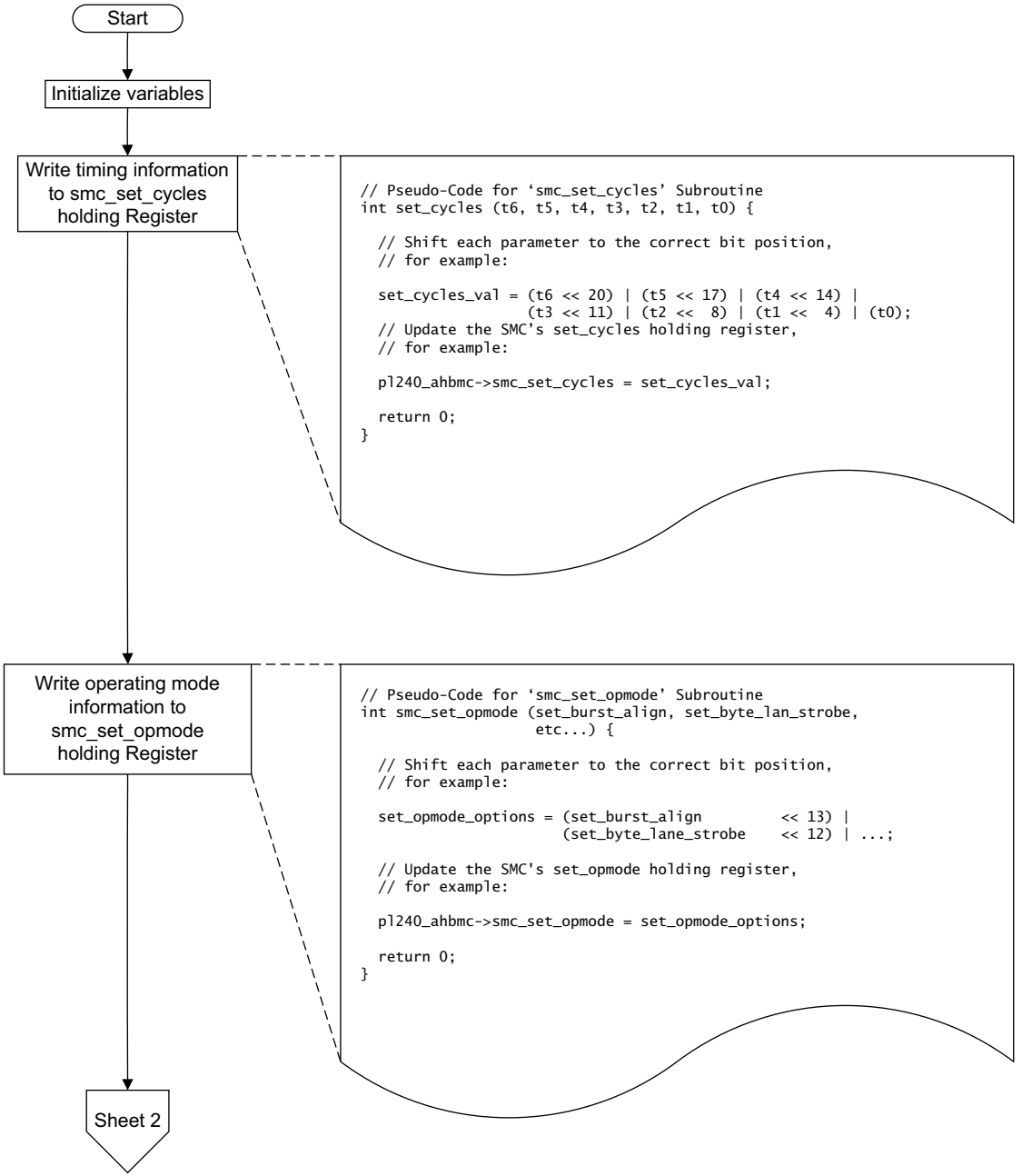


Figure 5-3 SMC and memory initialization sheet 1 of 3

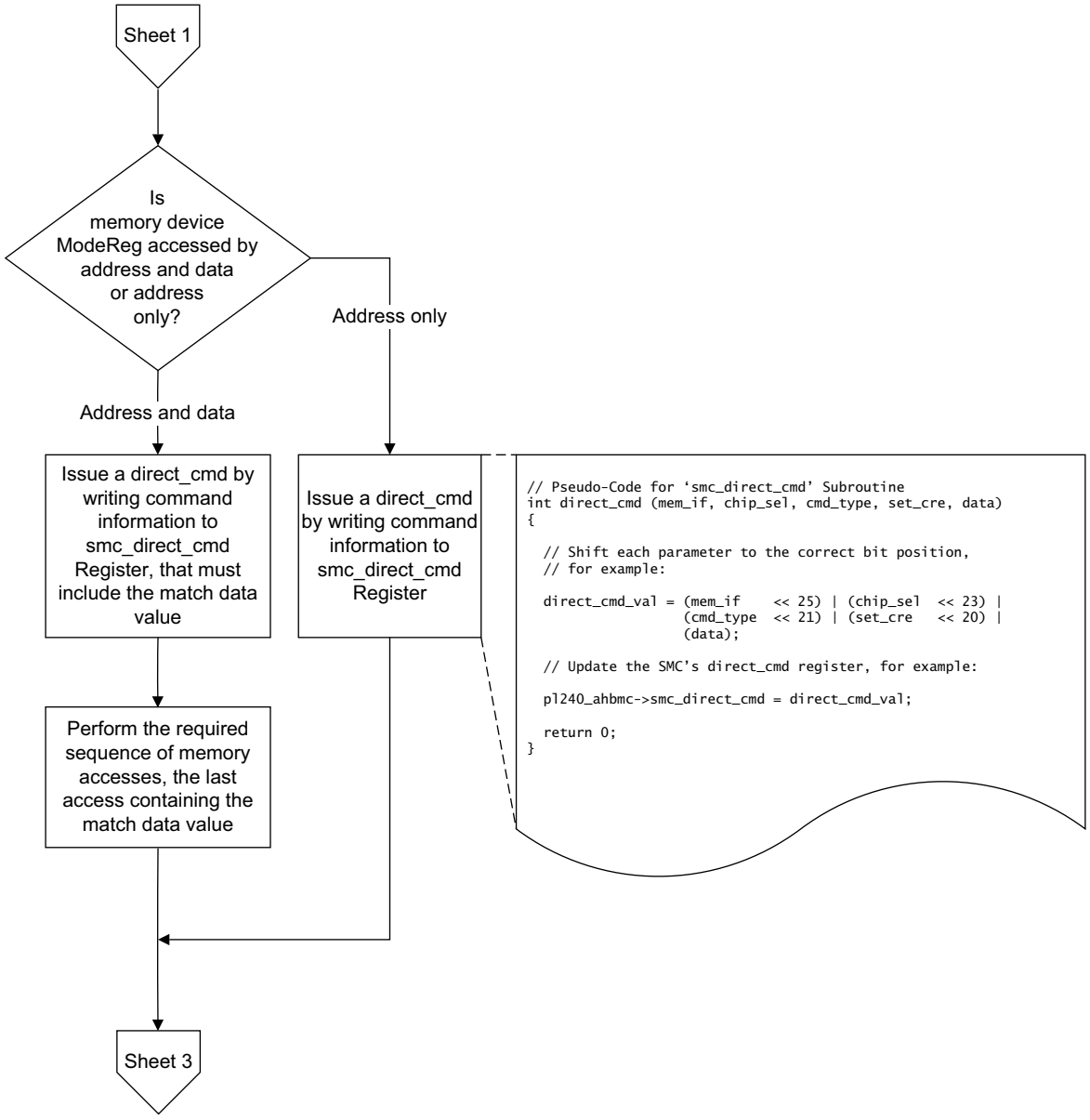


Figure 5-4 SMC and memory initialization sheet 2 of 3

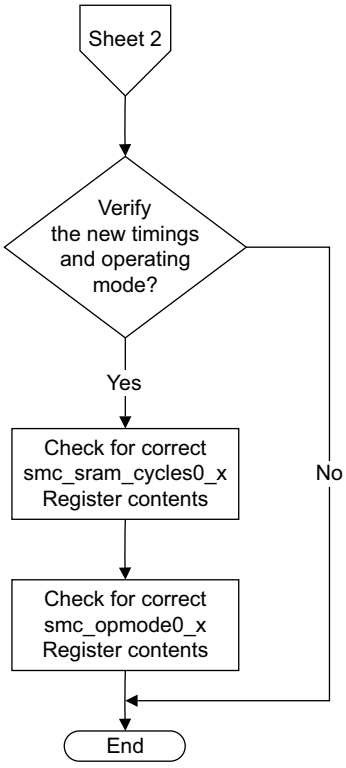


Figure 5-5 SMC and memory initialization sheet 3 of 3

Where:

x = denotes the appropriate chip select.

