PrimeCell® Static Memory Controller (PL350 series)

Revision: r2p0

Technical Reference Manual

ARM®

ARM DDI 0380F

PrimeCell Static Memory Controller (PL350 series)
Technical Reference Manual

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
26 August 2005	A	Non-Confidential	First release
08 March 2006	B	Non-Confidential	Updated for r1p0, configurable IP
15 June 2006	C	Confidential	First release for r1p1
22 December 2006	D	Non-Confidential	First release for r1p2
25 May 2007	E	Non-Confidential	First release for r2p0
20 July 2007	F	Non-Confidential	Maintenance update for r2p0

Product Status

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

Web Address

`http://www.arm.com`

Contents

PrimeCell Static Memory Controller (PL350 series) Technical Reference Manual

List of Tables
PrimeCell Static Memory Controller (PL350 series) Technical Reference Manual

List of Figures
PrimeCell Static Memory Controller (PL350 series) Technical Reference Manual

Preface

This preface introduces the *PrimeCell Static Memory Controller (PL350 series) Technical Reference Manual* (TRM). It contains the following sections:

- *About this manual* on page xvi
- *Feedback* on page xx.

About this manual

This is the TRM for the *PrimeCell Static Memory Controller (PL350 series)*. The *Static Memory Controller* (SMC) product range comprises a number of configurable memory controllers that support SRAM and NAND on the memory interface.

Product revision status

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

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

Intended audience

This manual is written for implementation engineers and architects. It provides a description of an optimal SMC architecture. The SMC product range provides an interface between the *Advanced eXtensible Interface* (AXI) system bus and off-chip memory devices.

Using this manual

This manual is organized into the following chapters:

Chapter 1 Introduction

Read this chapter for an introduction to the SMC product range and its features.

Chapter 2 Functional Overview

Read this chapter for an overview of the major functional blocks and the operation of the SMC product range.

Chapter 3 Programmer's Model

Read this chapter for a description of the registers.

Chapter 4 Programmer's Model for Test

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

Chapter 5 Device Driver Requirements

Read this chapter for a description of device driver requirements.

Chapter 6 Configurations

Read this chapter for a description of non-universal SMC configurations.

Appendix A Signal Descriptions

Read this appendix for a description of the signals.

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

Conventions

Conventions that this manual can use are described in:

- *Typographical*
- *Timing diagrams* on page xviii
- *Signals* on page xviii
- *Numbering* on page xix.

Typographical

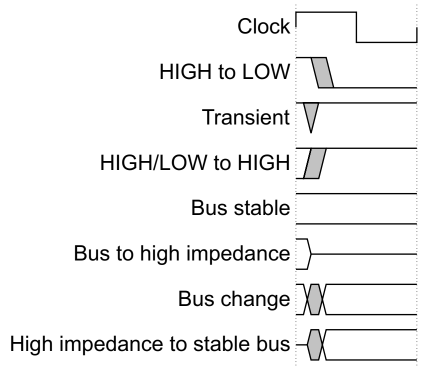
The typographical conventions are:

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.
`monospace` (mono underlined)	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 the register that is accessed.

Timing diagrams

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

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



Key to timing diagram conventions

Signals

The signal conventions are:

Signal level	The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.
Lower-case n	Denotes an active-LOW signal.
Prefix A	Denotes global 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 P	Denotes *Advanced Peripheral Bus* (APB) signals.
Prefix R	Denotes AXI read data channel signals.

Prefix W	Denotes AXI write data channel signals.

Numbering

The numbering convention is:

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

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

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

Further reading

This section lists publications by ARM, and by third parties.

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

ARM publications

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

- *PrimeCell Static Memory Controller (PL350 series) Integration Manual* (ARM DII 0137)
- *PrimeCell Static Memory Controller (PL350 series) Implementation Guide* (ARM DII 0138)
- *AMBA Designer (FD001) User Guide* (ARM DUI 0334)
- *AMBA® AXI Protocol v1.0 Specification* (ARM IHI 0022)
- *AMBA 3 APB Protocol v1.0 Specification* (ARM IHI 0024)
- *ARM PrimeCell External Bus Interface (PL220) Technical Reference Manual* (ARM DDI 0249).

Feedback

ARM welcomes feedback on the SMC 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) that your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the SMC (PL350 series). It contains the following sections:

- *About the SMC (PL350 series)* on page 1-2
- *Supported devices* on page 1-7
- *Product revisions* on page 1-8.

1.1 About the SMC (PL350 series)

The SMC is an *Advanced Microcontroller Bus Architecture* (AMBA) compliant *System-on-Chip* (SoC) peripheral. The product range consists of high-performance, area-optimized SRAM and NAND memory controllers with on-chip bus interfaces that conform to the AMBA *Advanced eXtensible Interface* (AXI) protocol.

The product range consists of a number of controllers that support either one or two memory interfaces of type NAND or SRAM. The controllers are:

SMC (PL351)	Single NAND interface.
SMC (PL352)	Single SRAM interface.
SMC (PL353)	SRAM and NAND interfaces.
SMC (PL354)	Dual SRAM interfaces.

The NAND memory interface type is defined as supporting NAND flash with multiplexed *Address/Data* (A/D) buses.

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.

You can configure aspects of the memory controller series to provide the optimum features, performance, and gate count required for your intended application. For a summary of the configurable features supported, see *Features of the SMC (PL350 series)* on page 1-4.

Figure 1-1 on page 1-3 shows the interfaces of the SMC (PL350 series) product range.

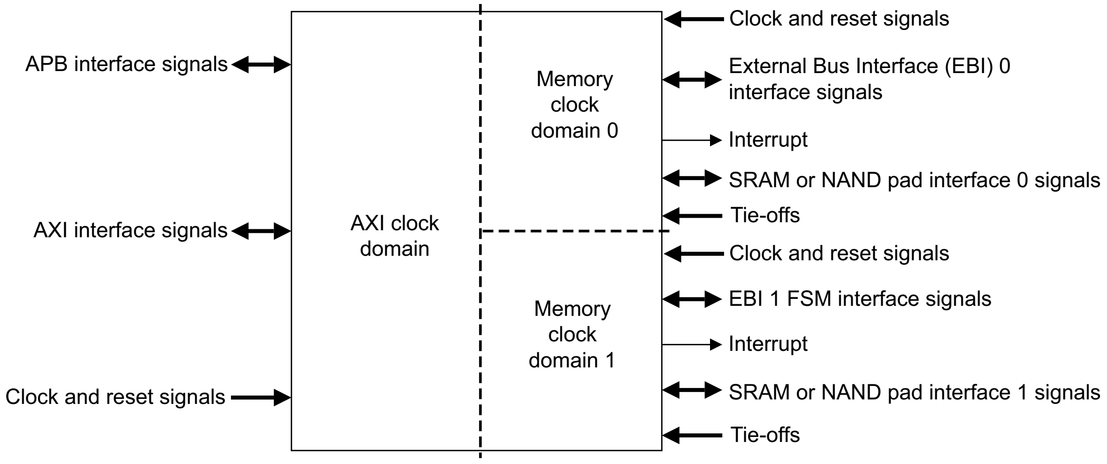


Figure 1-1 SMC interfaces

Figure 1-2 shows an example system containing the SMC (PL351) variant.

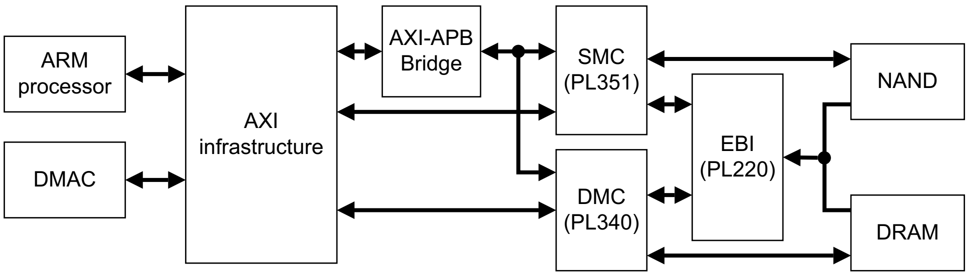


Figure 1-2 SMC (PL351) example system

The system contains:

- two bus masters:
  - — an ARM processor
  - — a *DMA controller* (DMAC).
- AXI infrastructure component
- two memory controllers:
  - — SMC (PL351)
  - — a *Dynamic Memory Controller* (DMC).
- an *External Bus Interface* (EBI).

The AXI interconnect enables each bus master to access both bus slaves. To reduce pin count, the EBI multiplexes the address and data pins of the SMC and DMC memory interfaces.

Figure 1-3 shows an example system containing the SMC (PL353) variant.

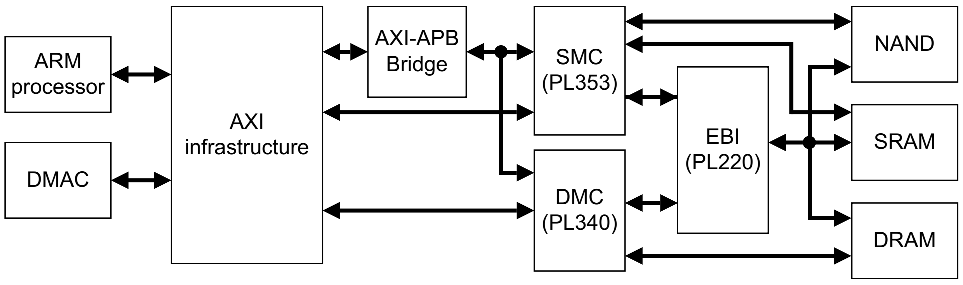


Figure 1-3 SMC (PL353) example system

This configuration of the SMC has two memory interfaces:

- one is connected to NAND flash
- one is connected to SRAM memory.

The EBI enables the address and data pins of three memory interfaces to be multiplexed. In this case, the three memory interfaces are:

- the NAND interface of the SMC (PL353)
- the SRAM interface of the SMC (PL353)
- the dynamic memory interface of the DMC.

1.1.1 Features of the SMC (PL350 series)

The SMC provides the following features:

- You can configure the SMC to support the following options:
  - Maximum NAND memory data widths of 8-bit or 16-bit.
  - Maximum SRAM memory data widths of 8-bit, 16-bit, or 32-bit.
  - AXI data width of 32-bit or 64-bit.
  - Up to four chip selects per memory interface.
  - Configurable command, read data, and write data FIFO depths.
  - An additional pipeline stage within the format logic enables higher AXI clock frequencies at the cost of an additional clock cycle of latency.
  - Configurable number of outstanding exclusive accesses supported.

— Optional 2-bit detection, 1-bit correction *Error Correction Code* (ECC) block for SLC NAND memories.

Table 1-1 and Table 1-2 list the supported memory widths and AXI data widths. You can program the memory width for each chip select, with the maximum available being the configured width, and the minimum being one quarter of the AXI data width.

Table 1-1 SRAM memory widths and AXI port widths

Memory width	AXI port width	
	32-bit	64-bit
8-bit	Y	N
16-bit	Y	Y
32-bit	Y	Y

Table 1-2 NAND memory widths and AXI port widths

Memory width	AXI port width	
	32-bit	64-bit
8-bit	Y	Y
16-bit	Y	Y

- Support for *ARM Architecture Version 6* (ARMv6) exclusive access transfers to SRAM.
- Programmable interrupt generation to indicate NAND flash status.
- Programmable cycle timings, and memory width per chip select.
- Programmable address cycles and command values for NAND flash accesses enabling operation with a variety of NAND devices.
- Atomic switching of memory device and controller operating modes.
- Support for the *PrimeCell EBI (PL220)* that enables sharing of external address and data bus pins between memory controller interfaces.
- Support for AXI low-power interface.
- Support for a **remap** signal for each interface.

- Support for multiple clock domains and configurable to be synchronous or asynchronous.

1.1.2 AXI interface attributes

The slave interface has the following attributes, that are fixed for a particular configuration of the SMC:

Write acceptance capability

The maximum number of active write transactions that a slave can accept.

Read acceptance capability

The maximum number of active read transactions that a slave can accept.

Combined acceptance capability

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

Write interleave depth

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

Read data reordering depth

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

Table 1-3 lists the attribute formats.

Table 1-3 Attribute formats

Attribute	Value
Combined acceptance capability	2 for SMC (PL351) and SMC (PL352) 3 for SMC (PL353) and SMC (PL354)
Write interleave depth	1
Read data reorder depth	Read acceptance capability

1.2 Supported devices

About the SMC (PL350 series) on page 1-2 describes the memory device types that each SMC configuration supports. The SMC is designed to support every static memory type. The *Release Note* provides a specific list of memory devices tested with each configuration.

Some memory devices, or series of memory devices, have specific limitations:

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.

The SMC (PL350 series) only does WE# controlled writes. This is to simplify the design by having fewer timing registers and simpler timing controls.

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

1.3 Product revisions

This section describes differences in functionality between product revisions:

r1p0-r1p1 Contains the following differences in functionality:

- periph_id_2 Register bits [7:4] now read back as `0x2`. See *Peripheral Identification Register 2* on page 3-35.

r1p1-r1p2 Contains the following differences in functionality:

- periph_id_2 Register bits [7:4] now read back as `0x3`. See *Peripheral Identification Register 2* on page 3-35.
- Addition of the refresh_period_0 and refresh_period_1 Registers. These enable the controller to insert an idle cycle during consecutive bursts, that enables a PSRAM device to initiate a refresh cycle.

r1p2 - r2p0 Contains the following differences in functionality:

- periph_id _2 Register bits [7:4] now read back as 0x4. See *Peripheral Identification Register 2* on page 3-35.
- Addition of Optional SLC ECC support for NAND interfaces.

Note

See the engineering errata that accompanies the product deliverables, for information about any functional differences that the new revision provides.

Chapter 2
Functional Overview

This chapter describes the SMC operation. It contains the following sections:

2.1 Functional description

Figure 2-1 on page 2-3 shows an SMC block diagram.

Note

Depending on the configuration, you can implement either one or two memory interfaces and associated clock domains.