Appendix A
Signal Descriptions

This appendix lists and describes the processor signals. It contains the following sections:

A.1 About the signals list

This appendix lists the PL245 signals. Figure A-1 shows how the signals are grouped.

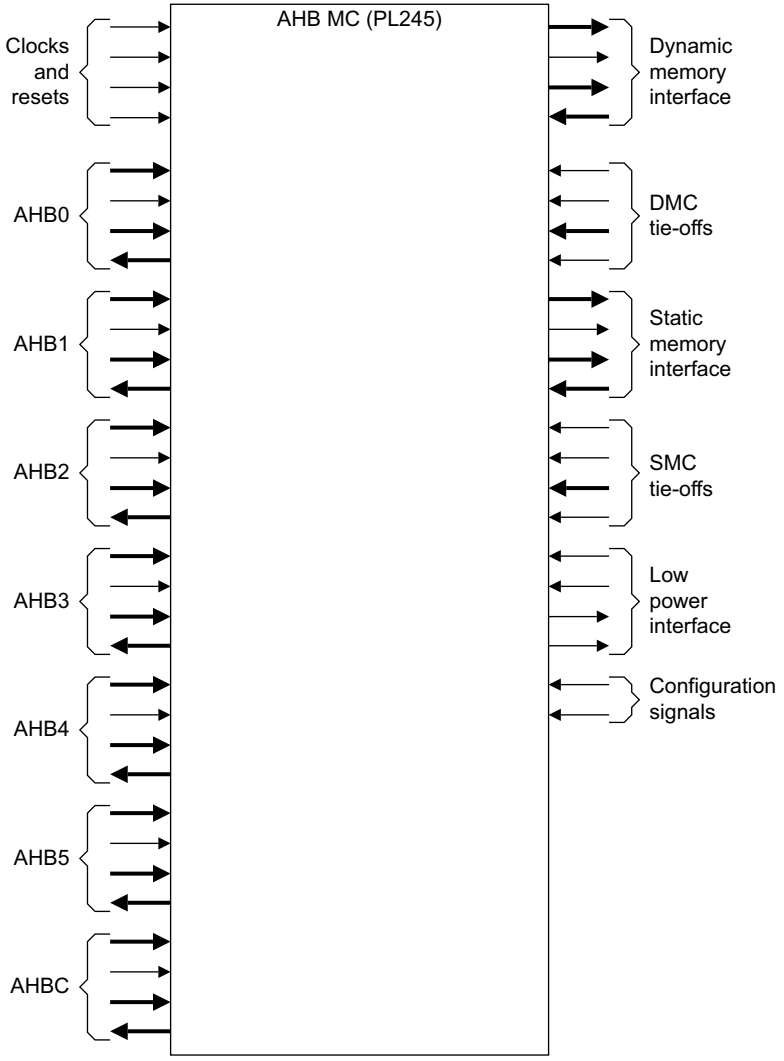


Figure A-1 AHB MC (PL245) grouping of signals

where:

AHBC = AHB Configuration signals

A.2 Clocks and resets

Table A-1 lists the clocks and resets signals.

Table A-1 Clocks and resets

Name	Type	Source/ destination	Description
hclk	Input	Clock source	AHB clock
hresetn	Input	Reset source	AHB clock domain reset
dmc_aclk	Input	Clock source	DMC AHB clock
dmc_mclk	Input	Clock source	DMC memory clock
dmc_mclkn	Input	Clock source	DMC inverted memory clock
dmc_mclkx2	Input	Clock source	DMC inverted memory clock
dmc_mclkx2n	Input	Clock source	DMC inverted memory clock
dmc_mresetn	Input	Reset source	DMC memory clock domain reset
smc_aclk	Input	Clock source	SMC AHB clock
smc_mclk0	Input	Clock source	SMC memory clock
smc_mclk0n	Input	Clock source	SMC inverted memory clock
smc_mreset0n	Input	Reset source	SMC memory clock domain reset

A.3 AHB signals

Table A-2 lists the AHB signals.