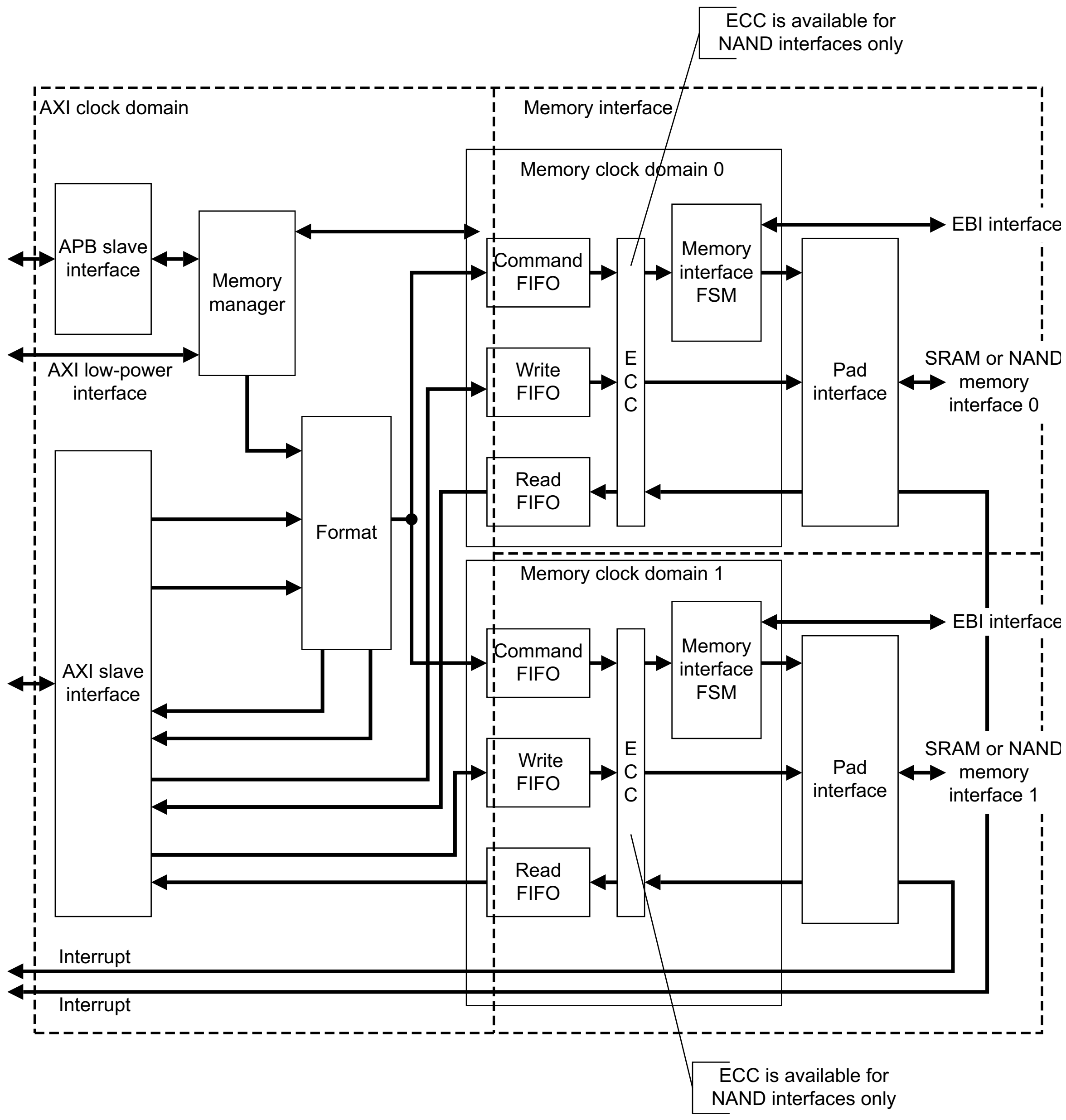


Figure 2-1 SMC block diagram

The main blocks of the SMC are:

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

2.1.1 AXI slave interface

The AXI slave interface comprises five AXI channels:

- Read-Address (AR)
- Write-Address (AW)
- Write (W)
- Read (R)
- Write response (B).

For information on the AXI protocol see the *AMBA AXI Protocol v1.0 Specification*.

Figure 2-2 on page 2-5 shows the AXI slave interface external signals.

Note

In Figure 2-2 on page 2-5:

- The **arcache**, **awcache**, **arprot** and **awprot** signals are shown for completeness only. The controller does not use the information transferred by these signals.
- The clock and reset signals are not shown, see Table A-1 on page A-2.
- See Table A-10 on page A-9 for information about **wdata[PORTWIDTH-1:0]** and **wstrb[PORTBYTES-1:0]**.
- See Table A-13 on page A-12 for information about **rdata[PORTWIDTH-1:0]**.

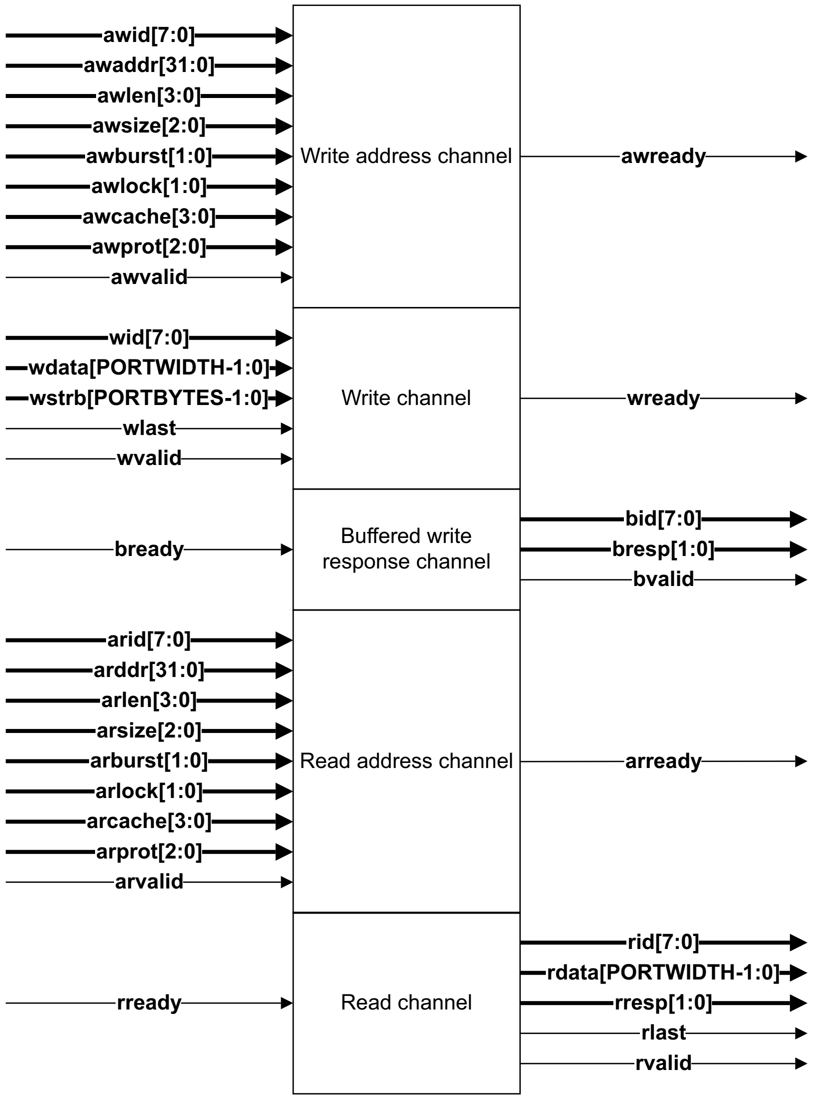


Figure 2-2 AXI slave interface signals

2.1.2 APB slave interface

The APB interface is a fully-compliant APB slave. The SMC has 4KB of memory allocated to it. For information describing the APB interface see the *AMBA 3 APB Protocol v1.0 Specification*.

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* for more information.

Figure 2-3 shows the APB external signals. Clock and reset signals are omitted.

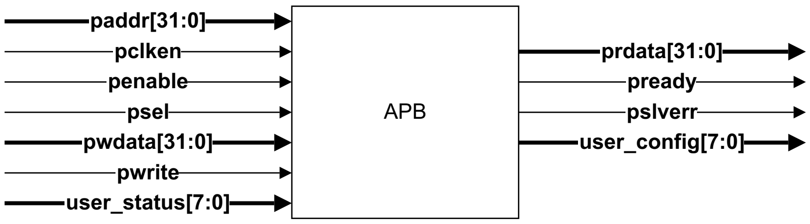


Figure 2-3 APB external signals

Note

- **aclk** is not shown
- **user_status[7:0]** and **user_config[7:0]** are general purpose APB accessible input and output signals respectively.

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

2.1.3 Format

The format block receives memory accesses from the AXI slave interface and the memory manager. Requests from AR and AW channels are arbitrated on a round-robin basis. Requests from the manager have the highest priority. The format block also maps AXI transfers onto appropriate memory transfers and passes these to the memory interface through the command FIFO.

See *Format block* on page 2-14 for more information.

2.1.4 Memory manager

The memory manager tracks and controls the current state of the SMC **aclk** domain FSM. The block is responsible for:

- Updating register values that are used in the `mclk<x>` domain, and controlling direct commands issued to memory.
- Controlling entry-to and exit-from low-power mode through the APB interface.
- The AXI low-power interface. See *AXI low-power operation* on page 2-26.

See *Memory manager operation* on page 2-26 for more information.

2.1.5 Memory interface

The SMC supports two memory interface types, SRAM and NAND. Both SRAM and NAND memory interfaces are composed of command, read data, and write data FIFOs, plus a control FSM. The memory interface FSM is specific to either SRAM or NAND. 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. NAND interfaces also have an optional SLC ECC block.

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

2.1.6 Pad interface

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

Figure 2-4 shows the SRAM interface, where x is the memory interface. 0 to 1, and n is the chip select, 0 to 3.

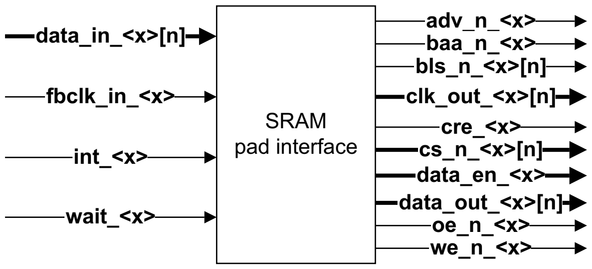


Figure 2-4 SRAM interface

Note

Figure 2-4 on page 2-7 does not show the clock and reset signals.

Figure 2-5 shows the NAND interface, where x is the memory interface, 0 to 1, and n is the chip select, 0 to 3.

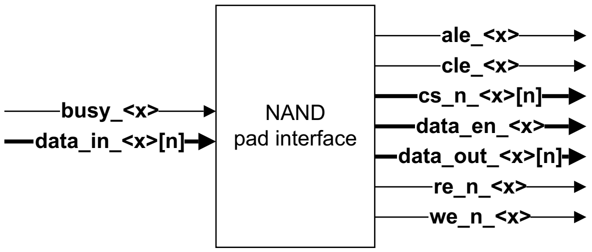


Figure 2-5 NAND interface

Interrupts

The SMC series provides interrupt outputs for use with NAND flash, because of the long wait times associated with this memory. Both the SRAM and NAND memory interface types support interrupts, and an interrupt is provided for each memory interface. The interrupt is triggered on the rising edge of:

- The **int** input for the SRAM memory interface type
- The **busy** input for the NAND memory interface type.

An additional interrupt is provided if an ECC block is included.

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

2.2 Functional operation

This section describes:

- *Operating states*
- *Clocking and resets* on page 2-10
- *Miscellaneous signals* on page 2-12
- *APB slave interface operation* on page 2-13
- *Format block* on page 2-14
- *Memory manager operation* on page 2-26
- *Interrupts operation* on page 2-31
- *Memory interface operation* on page 2-31
- *Error Correction Code operation* on page 2-48.

2.2.1 Operating states

The operation of the SMC is based on three operating states. This section describes each of the states. Figure 2-6 shows the state machine.

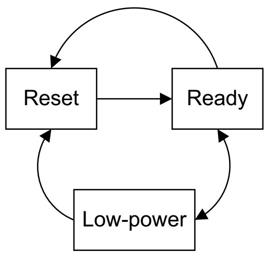


Figure 2-6 aclk domain FSM

The SMC states are as follows:

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

Ready Normal operation of the device. The APB interface can access the SMC register bank and the AXI interface can access external memory devices.

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

The state transitions are:

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

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

Ready to Low-power

The controller must enter the idle state before it can enter the low-power state. The controller enters low-power state when:

- The low_power_req bit is set in the APB memc_cfg_set Register. See *Set Configuration Register* on page 3-10.
- The AXI master asserts **csysreq**. See *AXI low-power interface signals* on page A-13.

Note

After the controller receives a low-power request, it does not respond to commands on the APB interface, until it enters the low-power state.

Low-power to Ready

The device exits the low-power state back to Ready when either:

- The low_power_exit bit is set in the APB memc_cfg_clr Register. *Clear Configuration Register* on page 3-11.
- The AXI master deasserts **csysreq**. See *AXI low-power interface signals* on page A-13.

Low-power to Reset

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

2.2.2 Clocking and resets

This section describes:

- *Clocking*
- *Reset* on page 2-12.

Clocking

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

- **aclk**
- **mclk0**
- **mclk0n**.

The SMC (PL353) and SMC (PL354) configurations support two memory interfaces and therefore implement an additional clock domain and the following associated inputs:

- **mclk1**
- **mclk1n**.

These clocks can be grouped into three clock domains:

AXI domain **aclk** is in this domain. You can only stop **aclk** when the SMC is in low-power mode.

Memory clock domains

All the clocks except **aclk** are in these domains. Ensure that the **mclk<x>** signal is clocked at the rate of the external memory clock speed. **mclk<x>n** is an inverted version of **mclk<x>**. In configurations implementing two memory interfaces, the two memory clock domains can additionally be asynchronous to each other. You can stop the clocks for the memory clock domain when the SMC is in low-power mode.

Note

The **<x>** notation represents memory interface 0 or 1.

See the *PrimeCell Static Memory Controller (PL350 series) Integration Manual* for the required relationships between the clocks.

You can tie-off the SMC **async<x>** and **msync<x>** pins so that the **aclk** and **mclk<x>** clock domains can operate synchronously or asynchronously with respect to each other. When you use the EBI (PL220), you must operate the SMC (PL353) and SMC (PL354) **mclk0** and **mclk1** clock domains synchronously at 1:1, n:1, or 1:n.

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 **aclk** and **mclk<x>** domains enables synchronous operation of the two clocks at multiples of each other, that is, ratios of n:1 and 1:m. Synchronous operation of the clocks can be 1:1, n:1, or 1:n.

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.