Table A-2 AHB signals

Name	Type	Source/ destination	Description
hsel<x>	Input	AHB	Transfer is to this port
haddr<x>[31:0]	Input	AHB	Address of transfer
htrans<x>[1:0]	Input	AHB	Transfer type
hwrite<x>	Input	AHB	Transfer is write or read
hsize<x>[2:0]	Input	AHB	Size of transfer
hburst<x>[2:0]	Input	AHB	Burst type of transfer
hprot<x>[3:0]	Input	AHB	Protection of transfer
hwdata<x>[31:0]	Input	AHB	Write data
hmastlock<x>	Input	AHB	Locked transfer
hready<x>	Input	AHB	System ready
hrdata<x>[31:0]	Output	AHB	Read data
hreadyout<x>	Output	AHB	Slave ready
hresp<x>[1:0]	Output	AHB	Slave response

where:

<x> = `0, 1, 2, 3, 4, 5 or c`, where `c` = configuration.

A.4 DMC memory interface signals

Table A-3 lists the DMC memory interface signals.

Table A-3 DMC memory interface signals

Name	Type	Source/ destination	Description
dmc_fbclk_in	Input	Memory	Fed back clock
dmc_dqs_in_<0-1>	Input	Memory	Data strobe in
dmc_dqs_in_n_<0-1>	Input	Memory	Inverted data strobe in
dmc_dq_in[15:0]	Input	Memory	Data in
dmc_clk_out[3:0]	Output	Memory	Clock out
dmc_ras_n	Output	Memory	Row address strobe
dmc_cas_n	Output	Memory	Column address strobe
dmc_we_n	Output	Memory	Write enable
dmc_cke	Output	Memory	Clock enable
dmc_cs_n[3:0]	Output	Memory	Chip select
dmc_add[15:0]	Output	Memory	Address
dmc_ba[1:0]	Output	Memory	Bank address
dmc_ap	Output	Memory	Auto precharge
dmc_dqs_out_<0-1>	Output	Memory	Data strobe out
dmc_dq_out[15:0]	Output	Memory	Data out
dmc_data_en	Output	Memory	Data enable
dmc_dqm[1:0]	Output	Memory	Data mask
dmc_use_ebi	Input	Memory	Use EBI tie-off
dmc_ebibackoff	Input	Memory	EBI back off
dmc_ebigrant	Input	Memory	EBI grant
dmc_ebireq	Output	Memory	EBI request

A.5 DMC miscellaneous signals

Table A-4 lists the DMC miscellaneous signals.

Table A-4 DMC miscellaneous signals

Name	Type	Source/ destination	Description
dmc_qos_override[15:0]	Input	System	QoS override
dmc_user_status[7:0]	Input	System	User signals to the configuration port
dmc_memory_width[1:0]	Input	System	Memory width tie-off
dmc_async	Input	System	AHB clock synchronous to memory clock
dmc_msync	Input	System	Memory clock synchronous to AHB clock
dmc_a_gt_m_sync	Input	System	AHB clock synchronous to and greater than memory clock
dmc_cke_init	Input	System	CKE polarity tie-off
dmc_dqm_init	Input	System	DQM polarity tie-off
dmc_user_config[7:0]	Output	System	User signals from the configuration port

A.6 SMC memory interface signals

Table A-5 lists the SMC memory interface signals.

Table A-5 SMC memory interface signals

Name	Type	Source/ destination	Description
smc_fbclk_in_0	Input	Memory	Fed back clock
smc_data_in_0[31:0]	Input	Memory	Data in
smc_wait_0	Input	Memory	Wait
smc_int_0	Input	Memory	Interrupt
smc_clk_out_0[3:0]	Output	Memory	Clock
smc_add_0[31:0]	Output	Memory	Address
smc_cs_n_0[3:0]	Output	Memory	Chip select
smc_we_n_0	Output	Memory	Write enable
smc_oe_n_0	Output	Memory	Output enable
smc_adv_n_0	Output	Memory	Address advance signal
smc_baa_n_0	Output	Memory	Bank address
smc_cre_0	Output	Memory	Configuration register enable
smc_bls_n_0[3:0]	Output	Memory	Byte lane strobes
smc_data_out_0[31:0]	Output	Memory	Data out
smc_data_en_0	Output	Memory	Data enable
smc_use_ebi	Input	Memory	Use EBI tie-off
smc_ebigrant0	Input	Memory	EBI grant
smc_ebibackoff0	Input	Memory	EBI back off
smc_ebireq0	Output	Memory	EBI request

A.7 SMC miscellaneous signals

Table A-6 lists the SMC miscellaneous signals.