Reset

The SMC has up to three reset inputs:

aresetn	This is the reset signal for the **aclk** domain.
mreset0n	This is the reset signal for the **mclk0** domain.
mreset1n	If a second memory interface is included, this is the reset signal for the **mclk1** domain.

You can change both reset signals asynchronously to their respective clock domain. Internally to the SMC, the deassertion of the **aresetn** signal is synchronized to **aclk**. The deassertion of **mreset0n** is synchronized internally to **mclk0** and **mclk0n**, and similarly, **mreset1n** is synchronized to **mclk1** and **mclk1n**.

2.2.3 Miscellaneous signals

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

user_config[7:0]	General purpose output ports that the write-only APB register drives directly. If you do not require these ports, leave them unconnected. See *user_config Register* on page 3-25.
user_status[7:0]	General purpose input ports that are readable from the APB interface through the 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 **aclk** signal, then you require external synchronization registers. See *user_status Register* on page 3-25.

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

a_gt_m<x>_sync
When HIGH, it indicates that **aclk** is faster than, and synchronous to, **mclk<x>**.

async<x>
When HIGH, it indicates **aclk** is synchronous to **mclk<x>**. Otherwise, they are asynchronous. Ensure that **async<x>** is tied to the same value as **msync<x>**.

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

msync<x>
When HIGH, indicates **mclk<x>** is synchronous to **aclk**. Otherwise, they are asynchronous. Ensure that **msync<x>** is tied to the same value as **async<x>**.

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

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

2.2.4 APB slave interface operation

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

To enable a clean registered interface to the external infrastructure the APB interface, always adds a wait state for all reads and writes by driving **pready** LOW during the first cycle of the access phase.

Note

The *AMBA 3 APB Protocol v1.0 Specification* defines 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.

2.2.5 Format block

This section describes:

- *Hazard handling*
- *Exclusive accesses* on page 2-15
- *NAND memory accesses* on page 2-17
- *SRAM memory accesses* on page 2-24.

Hazard handling

The following types of hazard exist:

- Read after read (RAR)
- Write after write (WAW)
- Read after write (RAW)
- Write after read (WAR).

The AXI specification defines that RAW and WAR ordering is determined by the master, whereas RAR and WAW ordering is enforced by the slave. If an AXI master requires ordering between reads and writes to certain memory locations, it must wait for a write response before issuing a read from a location it has written to (RAW). It must also wait for read data before issuing a write to a location it has read from (WAR). The SMC ensures the ordering of read transfers from a single master is maintained (RAR), and additionally, that the ordering of write transfers from a single master is maintained (WAW).

RAR

RAR hazards only occur in configurations that have two memory interfaces.

The SMC can reorder reads from different masters that connect to different memory interfaces. This situation is likely to occur, for example, in an SMC (PL353 series) configuration, when one master is accessing an SRAM memory interface clocked at 133MHz, and another master is accessing the NAND memory interface clocked at 50MHz. Read data from the SRAM memory is available before data from the NAND memory. This enables the SMC to potentially return read data out of order. The SMC (PL350 series) contains internal hazard checking to ensure the AXI reads from a single master have the order maintained.

WAW

WAW hazards only occur in configurations that have two memory interfaces.

As for RAR hazards, writes to different memory interfaces are able to complete out of order. This enables the write responses to be returned out of order. The SMC internal hazard checking logic ensures only writes from different masters are completed out of order.

Exclusive accesses

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

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

Exclusive access monitors implement the exclusive access functionality. Each monitor can track a single exclusive access. The number of monitors is a configurable option.

If an exclusive write fails, the data mask for the write is forced LOW, so that the data is not written.

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

For the purposes of monitoring, address comparison is made using a bit mask derived in the following fashion.

Consider the byte addresses accessed by a transaction. All the least significant bits, up to and including, the most significant bit that vary between those addresses are set to logic zero in the mask. All the stable address bits above this point are set to logic one.

Example 2-1 provides information about three transactions.

Example 2-1

Exclusive Read	Address = `0x100`, size = WORD, length = 1, ID = 0.
Write	Address = `0x104`, size = WORD, length = 2, ID = 1.
Exclusive Write	Address = `0x100`, size = WORD, length = 1, ID = 0.

The write transaction accesses the address range `0x104-0x10B`. Therefore, address bit 3 is the most significant bit that varies between byte addresses. The bit mask is therefore formed so that address bits 3 down to 0 are not compared. This has the effect that the masked write, as far as the monitoring logic has calculated, has accessed the monitored address. Therefore the exclusive write is marked as having failed.

Table 2-1 lists the address comparison steps:

Table 2-1 Address comparison steps example

Step		Binary	Hex
1	Monitored address	b000100000000	`0x100`
2	Write address	b000100000100	`0x104`
3	Write accesses	b000100000100	`0x104`
		b000100000101	`0x105`
		b000100000110	`0x106`
		b000100000111	`0x107`
		b000100001000	`0x108`
		b000100001001	`0x109`
		b000100001010	`0x10A`
		b000100001011	`0x10B`
4	Generate a comparison mask	b111111110000	`0xFF0`
5	Monitored address ANDed with mask	b000100000000	`0x100`
6	Write Address ANDed with mask	b000100000000	`0x100`
7	Compare steps 5 and 6		
8	Mark exclusive write as failed		

This example shows how the logic can introduce false-negatives in exclusive access monitoring, because in reality the write has not accessed the monitored address. The implementation has been chosen to reduce design complexity, but always provides safe behavior.

When calculating the address region accessed by the write, the burst type is always taken to be INCR. Therefore, a wrapped transaction in Example 2-1 on page 2-15 that wraps down to `0x0` rather than cross the boundary, is treated in the same way. This is the same for a fixed burst that does not cross the boundary or wrap down to `0x0`.

NAND memory accesses

This section describes:

- *Two phase NAND accesses*
- *NAND command phase transfers* on page 2-18
- *NAND data phase transfers* on page 2-19.

Two phase NAND accesses

The SMC defines two phases of commands when transferring data to or from NAND flash.

Command phase

Commands and optional address information are written to the NAND flash. The command and address can be associated with either a data phase operation to write to or read from the array, or a status/ID register transfer.

Data phase Data is either written to or read from the NAND flash. This data can be either data transferred to or from the array, or status/ID register information.

The SMC uses information contained in the AXI address bus, either **awaddr[ ]** or **araddr[ ]** signals, to determine whether the AXI transfer is a command or data phase access.

This information contained in the address bus additionally determines:

- the value of the command
- the number of address cycles
- the chip select to be accessed.

During a command phase transfer, the address to be written to the NAND memory is transferred to the SMC using the AXI write channel.

Note

The size of the AXI transfer for data phase transfers must be larger than the width of the memory interface.

Table 2-2 lists the fields of **awaddr[ ]** and **araddr[ ]** signals that control a NAND flash transfer.

Table 2-2 NAND AXI address setup

AXI address	Command phase	Data phase
[31:24]	Chip address	Chip address
[23]	NoOfAddCycles_2	Reserved
[22]	NoOfAddCycles_1	Reserved
[21]	NoOfAddCycles_0	ClearCS
[20]	End command valid	End command valid[a]
[19]	0	1
[18:11]	End command	End command[b]
[10:3]	Start command	[10] ECC Last [9:3] Reserved
[2:0]	Reserved[c]	Reserved[c]

a. For a read data phase transaction, the end command valid must be 0.
b. End command data is ignored if end command valid is not true.
c. The bottom three bits of a NAND access determine the valid data byte lanes, in the same way as for a standard AXI access.

NAND command phase transfers

A command phase transfer is always performed as an AXI write. The AXI **awaddr[ ]** bus, and Table 2-2 contain the following information:

Address cycles

The number of address cycles can be any value from zero to seven. Generally, up to five cycles are used during an array read or write, but a maximum of seven enables support for future devices.

Start command

The NAND command is used to initiate the required operation, for example:

- page read

- page program
- random page read
- status or ID register read.

End command

The value of the second command, if required. This command is executed when all address cycles have completed. For example, some NAND memories require an additional command, following the address cycles, for a page read.

End command valid

Indicates whether the end command must be issued to the NAND flash.

Each address cycle consumes eight bits of address information. This is transferred to the SMC through the AXI write channel.

Note

To ease system integration, the SMC supports the use of multiple AXI write transactions to transfer address information. The following restrictions apply in this case:

1. The AXI address [31:3] bits must not change between transactions. The first transaction must be doubleword aligned.
2. All other address information must be the same, with the exception of transaction length.
3. Data must be transferred in incrementing, consecutive accesses, that is, not wrapping, fixed, or sparse.
4. Extra or unused beats in the last transaction must have write strobes disabled.
5. Total number of beats must be less than the write FIFO depth.

NAND data phase transfers

Transfers data to or from the NAND flash, and can be performed as either an AXI read or write, depending on the required operation. The **araddr[ ]** or **awaddr[ ]** bus, and Table 2-2 on page 2-18 contain this information:

End command

The value of a command that is issued following the data transfer. This is required by some memories to indicate a page program following input of write data.

Note

End commands are not supported for read data phase transfers.

End command valid

Indicates whether the end command must be issued to NAND flash.

ClearCS When set, the chip select for a NAND flash is de-asserted on completion of this command. When not set, the chip select remains asserted.

ECC Last When set, this bit indicates to the ECC block that the current command is the last access to the NAND page. It is ignored if the ECC block is not enabled. See *Error Correction Code operation* on page 2-48.

A NAND flash data phase program or read operation is expected to require multiple AXI transfers because of the large page size of NAND memories. You can therefore use the ClearCS bit to hold a chip select asserted while multiple AXI transfers transfer data to or from the NAND flash internal page. On the last AXI transfer, you can de-assert the chip select by setting the ClearCS bit.

Note

Using the optional ClearCS functionality causes the controller to keep requesting the EBI, until the CS is cleared. You must take care, at the system level, to ensure that this does not cause a deadlock when using the EBI.

Figure 2-7 on page 2-21 and Figure 2-8 on page 2-22 show the steps taken to perform NAND flash page read and page program operations respectively.

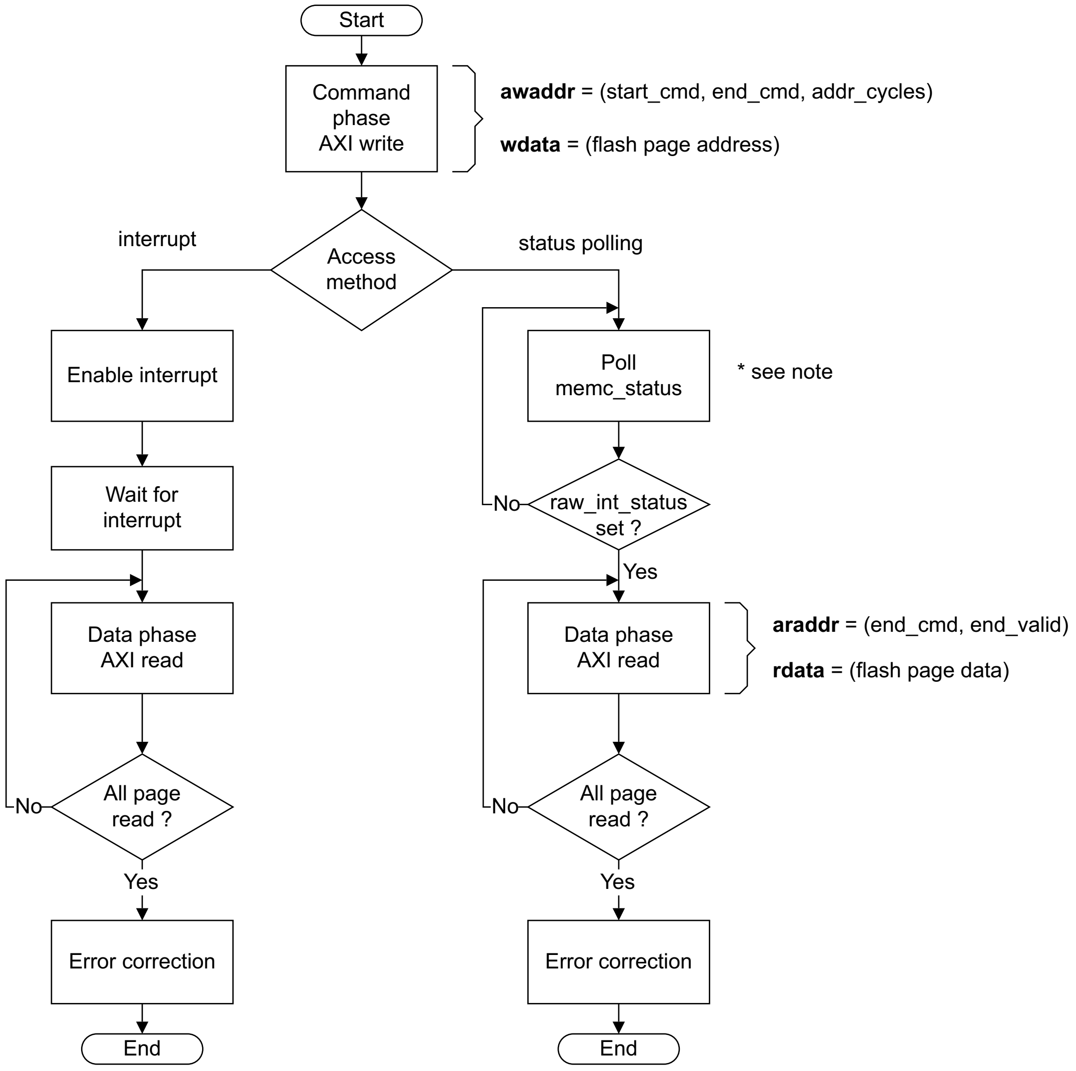


Figure 2-7 NAND flash page read operations

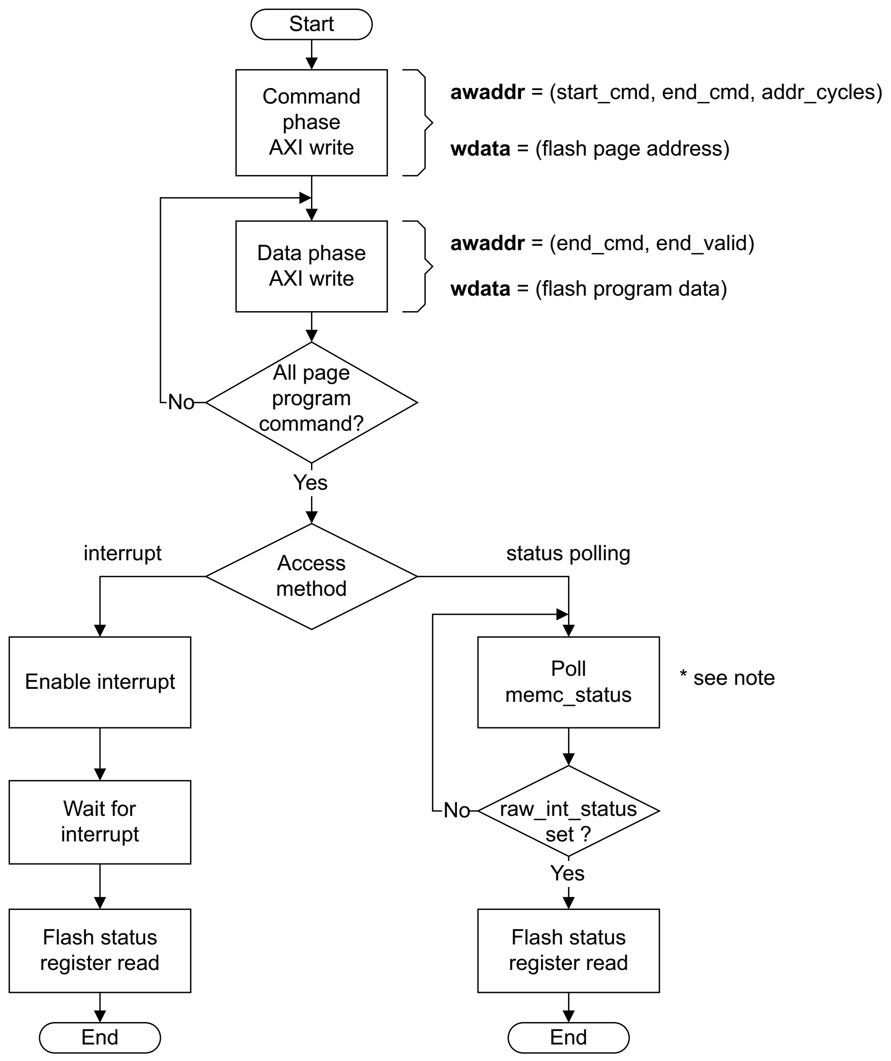

Start
Command phase AXI write
awaddr = (start_cmd, end_cmd, addr_cycles)
wdata = (flash page address)
Data phase AXI write
awaddr = (end_cmd, end_valid)
wdata = (flash program data)
All page program command?
No
Yes
Access method
interrupt
status polling
Enable interrupt
Wait for interrupt
Flash status register read
End
Poll memc_status
* see note
raw_int_status set ?
No
Yes
Flash status register read
End

Figure 2-8 NAND flash page program operations

Note

You can poll for either a page program or page read completion in two ways:

- Poll the raw_int_status bit in the memc_status Register to determine when the memory **busy** output has gone HIGH, indicating a page program completion or read data ready.

- In a system with multiple NAND flash devices connected to the SMC. The **busy** outputs are wire-ANDed to produce the single **busy** input to the SMC, that only transitions HIGH when all devices have completed. You can determine the status register of each NAND chip by reading the individual device status register.

Figure 2-9 shows the steps taken to perform a NAND flash status register read.

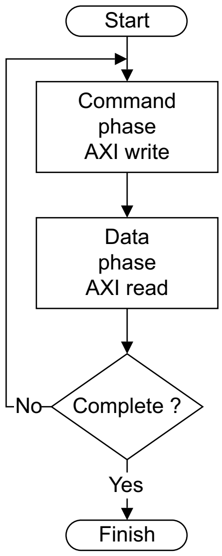


Figure 2-9 NAND flash status register read

The process boxes that Figure 2-9 shows are defined as:

Command phase AXI write

awaddr = (`start_cmd = STATUS_READ_CMD`,

`end_cmd = 0x0`

`end_valid = 0x0`

`addr_cycles = 0x0`)

Note

Ensure burst length is 1, **awlen** = `0x0`.

wdata = (don’t care).

Data phase AXI read

araddr = (`ClearCS` = `0x1`).

Note

Ensure burst length is 1, **arlen** = `0x0`.

The transfer size is eight or sixteen, **arsize** = `0x0` or `0x1`.

rdata = NAND flash status output.

Note

Certain NAND flash devices can support multiple status register reads without reissuing the STATUS_READ_CMD. In this case, you can modify the flow that *NAND flash status register read* on page 2-23 describes to include multiple data phase transfers for each command phase transfer.

The upper byte of the address read or write bus, **awaddr[31:24]** or **araddr[31:24]**, and the value of the address_match[ ] and address_mask[ ] buses determine the chip select being accessed. To match a chip, **axaddr[31:24]** & **address_mask[ ]** must equal **address_match[ ]**.

The values for the address_mask and address_match[] buses must be set so that no address maps onto more than one chip. If an address does not map onto any chip, the behavior of the controller is undefined.

SRAM memory accesses

This section describes:

- *Standard SRAM accesses*
- *Memory address shifting* on page 2-25
- *Memory burst alignment* on page 2-25
- *Memory burst length* on page 2-26
- *Booting using the SRAM interface* on page 2-26.

Standard SRAM accesses

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

The upper byte of the address read or write bus, **awaddr[31:24]** or **araddr[31:24]**, and the value of the address_match[ ] and address_mask[ ] buses determine the chip select being accessed. To match a chip, **axaddr[31:24]** & **address_mask[ ]** must equal **address_match[ ]**. The values for the address_mask and address_match[] buses must be set so that no address maps onto more than one chip. If an address does not map onto any chip, the behavior of the controller is undefined.

In addition to reads and writes, exclusive reads and writes are supported in accordance with the AMBA AXI Protocol v1.0 Specification.

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

Note

The **arcache**, **awcache**, **arprot** and **awprot** signals are included in the AXI interface list for completeness only. The controller does not use the information transferred by these signals.

Memory address shifting

To produce the address presented to the memory device, the AXI address is aligned to the memory width. This is done because the AXI address is a byte-aligned address, whereas 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 an AXI burst crosses a memory burst boundary, the SMC partitions the AXI transfer into multiple memory bursts, terminating a memory transfer at the burst boundary. 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 AXI 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. The length of memory bursts are however automatically limited by the size of the read or write data FIFOs.

For read transfers, the maximum memory burst length on the memory interface is the depth of the read data FIFO. For writes, the maximum burst length is dependent on:

- the beat size of the AXI transfer, `asize`
- the memory data bus width, `mw`
- the depth of the write data FIFO depth, `wfifo_depth`.

The formula to determine the maximum memory write burst length is:

Memory write burst length = `((1<<asize) x wfifo_depth) / (1<<mw)`

Booting using the SRAM interface

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 **remap_<x>** port signal is HIGH, the memory with the bootable chip select is set by the **sram_mw[ ]** tie-off port signal.

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

2.2.6 Memory manager operation

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

This subsection describes:

- *AXI low-power operation*
- *Chip configuration registers* on page 2-27
- *Direct commands* on page 2-28.

AXI low-power operation

The SMC accepts requests to enter the low-power state through either the AXI 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 AXI interface
- all FIFOs are empty
- all memory interface blocks are IDLE.

When the low-power state is entered, the AXI outputs **awready**, **arready**, and **wready** are driven LOW to prevent any new AXI transfers being accepted. No new AXI 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 with that of the memory device.

The set_cycles Register and set_opmode Register act as holding registers for new operating parameters until the SMC detects the memory device has switched modes. This enables a memory device to be made to change its operating mode while still being accessed.

Figure 2-10 on page 2-28 shows the memory manager containing a bank of registers for each memory chip supported by the SMC configuration. The manager register bank consists of all the timing parameters `chip<x>_cycles`, and access modes `chip<x>_opmode`. These are required for the controller to correctly time any type of access to a supported memory type.

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

- the direct_cmd Register indicates only a register update is taking place
- the direct_cmd register indicates a mode register access either using the direct_cmd register or using the AXI interface and the command has completed.

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

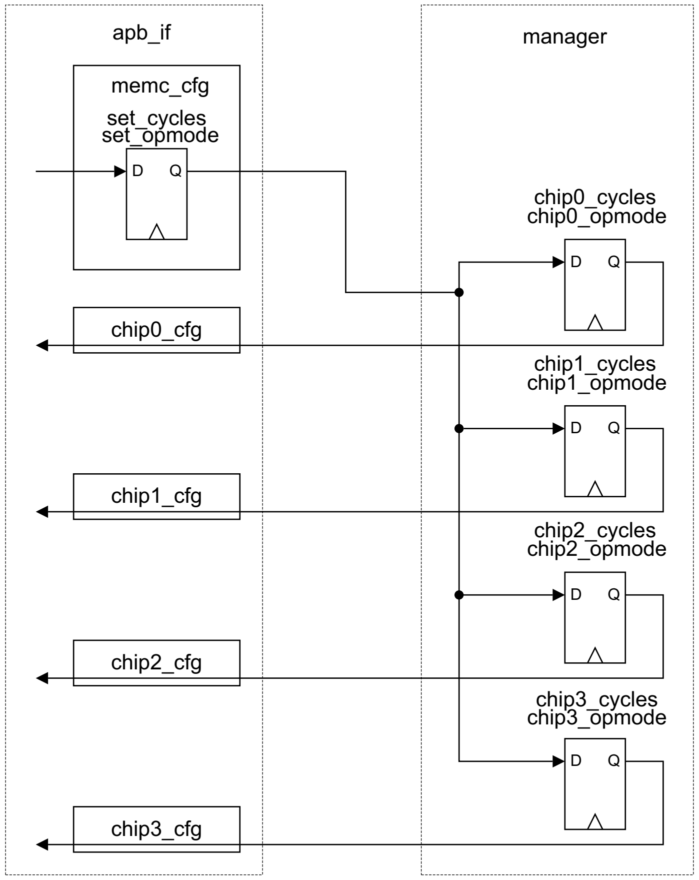


Figure 2-10 Chip configuration registers

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. These are:

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 implemented through the APB direct_cmd Register. Figure 2-11 shows the sequence of events.

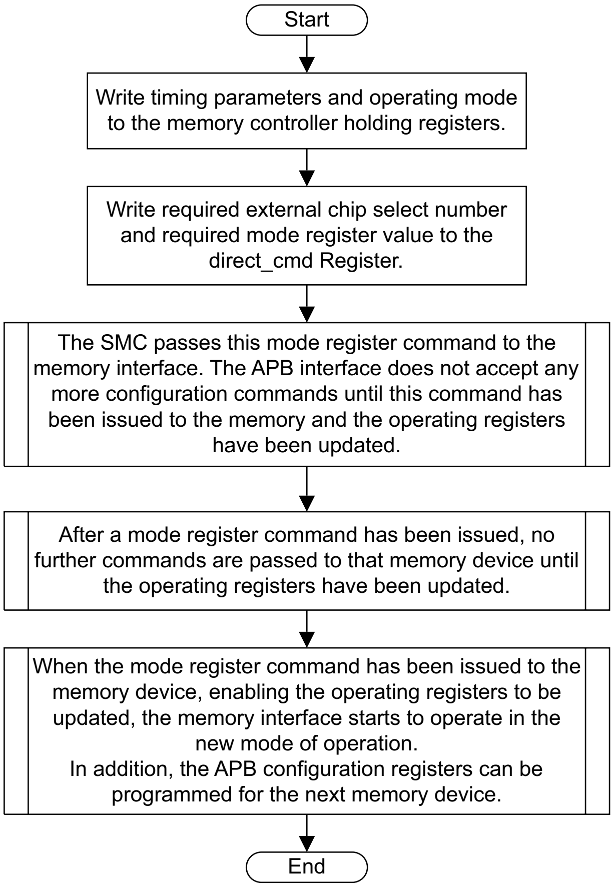

Start
Write timing parameters and operating mode to the memory controller holding registers.
Write required external chip select number and required mode register value to the direct_cmd Register.
The SMC passes this mode register command to the memory interface. The APB interface does not accept any more configuration commands until this command has been issued to the memory and the operating registers have been updated.
After a mode register command has been issued, no further commands are passed to that memory device until the operating registers have been updated.
When the mode register command has been issued to the memory device, enabling the operating registers to be updated, the memory interface starts to operate in the new mode of operation.
In addition, the APB configuration registers can be programmed for the next memory device.
End

Figure 2-11 Device pin mechanism

Software mechanism

For memories that require a sequence of read and write commands, for example, most NOR flash devices use the AXI interface, with the write data bus used to indicate when the last transfer has completed and when it is safe for the SMC to update the chip configuration registers. Figure 2-12 shows the sequence of events.