Table A-6 SMC miscellaneous signals

Name	Type	Source/ destination	Description
smc_async0	Input	Tie-off	SMC clock synchronous to memory clock
smc_msync0	Input	Tie-off	Memory clock synchronous to SMC clock
smc_a_gt_m0_sync	Input	Tie-off	When HIGH, indicates that **smc_aclk** is greater than and synchronous to **smc_mclk0**
smc_address_mask0_0[7:0]	Input	Tie-off	Address mask for chip select 0
smc_address_match0_0[7:0]	Input	Tie-off	Address match for chip select 0
smc_address_mask0_[[7:0]	Input	Tie-off	Address mask for chip select 1
smc_address_match0_1[7:0]	Input	Tie-off	Address match for chip select 1
smc_address_mask0_2[7:0]	Input	Tie-off	Address mask for chip select 2
smc_address_match0_2[7:0]	Input	Tie-off	Address match for chip select 2
smc_address_mask0_3[7:0]	Input	Tie-off	Address mask for chip select 3
smc_address_match0_3[7:0]	Input	Tie-off	Address match for chip select 3
smc_remap_0	Input	Tie-off	Remap
smc_mux_mode_0	Input	Tie-off	Multiplexed address and data mode
smc_sram_mw_0[1:0]	Input	Tie-off	Memory width tie-off
smc_user_status[7:0]	Input	Tie-off	User signals to the configuration port
smc_user_config[7:0]	Output	System	User signals from the configuration port
smc_int	Output	System	Interrupt

A.8 Low-power interface

Table A-7 lists the low-power interface signals.

Table A-7 Low-power interface signals

Name	Type	Source/ destination	Description
ahb_csysreq	Input	System controller	AHB interfaces low-power mode request
dmc_csysreq	Input	System controller	DMC low-power mode request
smc_csysreq	Input	System controller	SMC low-power mode request
ahb_csysack	Output	System controller	AHB interfaces low-power mode acknowledge
ahb_cactive	Output	System controller	AHB interfaces active
dmc_csysack	Output	System controller	DMC low-power mode acknowledge
dmc_cactive	Output	System controller	DMC active
smc_csysack	Output	System controller	SMC low-power mode acknowledge
smc_cactive	Output	System controller	SMC active

A.9 Configuration signals

Table A-8 lists the configuration signals.

Table A-8 Configuration signals

Name	Type	Source/ destination	Description
big_endian	Input	Tie-off	Big-endian mode configuration tie-off
remap	Input	System	Remap signal

A.10 Scan chain signals

Table A-9 lists the scan chain signals.

Table A-9 Scan chain signals

Name	Type	Source/ destination	Description
se	Input	Scan chains	Scan enable for all clock domains
ahb_rst_bypass	Input	Scan chains	AHB reset bypass
dmc_rst_bypass	Input	Scan chains	DMC reset bypass
dmc_dft_en_clk_out	Input	Scan chains	DMC DFT enable for clock out
smc_rst_bypass	Input	Scan chains	SMC reset bypass
smc_dft_en_clk_out	Input	Scan chains	SMC DFT enable for clock out
si_hclk	Input	Scan chains	Scan in for **hclk** domain
dmc_si_aclk	Input	Scan chains	DMC scan in for **dmc_aclk** domain
dmc_si_mclk	Input	Scan chains	DMC scan in for **dmc_mclk** domain
dmc_si_mclkn	Input	Scan chains	DMC scan in for **dmc_mclkn** domain
dmc_si_mclkx2	Input	Scan chains	DMC scan in for **dmc_mclkx2** domain
dmc_si_mclkx2n	Input	Scan chains	DMC scan in for **dmc_mclkx2n** domain
dmc_si_fbclk_in	Input	Scan chains	DMC scan in for **dmc_fbclk_in** domain
dmc_si_dqs_in	Input	Scan chains	DMC scan in for **dmc_dqs_in**
dmc_si_neg_dqs	Input	Scan chains	DMC scan in for **dmc_dqs_in_n**
smc_si_aclk	Input	Scan chains	SMC scan in for **smc_aclk** domain
smc_si_mclk0	Input	Scan chains	SMC scan in for **smc_mclk0** domain
smc_si_mclk0n	Input	Scan chains	SMC scan in for **smc_mclk0n** domain
smc_si_fbclk_in_0	Input	Scan chains	SMC scan in for **smc_fbclk_in_0** domain
smc_si_int_0	Input	Scan chains	SMC scan in for **smc_int_0** domain
so_hclk	Output	Scan chains	Scan out for **hclk** domain
dmc_so_aclk	Output	Scan chains	DMC scan out for **dmc_aclk** domain

Table A-9 Scan chain signals (continued)