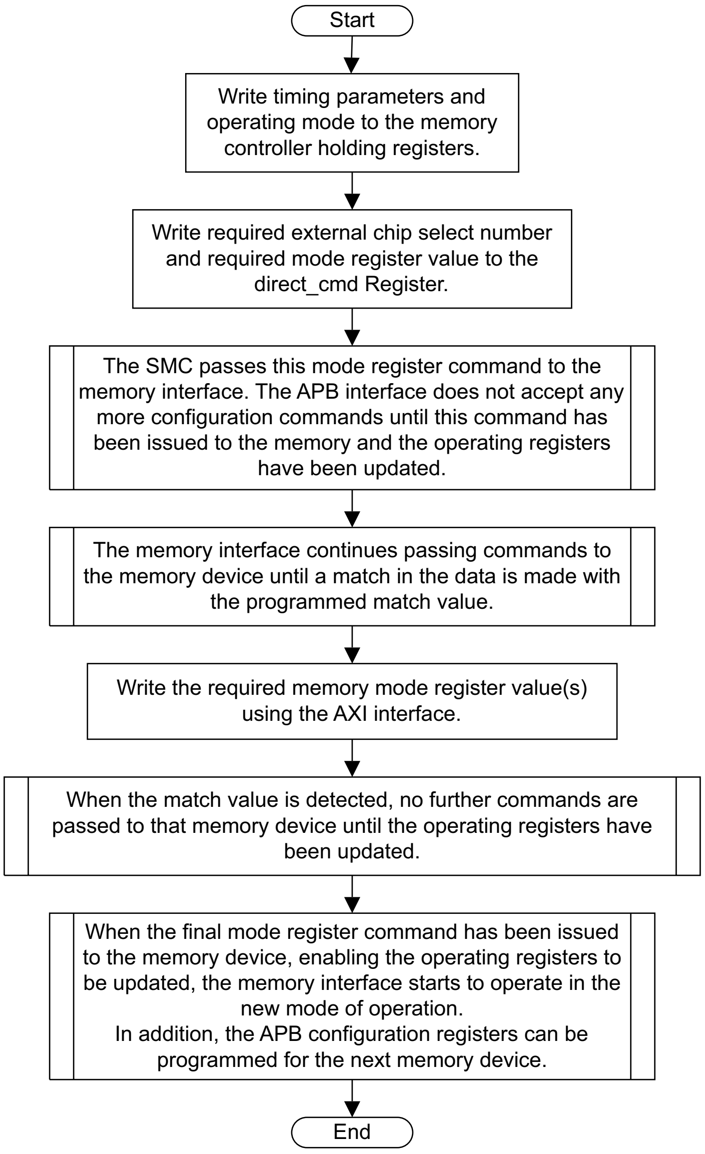

Start
Write timing parameters and operating mode to the memory controller holding registers.
Write required external chip select number and required mode register value to the direct_cmd Register.
The SMC passes this mode register command to the memory interface. The APB interface does not accept any more configuration commands until this command has been issued to the memory and the operating registers have been updated.
The memory interface continues passing commands to the memory device until a match in the data is made with the programmed match value.
Write the required memory mode register value(s) using the AXI interface.
When the match value is detected, no further commands are passed to that memory device until the operating registers have been updated.
When the final mode register command has been issued to the memory device, enabling the operating registers to be updated, the memory interface starts to operate in the new mode of operation.
In addition, the APB configuration registers can be programmed for the next memory device.
End

Figure 2-12 Software mechanism

2.2.7 Interrupts operation

The busy outputs of each chip are wire-ANDed together, external to the controller, to create a single **busy** (NAND interfaces) or **int** (SRAM interfaces) signal. This signal creates an internal interrupt input per memory interface. Multiple outstanding accesses to NAND chips only trigger an interrupt when all chips have completed the respective operations. During the busy phase, you can read the status register of each chip to determine whether the chips that have completed.

An interrupt is cleared by the next AXI read to any chip select on the appropriate memory interface, or by a write to the appropriate bit in the `memc_cfg_clr` register.

The interrupt outputs are generated through a combinational path from the relevant input pin. This enables the SMC to be placed in low-power state, and the clocks stopped, while waiting for an interrupt.

When interrupts are disabled, a synchronized version of the interrupt input is still readable through the APB interface. This enables software to poll, rather than use an interrupt to determine when NAND operations can proceed. There is also an interrupt for each ECC block. See *Error Correction Code operation* on page 2-48.

2.2.8 Memory interface operation

The memory interface issues commands to the memory from the command FIFO, and controls the cycle timings of these commands. It only issues a new command after the previous command is complete and any turn-around times have been met. It only issues a read command when there is space for all the impending data in the read data FIFO.

Note

- The `rd_bl` parameter in the opmode Register must not be set greater than the read data FIFO depth.
- The controller does not perform WRAP transfers on the memory interface. For memory devices that only operate in WRAP mode, you must align transfers to a memory burst boundary. If the controller performs transfers that cross a memory boundary, then you must program the memory device to operate in INCR mode.

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

Figure 2-13 on page 2-33 to Figure 2-24 on page 2-45 show the timing parameters. They are divided into the following:

- *SRAM timing tables and diagrams* on page 2-32
- *NAND timing tables and diagrams* on page 2-43.

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 **mclk<x>n**, equivalent to the falling edge of **mclk<x>**, for both synchronous and asynchronous accesses. The clock output to memory, **clk_out**, is driven directly by **mclk<x>**, 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 **mclk<x>n**, equivalent to the falling edge of **mclk<x>**, for asynchronous reads. For synchronous reads, read data is registered using the fed-back clock, **fbclk_in**. For synchronous and asynchronous accesses, the data is then pushed onto the read data FIFO to be returned by the AXI interface.

This subsection describes:

- *Asynchronous read*
- *Asynchronous read in multiplexed mode* on page 2-33
- *Asynchronous write* on page 2-34
- *Asynchronous write in multiplexed mode* on page 2-35
- *Asynchronous page mode read* on page 2-36
- *Synchronous burst read* on page 2-37
- *Synchronous burst read in multiplexed mode* on page 2-38
- *Synchronous burst write* on page 2-39
- *Synchronous burst write in multiplexed mode* on page 2-40
- *Synchronous read and asynchronous write* on page 2-41
- *Programming tRC and tWC when the controller operates in synchronous mode* on page 2-42
- *Chip select assertion for SRAM memory interfaces* on page 2-43.

Asynchronous read

Table 2-3 on page 2-33 and Table 2-4 on page 2-33 list the opmode<x>_<n> and sram_cycles Register settings. See *Register summary* on page 3-4.

Where:

- x = 0 or 1, for interface 0 and 1

- n = 0 to 3, for chip select.

Table 2-3 Asynchronous read opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	0	3’b000	-	-	-	-	-	-

Table 2-4 Asynchronous read sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	4’b0011	-	3’b001	-	-	-

Figure 2-13 shows a single asynchronous read transfer with an initial access time, t_{RC}, of 3 cycles and an output enable assertion delay, t_{CEOE}, of one cycle.

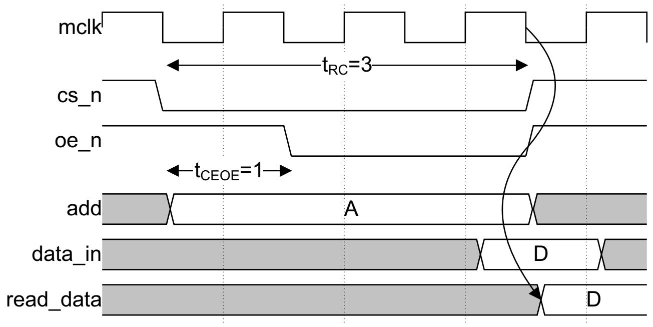


Figure 2-13 Asynchronous read

Asynchronous read in multiplexed mode

Table 2-5 and Table 2-6 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-5 Asynchronous read in multiplexed mode opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	0	3’b000	-	-	-	1	-	-

Table 2-6 Asynchronous read in multiplexed mode sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	4’b0111	-	3’b101	-	-	-

Figure 2-14 shows a single asynchronous read transfer in multiplexed SRAM mode, with t_{RC}=7, and t_{CEOE}=5.

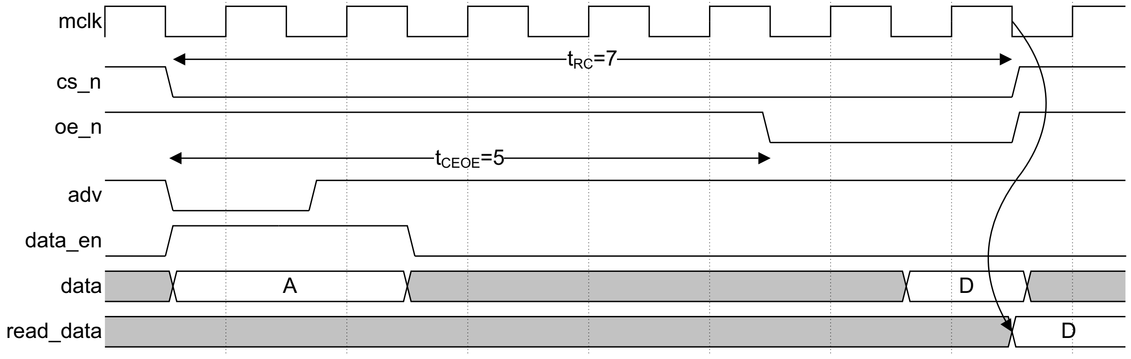


Figure 2-14 Asynchronous read in multiplexed mode

Note

In multiplexed mode, both address and data are output by the SMC on the **data_out** output bus. Read data is accepted on the **data_in** bus. The address is still driven onto the address bus in multiplexed mode. This enables you to use the upper address bits for memories that require more address bits than data bits.

Asynchronous write

Table 2-7 and Table 2-8 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-7 Asynchronous write opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	-	-	0	3’b000	-	-	-	-

Table 2-8 Asynchronous write sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	-	4’b0100	-	3’b010	-	-

Figure 2-15 on page 2-35 shows an asynchronous write with a write cycle time t_{WC} of four cycles and a **we_n** assertion duration, t_{WP}, of two cycles.

Note

The timing parameter t_{WP} controls the deassertion of **we_n**. You can use it to vary the hold time of **cs_n**, **addr** and **data**. This differs from the read case where the timing parameter t_{CEOE} controls the delay in the assertion of **oe_n**. Additionally, **we_n** is always asserted one cycle after **cs_n** to ensure the address bus is valid.

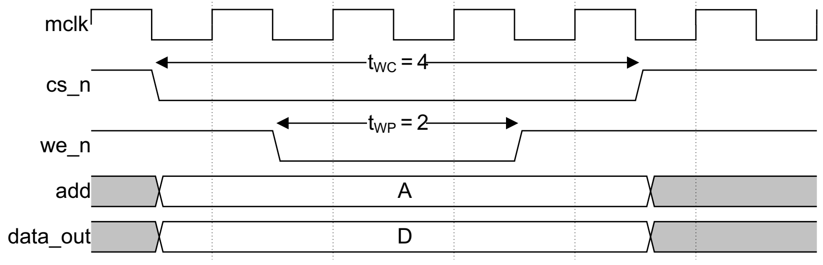


Figure 2-15 Asynchronous write

Asynchronous write in multiplexed mode

Table 2-9 and Table 2-10 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-9 Asynchronous write in multiplexed mode opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	-	-	0	3'b000	0	0	-	-

Table 2-10 Asynchronous write in multiplexed mode sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	-	4'b0111	-	3'b100	-	-

Figure 2-16 on page 2-36 shows an asynchronous write in multiplexed mode. t_{WC} is seven cycles, and t_{WP} is four cycles.

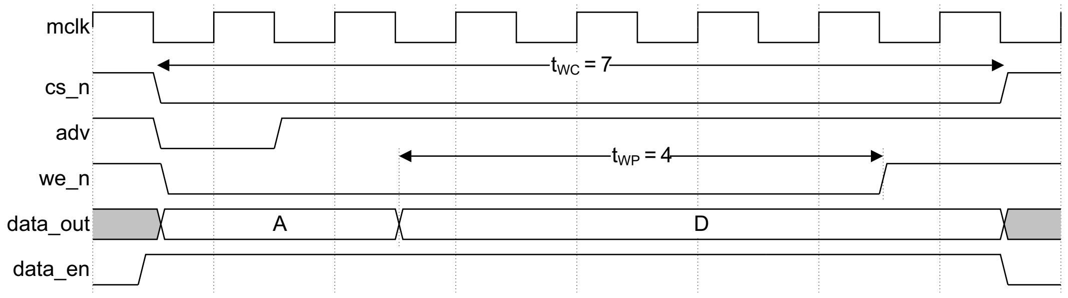


Figure 2-16 Asynchronous write in multiplexed mode

Note

For writes in multiplexed mode, you must set $t_{WC} \geq t_{WP} + 2$. The constant value of two represents the two **mclk** cycles of the address phase.

Asynchronous page mode read

Table 2-11 and Table 2-12 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-11 Page read opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	0	<page length>	-	-	-	-	-	1

Table 2-12 Page read sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	4’b0011	-	3’b010	-	3’b001	-

Figure 2-17 on page 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.

You enable Page mode 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.

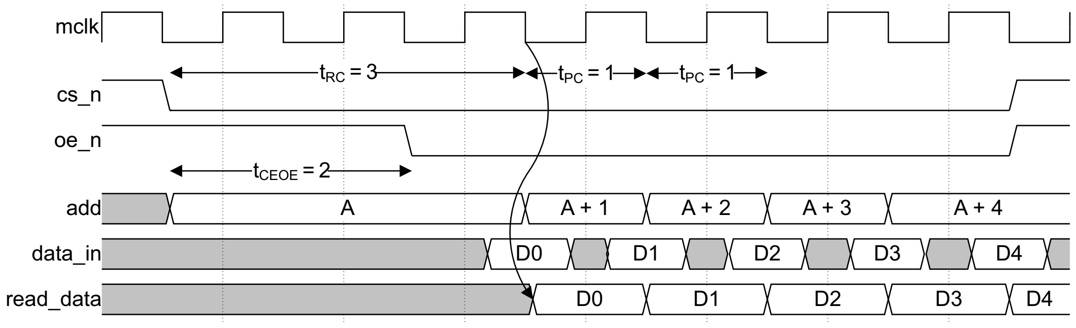


Figure 2-17 Page read

Synchronous burst read

Table 2-13 and Table 2-14 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-13 Synchronous burst read opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	1	<burst length>	-	-	-	1	-	-

Table 2-14 Synchronous burst read sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	4'b0100	-	3'b010	-	-	-

Figure 2-18 on page 2-38 shows a burst read with the **wait** output of the memory used to delay the transfer.

Note

- Synchronous memories have a configuration register enabling **wait** to be asserted either on the same clock cycle as the delayed data, or a cycle early. The SMC only supports **wait** being asserted one cycle early, enabling **wait** to be initially sampled with the fed-back clock and then with **mclk** before being used by the FSM. This enables the easiest timing closure. Additionally, you must configure the memory for **wait** to be active LOW.
- In synchronous operation, the SMC relies on the **wait** signal being de-asserted HIGH to indicate that the memory can finish the transfer. When in synchronous mode, some memories do not de-assert the **wait** 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.

- You must set t_{RC} to a value that enables **wait_reg_mclk** to stabilize. See Figure 2-18.