Name	Type	Source/ destination	Description
dmc_so_mclk	Output	Scan chains	DMC scan out for **dmc_mclk** domain
dmc_so_mclkn	Output	Scan chains	DMC scan out for **dmc_mclkn** domain
dmc_so_dqs_in	Output	Scan chains	DMC scan out for **dmc_dqs_in**
dmc_so_neg_dqs	Output	Scan chains	DMC scan out for **dmc_dqs_in_n**
dmc_so_fbclk_in	Output	Scan chains	DMC scan out for **dmc_fbclk_in** domain
smc_so_aclk	Output	Scan chains	SMC scan out for **smc_aclk** domain
smc_so_mclk0	Output	Scan chains	SMC scan out for **smc_mclk0** domain
smc_so_mclk0n	Output	Scan chains	SMC scan out for **smc_mclk0n** domain
smc_so_int_0	Output	Scan chains	SMC scan out for **smc_int_0** domain

Glossary

This glossary describes some of the terms used in technical documents from ARM Limited.

Advanced High-performance Bus (AHB)
A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and ARM Limited recommends only a subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)
A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. 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.

Advanced Peripheral Bus (APB)
: 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.

AHB
: *See* Advanced High-performance Bus.

Aligned
: A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.

AMBA
: *See* Advanced Microcontroller Bus Architecture.

APB
: *See* Advanced Peripheral Bus.

Beat
: Alternative word for an individual transfer within a burst. For example, an INCR4 burst comprises four beats.

  See also Burst.

BE-8
: Big-endian view of memory in a byte-invariant system.

  See also BE-32, LE, Byte-invariant and Word-invariant.

BE-32
: Big-endian view of memory in a word-invariant system.

  See also BE-8, LE, Byte-invariant and Word-invariant.

Big-endian
: Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory.

  See also Little-endian and Endianness.

Big-endian memory
: Memory in which:

  - a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address
  - a byte at a halfword-aligned address is the most significant byte within the halfword at that address.

  See also Little-endian memory.

Boundary scan chain

A boundary scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.

Burst

A group of transfers to consecutive addresses. Because the addresses are consecutive, there is no requirement to supply an address for any of the transfers after the first one. This increases the speed at which the group of transfers can occur. Bursts over AMBA are controlled using signals to indicate the length of the burst and how the addresses are incremented.

See also Beat.

Byte

An 8-bit data item.

Byte lane strobe

A signal that is used for unaligned or mixed-endian data accesses to determine the byte lanes that are active in a transfer. One bit of this signal corresponds to eight bits of the data bus.

Multi-master AHB

Typically a shared, not multi-layer, AHB interconnect scheme. More than one master connects to a single AMBA AHB link. In this case, the bus is implemented with a set of full AMBA AHB master interfaces. Masters that use the AMBA AHB-Lite protocol must connect through a wrapper to supply full AMBA AHB master signals to support multi-master operation.

Endianness

Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping.

See also Little-endian and Big-endian.

Little-endian

Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory.

See also Big-endian and Endianness.

Little-endian memory

Memory in which:

- a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address
- a byte at a halfword-aligned address is the least significant byte within the halfword at that address.

See also Big-endian memory.

SBO *See* Should Be One.

SBZ *See* Should Be Zero.

SBZP *See* Should Be Zero or Preserved.

Scan chain A scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.

Should Be One (SBO)
Should be written as 1 (or all 1s for bit fields) by software. Writing a 0 produces Unpredictable results.

Should Be Zero (SBZ)
Should be written as 0 (or all 0s for bit fields) by software. Writing a 1 produces Unpredictable results.

Should Be Zero or Preserved (SBZP)
Should be written as 0 (or all 0s for bit fields) by software, or preserved by writing the same value back that has been previously read from the same field on the same processor.

Unaligned A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.

Undefined Indicates an instruction that generates an Undefined instruction trap. See the *ARM Architecture Reference Manual* for more information on ARM exceptions.

UNP *See* Unpredictable.

Unpredictable Means that the behavior of the ETM cannot be relied on. Such conditions have not been validated. When applied to the programming of an event resource, only the output of that event resource is Unpredictable.

Unpredictable behavior can affect the behavior of the entire system, because the ETM is capable of causing the core to enter debug state, and external outputs can be used for other purposes.

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.

Remapping Changing the address of physical memory or devices after the application has started executing. This is typically done to permit RAM to replace ROM when the initialization has been completed.

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 0 and read as 0.