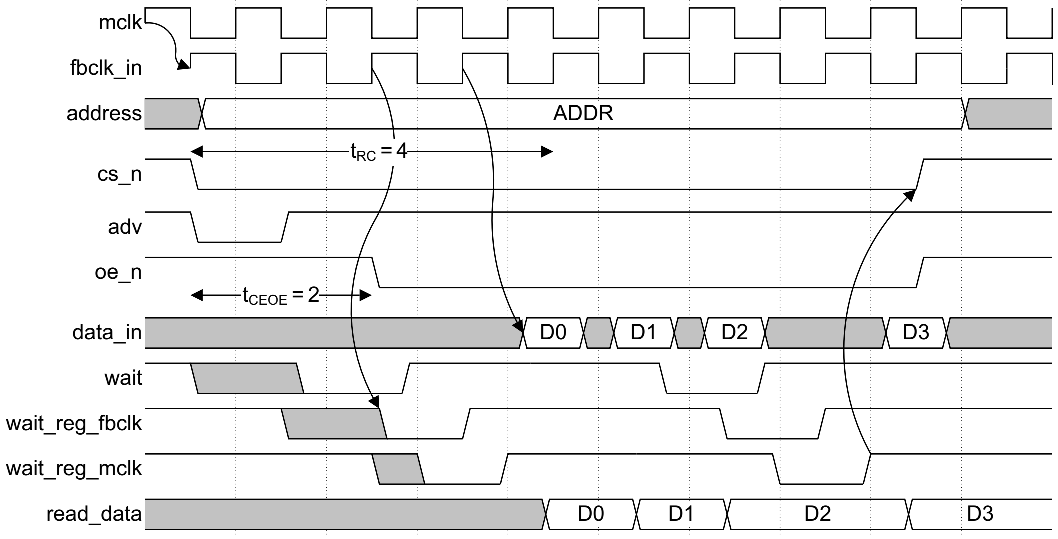


Figure 2-18 Synchronous burst read

Synchronous burst read in multiplexed mode

Table 2-15 and Table 2-16 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-15 Synchronous burst read in multiplexed mode opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	1	<burst length>	-	-	-	-	-	-

Table 2-16 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	4'b0100	-	3'b010	-	-	-

Figure 2-19 shows the same synchronous read burst transfer as Figure 2-18 on page 2-38, but in multiplexed mode.

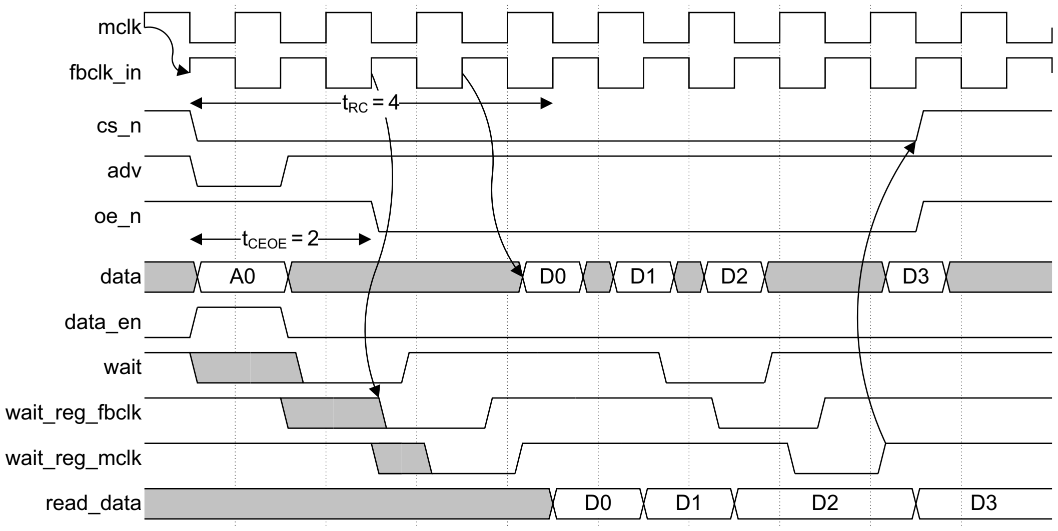


Figure 2-19 Synchronous burst read in multiplexed mode

Synchronous burst write

Table 2-17 and Table 2-18 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-17 Synchronous burst write opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	-	-	1	<burst length>	-	1	-	-

Table 2-18 Synchronous burst write sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	-	4’b0100	-	3’b001	-	-

Figure 2-20 on page 2-40 shows a synchronous burst write transfer that is delayed by the **wait** signal. You must configure the memory to assert **wait** one cycle early and with an active LOW priority. The **wait** signal is again registered with the fed-back clock and **mclk** before being used. The **wait** signal is used in the **mclk** domain to the memory interface FSM.

—— **Note** ——

- Synchronous memories have a configuration register enabling **wait** to be asserted either on the same clock cycle as the delayed data, or a cycle early. The SMC only supports **wait** being asserted one cycle early, enabling **wait** to be initially sampled with the fed-back clock and then with **mclk** before being used by the FSM. This enables the easiest timing closure. Additionally, you must configure the memory for **wait** to be active LOW.
- You must set t_{WC} to a value that enables **wait_reg_mclk** to stabilize. See Figure 2-20.

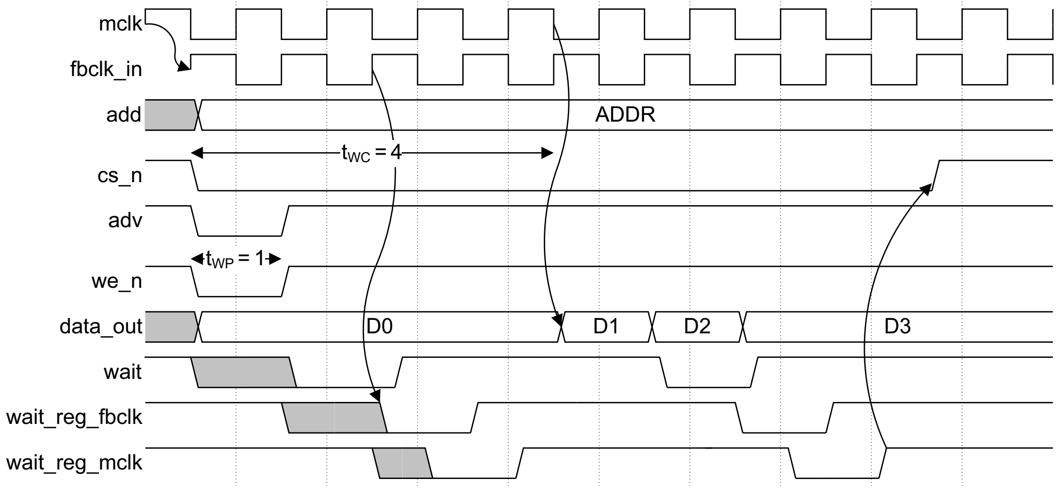


Figure 2-20 Synchronous burst write

Synchronous burst write in multiplexed mode

Table 2-19 and Table 2-20 on page 2-41 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-19 Synchronous burst write in multiplexed mode opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	-	-	1	<burst length>	-	1	-	-

Table 2-20 Synchronous burst write in multiplexed mode sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	-	4’b0100	-	3’b001	-	-

Figure 2-21 shows the same synchronous burst write as Figure 2-20 on page 2-40, but in multiplexed mode.

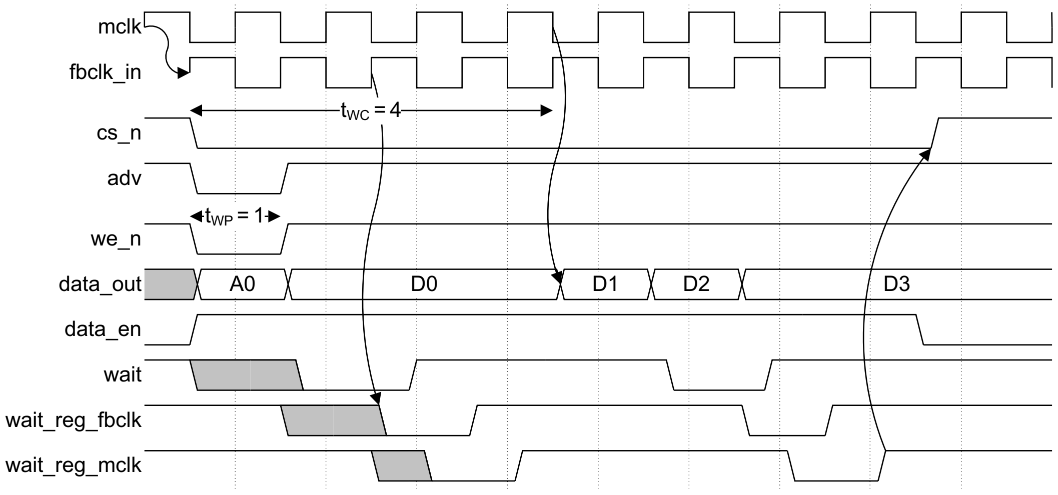


Figure 2-21 Synchronous burst write in multiplexed mode

Synchronous read and asynchronous write

Table 2-21 and Table 2-22 list the opmode<x>_<n> and sram_cycles Register settings.

Table 2-21 Synchronous read and asynchronous write opmode Register settings

Field	mw	rd_sync	rd_bl	wr_sync	wr_bl	baa	adv	bls	ba
Value	-	1	3’b001	0	3’b000	0	1	0	-

Table 2-22 Synchronous read and asynchronous write sram_cycles Register settings

Field	t_rc	t_wc	t_ceoe	t_wp	t_pc	t_tr
Value	4’b0100	4’b0110	3’b010	3’b001	-	3’b011

Figure 2-22 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.

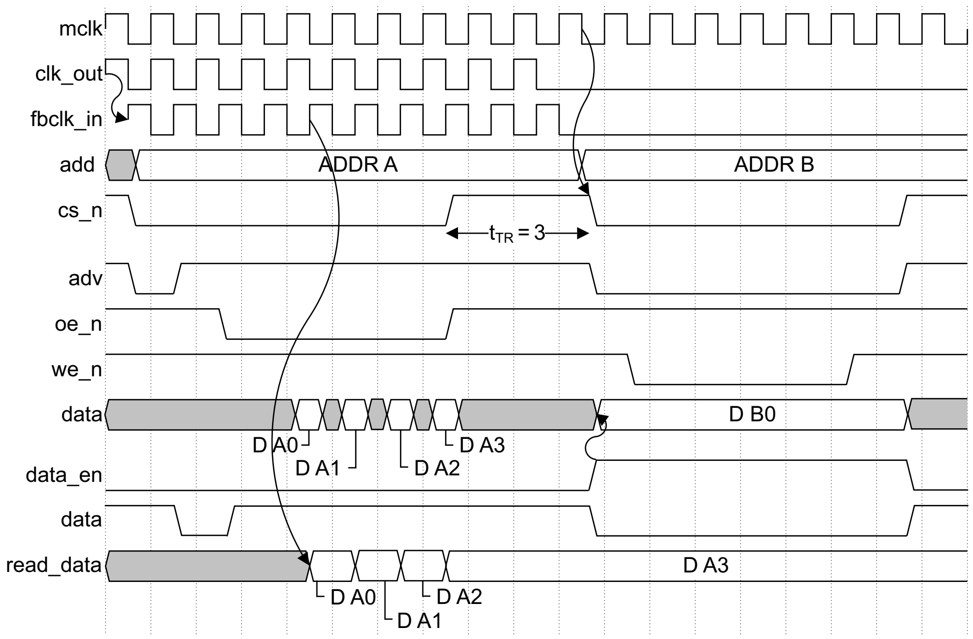


Figure 2-22 Synchronous read and asynchronous write

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.

Chip select assertion for SRAM memory interfaces

During repeated access to the same chip, the SMC can keep chip select asserted. To support memories that require chip select to be deasserted periodically, you can program the refresh_period_<x> Register to set a maximum number of consecutive memory bursts. You can set the number of consecutive bursts from one to 15, inclusive. See *refresh_period_0 Register* on page 3-18 and *refresh_period_1 Register* on page 3-19.

NAND timing tables and diagrams

All NAND control and data outputs are registered on the falling edge of **mclk**. Additionally, read data from the memory device is registered by the SMC on the falling edge of **mclk** before being pushed onto the read data FIFO.

Note

NAND opmode registers only set memory width and are not included in this section.

The following apply to NAND accesses:

Command phases

When doing a command with address cycles = 0, always enable at least one byte lane.

Data phases

Read data phases cannot have end commands associated with them.

This section describes:

- *Command phase access*
- *Data phase access* on page 2-45
- *Command-to-data phase access timing* on page 2-46.

Command phase access

Table 2-23 lists the nand_cycles Register settings.

Table 2-23 NAND flash address input nand_cycles Register settings

Field	t_rc	t_wc	t_rea	t_wp	t_clr	t_ar	t_rr
Value	-	4'b0010	-	3'b001	-	-	-

Figure 2-23 shows an address input phase. The cycle time is set to two, and the **we_n** assertion duration set to one. The address consists of three cycles, and the second command is also required.

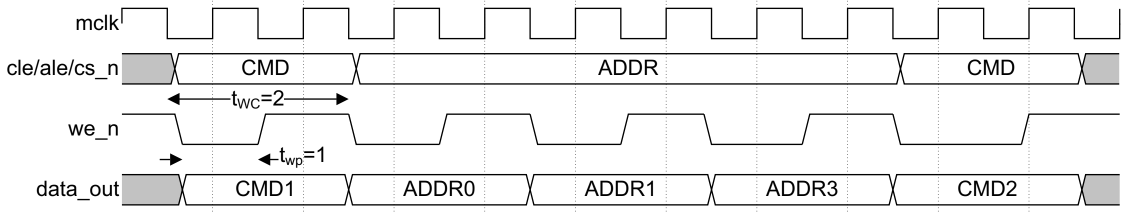


Figure 2-23 NAND flash address input

Table 2-24 lists example **awaddr** fields for NAND flash address input.

Table 2-24 NAND flash address input example awaddr fields

awaddr bits	Value	Description
[31:24]	Chip select	-
[23:21]	3'b011	Three address cycles
[20]	1	End command required
[19]	0	Command phase transfer

Table 2-24 NAND flash address input example awaddr fields (continued)

awaddr bits	Value	Description
[18:11]	CMD2	-
[10:3]	CMD1	-
[2:0]	3'b000	Address alignment

Data phase access

Table 2-25 lists the nand_cycles Register settings.

Table 2-25 NAND flash read nand_cycles Register settings

Field	t_rc	t_wc	t_rea	t_wp	t_clr	t_ar	t_rr
Value	4’b0011	-	3’b010	-	-	-	-

Figure 2-24 shows a read from NAND flash. The cycle time is set to three and the **re_n** assertion delay to two cycles. Three data items are read.

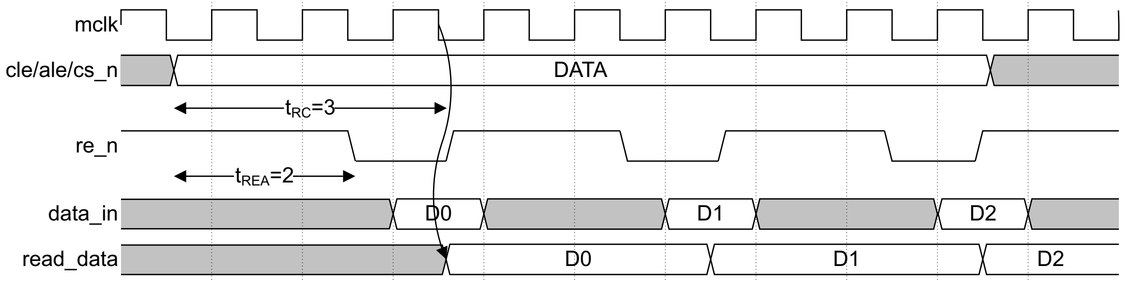


Figure 2-24 NAND flash read

Table 2-26 lists example **araddr** fields for NAND flash page read.

Table 2-26 NAND flash page read example araddr fields

araddr bits	Value	Description
[31:24]	Chip select	-
[23:22]	2'b00	Reserved
[21]	1	Last transfer, de-assert chip select when complete
[20]	0	End command required

Table 2-26 NAND flash page read example araddr fields (continued)

araddr bits	Value	Description
[19]	1	Data phase transfer
[18:11]	CMD2	-
[10:3]	8'b0000_0000	Reserved
[2:0]	3'b000	Address alignment

Command-to-data phase access timing

Table 2-27 lists the address latch to data phase Register settings.

Table 2-27 Address latch to data phase Register settings

Field	t_rc	t_wc	t_rea	t_wp	t_clr	t_ar	t_rr
Value	4’b0011	4'b0010	3’b010	3'b001	-	3'b010	-

Figure 2-25 shows t_{AR} that is the number of extra cycles delay between Address latch (**ale**) falling and the start of a new `data_phase` command.

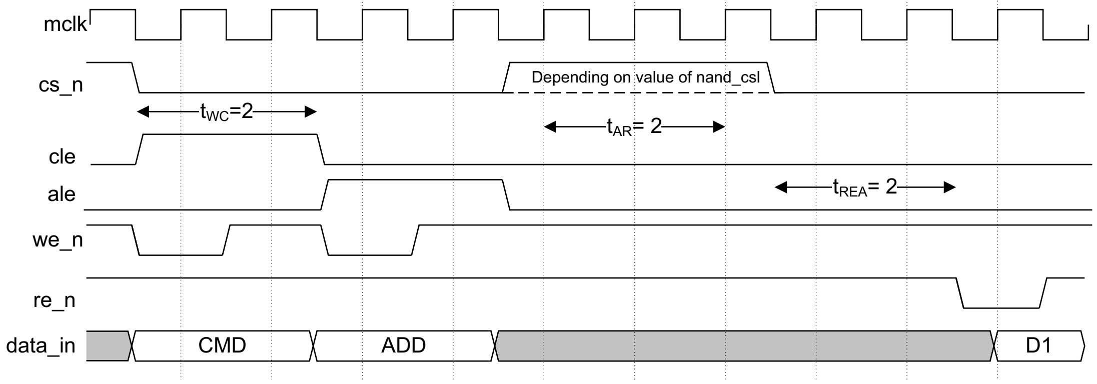


Figure 2-25 Address latch to data phase command

Table 2-28 lists the busy synchronization to data phase Register settings.

Table 2-28 Busy synchronization to data phase Register settings

Field	t_rc	t_wc	t_rea	t_wp	t_clr	t_ar	t_rr
Value	4’b0011	-	3’b010	-	-	=	3'b010

Figure 2-26 shows t_{RR} that is the number of extra cycles delay between the synchronization of the **busy** signal and the start of the next `data_phase` command. It is is only used when **nand_boot_en** is asserted.

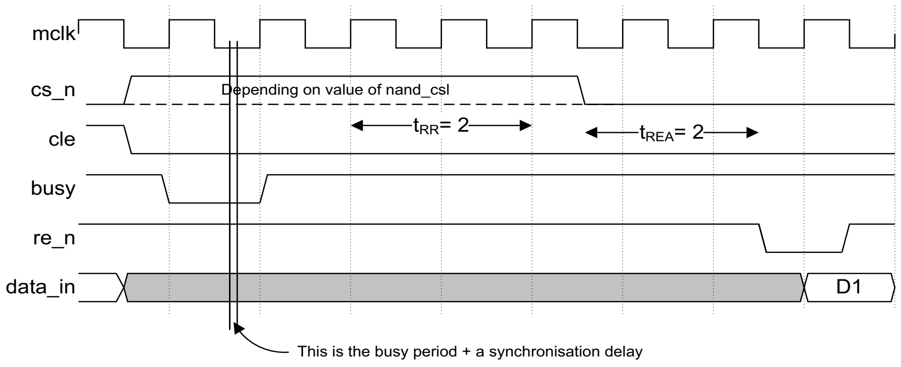

mclk
cs_n
cle
busy
re_n
data_in
Depending on value of nand_csl
t_{RR}= 2
t_{REA}= 2
D1
This is the busy period + a synchronisation delay

Figure 2-26 Busy synchronization to data phase command

Table 2-29 lists the command latched to data phase Register settings.

Table 2-29 Command latched to data phase Register settings

Field	t_rc	t_wc	t_rea	t_wp	t_clr	t_ar	t_rr
Value	4'b0011	4'b0010	3'b010	-	3'b010	=	-

Figure 2-27 shows the t_{CLR} delay that is the number of extra cycles delay between a command being latched, **cle** HIGH, and the start, **CS** asserted, of a `data_phase` command.

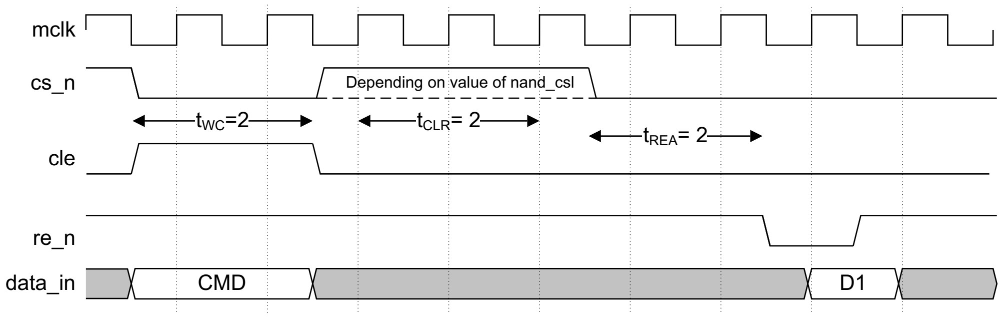


Figure 2-27 Command latched to data phase command

Note

The t_{CLR} delay is applied before both read and write data phase commands.

2.2.9 Error Correction Code operation

An ECC block can be included for each NAND interface at the configuration stage. It operates on a number of 512 byte blocks of NAND memory and can be programmed to store the ECC codes after the data in memory. For writes, the ECC is written to the spare area of the page. For reads, the result of a block ECC check are made available to the device driver.

Note

Because there is a no standard interface for NAND memory devices, it is important to know the characteristics of a particular memory type, before you enable ECC functionality within the SMC.

A configuration option enables an extra block of 4, 8, 16 or 32 bytes to be included at the end of the page, before the start of the ECC codes. Figure 2-28 shows the ECC block structure in memory.

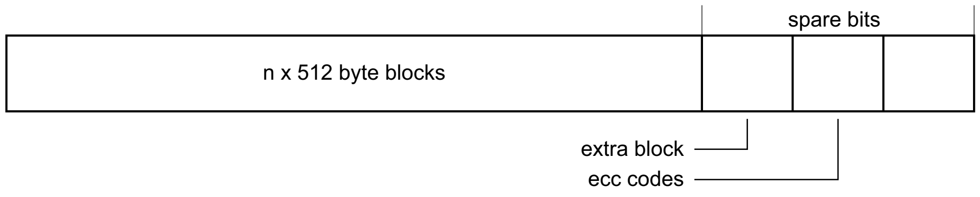


Figure 2-28 ECC block structure

Operation

The ECC calculation uses a simple *Hamming* code, using 1 bit correction 2 bit detection. It starts when a valid read or write command with a 512 byte aligned address is detected on the memory interface, and the block is enabled using the `ecc_memcfg` register. Values stored in the `ecc_memcommand<x>` registers are used to detect the start of an address phase access.

Figure 2-29 on page 2-49 gives an overview of how the ECC operates.

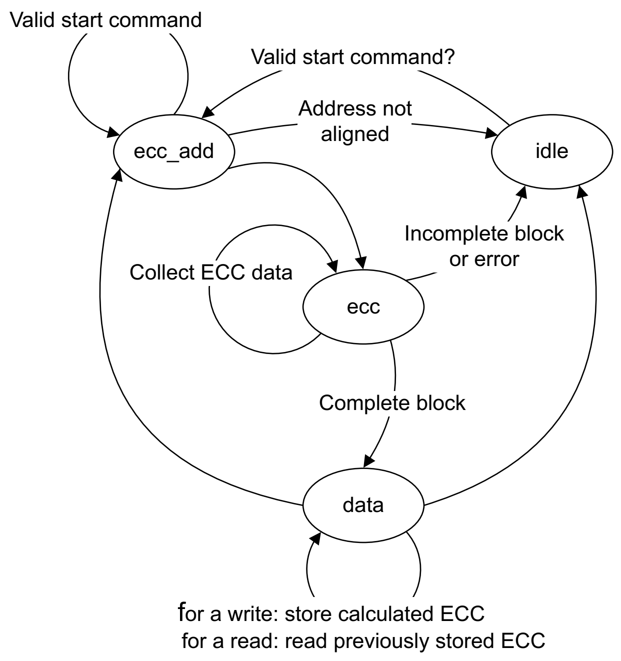


Figure 2-29 ECC state diagram

A 24-bit ECC code is generated for each 512 byte block and a shorter code between 10 and 16 bits for the extra block.

——— **Note** ———

For a 16-bit interface, ECC codes are written to memory aligned to 16-bit boundaries.

———————

Figure 2-30 on page 2-50 shows the basic operation with no reading ECC values between blocks.

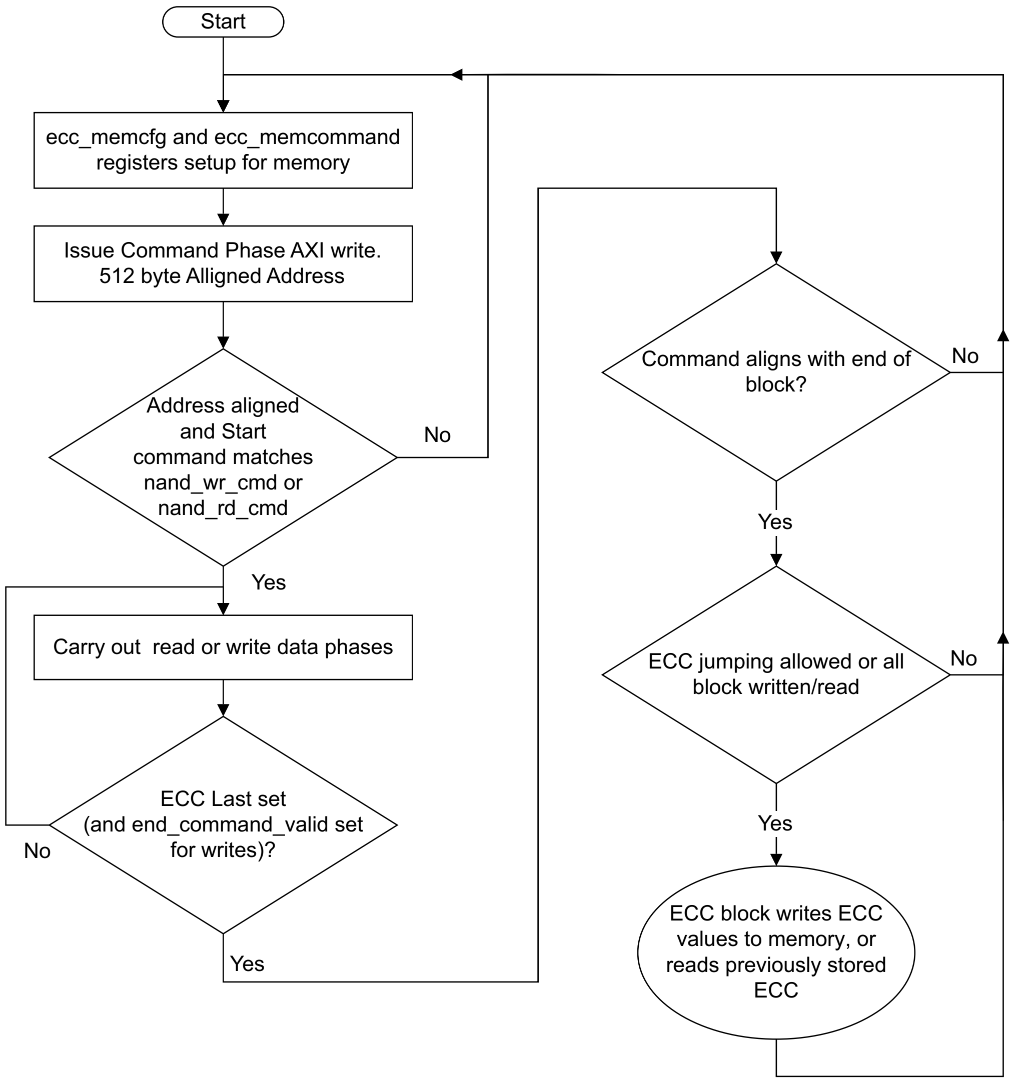

Start
ecc_memcfg and ecc_memcommand registers setup for memory
Issue Command Phase AXI write. 512 byte Alligned Address
Address aligned and Start command matches nand_wr_cmd or nand_rd_cmd
No
Yes
Carry out read or write data phases
ECC Last set (and end_command_valid set for writes)?
No
Yes
Command aligns with end of block?
No
Yes
ECC jumping allowed or all block written/read
No
Yes
ECC block writes ECC values to memory, or reads previously stored ECC

Figure 2-30 Basic operation

Addressing

The ECC block supports two addressing modes. This must be set correctly for the type of memory in use, because it is used when generating addresses to move around the NAND page, and for detecting 512 byte aligned addresses.

Normal mode addressing

The normal mode, setting `ecc_ignore_add_eight` = 0, expects the first two bytes to contain just the column address bits as Table 2-30 shows.

Table 2-30 Normal mode addressing

Cycle	I/O 7	I/O 6	I/O 5	I/O 4	I/O 3	I/O 2	I/O 1	I/O 0
1st	A7	A6	A5	A4	A3	A2	A1	A0
2nd	L[a]	L[a]	L[a]	L[a]	A11[b]/L	A10	A9	A8
3rd-7th	DC	DC	DC	DC	DC	DC	DC	DC

a. These bits must be Low otherwise the behavior is undefined.
b. A11 might be present, depending on the memory width.

DC = Don't care L = LOW

This mode supports all random access, column change commands, and up to four 512 blocks. See *Address jumping* on page 2-52.

Secondary mode addressing

The second mode, setting `ecc_ignore_add_eight` = 1, supports memories with 512 bits where the address formatting is as Table 2-31 lists.

Table 2-31 Second mode addressing

Cycle	I/O 7	I/O 6	I/O 5	I/O 4	I/O 3	I/O 2	I/O 1	I/O 0
1st	A7	A6	A5	A4	A3	A2	A1	A0
2nd	DC	DC	DC	DC	DC	DC	DC	DC
3rd-7th	DC	DC	DC	DC	DC	DC	DC	DC

DC = Don't care

Note

In this mode, A8 is passed as part of the start command and is not present in the data.

In this mode, random accesses are not possible. The `nand_rd_col_change` bits in the `ecc_memcommand2` register can be used as an alternative read start command. This enables ECC calculation on just the extra block at the end of the page.

For writes, issuing a zero address cycle, pointer change command, that matches the `nand_rd_col_change` command, tells the ECC block that the next write command is to the extra bits. This only applies to the subsequent write command, even if the memory only requires one pointer access for multiple writes.

Data

When a valid start address has been sent, data can be read or written using a series of NAND data phase commands. See *NAND data phase transfers* on page 2-19. The last access must align with the end of a 512 byte block, or the extra block if it is enabled. You must set ECC Last on the last data phase access, to tell the ECC block not to expect any more data.

If an access to a different chip, is received during an ECC operation, the ECC block aborts and issues a `Data stop after incomplete block` error in the `ecc_status` register. No more ECC codes are read or written to memory.

Address jumping

To enable you to write individual 512 byte blocks, or the ECC extra block, the controller can issue address phase commands to move around the NAND page.

The `ecc_jump` setting in the status register controls how the controller jumps to the correct place in memory. It can be programmed to jump using full NAND commands, column change (random access) commands, or not to jump at all.

When using full NAND commands, the commands stored in `ecc_memcommand1` are used to control the NAND address pointer. They are also used to detect the start of a NAND read or write. When the ECC block is set to jump, using column change commands, the commands stored in `ecc_memcommand2` are used. The value used as the end command for a write access is taken directly from the previous AXI command that had the `end_command_valid` bit asserted.

The ECC values for writes are only written to memory after the end command is received. For reads, the `ecc_read_end` setting can be used to read ECC codes from memory between every block.

Address modes

This section describes the different methods used to control the address pointer, when writing ECC codes. The same methods can be applied to reads, except that end commands may be output after the address, if enabled in the `ecc_memcommand<x>` registers, but never after a data transfer.

ecc_jump = no jumping

If `ecc_jump` is set to no jumping, and not all blocks in a page are read or written, then an error is generated. However, the calculated ECC codes are available using the APB interface. If required, you can then use software to write them to memory. See Figure 2-31.

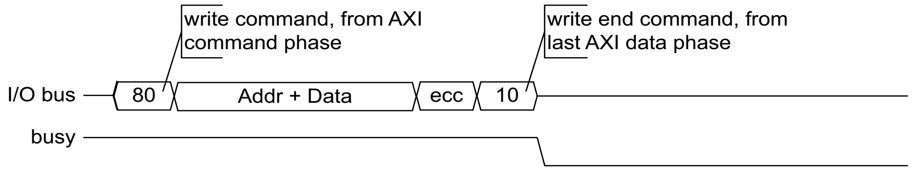

write command, from AXI command phase
write end command, from last AXI data phase
I/O bus
80
Addr + Data
ecc
10
busy

Figure 2-31 Every block is written

Note

The command values shown in these diagrams, for example 80, 10, or 15, are representative and may not match your particular NAND device.

ecc_jump = column change

If `ecc_jump` is set to column change commands, the controller issues a `col_change` command, with two address cycles. See Figure 2-32.

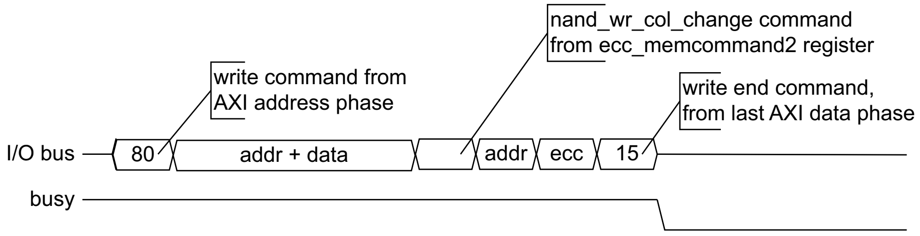

nand_wr_col_change command from ecc_memcommand2 register
write command from AXI address phase
write end command, from last AXI data phase
I/O bus
80
addr + data
addr
ecc
15
busy

Figure 2-32 Not every block written, random access

ecc_jump = full command

If a full command is used, the controller issues an entire new command phase access with the same number of address cycles as the initial write. See Figure 2-33 on page 2-54.

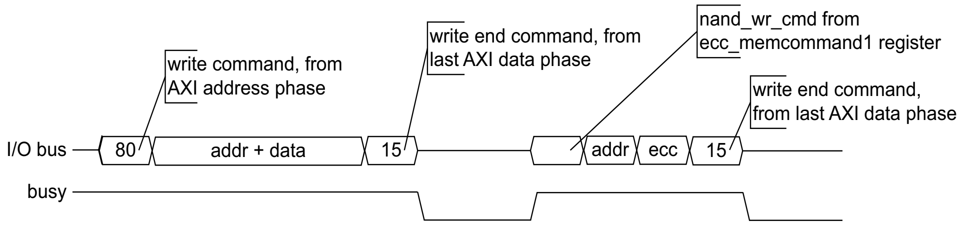

write command, from AXI address phase
write end command, from last AXI data phase
nand_wr_cmd from ecc_memcommand1 register
write end command, from last AXI data phase
I/O bus
80
addr + data
15
addr
ecc
15
busy

Figure 2-33 Full jumping

Note

- If `ecc_jump` is set to use full commands, this counts against the maximum number of program operations before a NAND page must be erased.
- If `ecc_read_end` is set to read between blocks, then each boundary must be aligned with the end of a data phase access. Otherwise, data phases accesses can cross boundaries between blocks.

ignore_add_8 and ecc_jump is not no_jump

If not all blocks are written, the controller issues a pointer change command using the value in the `nand_rd_col_change` bits of the `mem_command` register. For reads, the `nand_rd_col_change` bits are used instead of the standard read command, to access the extra bits at the end of the page. See Figure 2-34.

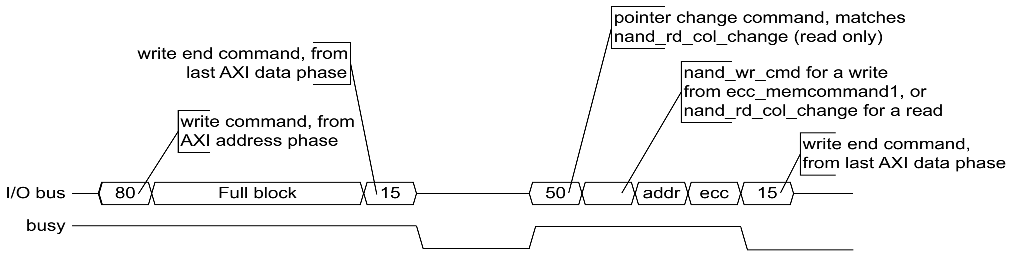

pointer change command, matches nand_rd_col_change (read only)
write end command, from last AXI data phase
nand_wr_cmd for a write from ecc_memcommand1, or nand_rd_col_change for a read
write command, from AXI address phase
write end command, from last AXI data phase
I/O bus
80
Full block
15
50
addr
ecc
15
busy

Figure 2-34 addressing mode ignore_add_8 with extra blocks

Note

After writing or reading ECC values in the secondary addressing mode, see *Secondary mode addressing* on page 2-51, the ECC block does not return the pointer to its previous state. Software might have to correct the pointer, depending on the memory and if the ECC block was forced to jump into the extra data area.

Cache mode accesses

If performing cache mode reads, the entire page must be read and ECC Last only issued on the last data_phase access of the last page. Undefined behavior occurs if you attempt to read data beyond the page size.

——— **Note** ———

- `ecc_jump` must be set to no jump to prevent the controller from attempting to move the address pointer around the cache register.
- If multiple pages are read, it is the job of software to maintain a count of the number of pages. All block valid and read flags are cleared when the first block of a new page is read.

Error Codes

The error code available from the `ecc_status` register applies to the previous ECC operation. It must only be considered valid when the ECC block is not busy.

Interrupts

Interrupts can be generated:

- when the ECC block detects an error on a read
- when an ECC code is read from memory, if the `ecc_int_read` bit is set
- when an error occurs, if the `ecc_int_abort` bit is set.

Interrupts can be cleared by:

- writing to the interrupt flag in the `ecc_status` register
- writing any value to the block register.

——— **Note** ———

To enable the external interrupt, the `ecc_intx_en` bit must be set using the `memc_cfg_set` register in PL35x.

Correcting errors

If an error occurs, `ecc_fail` for that block is set. If the error is correctable, then the `ecc_correct` flag is set and the `ecc_value` gives the location of the bit that must be corrected.

The bottom three bits give the bit number, and the remaining bits form the byte number. For example, an `ecc_value` of 0x101 implies bit 1 of byte 32 is incorrect.

Chapter 3
Programmer's Model

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

- *About the programmer's model* on page 3-2
- *Register summary* on page 3-3
- *Register descriptions* on page 3-7.

Note

See also Chapter 6 *Configurations*.

3.1 About the programmer's model

The SMC has 4KB of memory allocated to it from a base address of `0x000` to a maximum address of `0xFFF`. Figure 3-1 shows that the register map address range is split into five regions:

Memory controller configuration registers

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

Chip select configuration registers

These registers hold the operating parameters of each chip select. If the SMC is not configured to support all chip selects, the corresponding registers are not implemented.

User configuration registers

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

Integration test registers

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

PrimeCell Id registers

These registers enable the identification of system components by software.

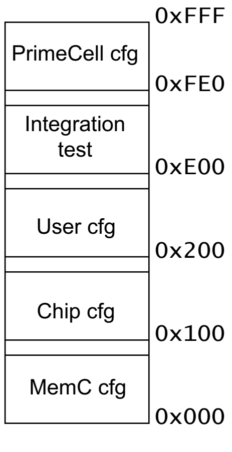


Figure 3-1 Register map

3.2 Register summary

Figure 3-2 shows the chip<*n*> configuration register map where:

n = 0 to 3.

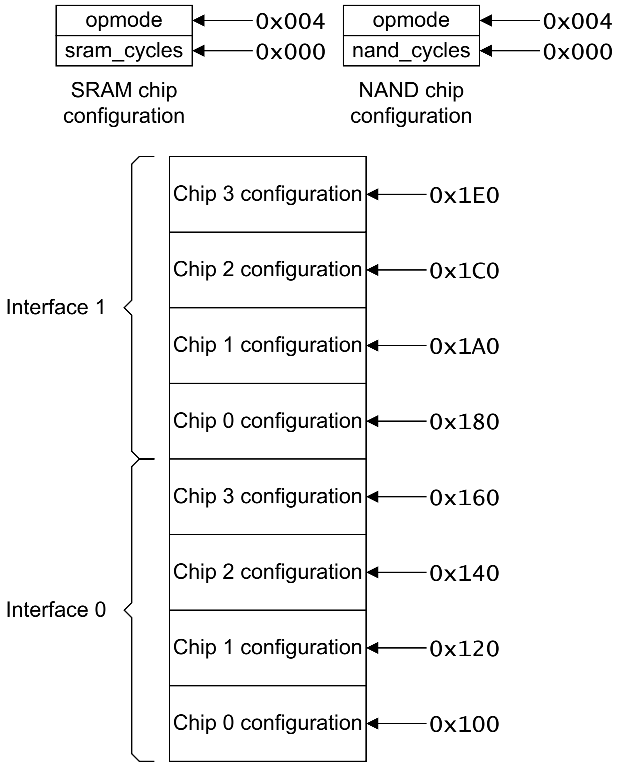


Figure 3-2 Chip<n> configuration register map

Note

Figure 3-2 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 3-3 shows the user configuration memory register map.

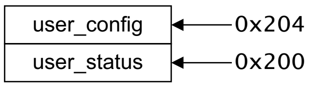


Figure 3-3 User configuration register map

Figure 3-4 shows the ECC register map.

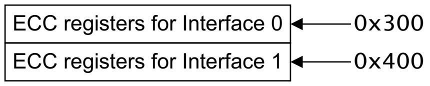


Figure 3-4 ECC register map

Figure 3-5 shows the PrimeCell configuration register map.

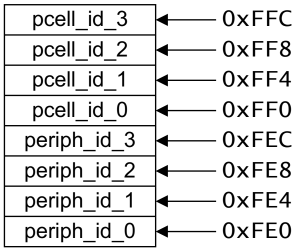


Figure 3-5 PrimeCell configuration register map

Table 3-1 lists the SMC registers.

Table 3-1 Register summary

Name	Base offset	Typ e	Reset value	Description
memc_status	`0x000`	RO	`0x00000000`	*Memory Controller Status Register* on page 3-7.
memif_cfg	`0x004`	RO	Configuration dependent	*Memory Interface Configuration Register* on page 3-8.
memc_cfg_set	`0x008`	WO	N/A	*Set Configuration Register* on page 3-10.
memc_cfg_clr	`0x00C`	WO	N/A	*Clear Configuration Register* on page 3-11.
direct_cmd	`0x010`	WO	N/A	*Direct Command Register* on page 3-12.
set_cycles	`0x014`	WO	N/A	*set_cycles Register* on page 3-14.
set_opmode	`0x018`	WO	N/A	*set_opmode Register* on page 3-15.
refresh_period_0	`0x020`	RW	`0x00000000`	*refresh_period_0 Register* on page 3-18.
refresh_period_1	`0x024`	RW	`0x00000000`	*refresh_period_1 Register* on page 3-19.

Table 3-1 Register summary (continued)

Name	Base offset	Typ e	Reset value	Description
sram_cycles<x>_<n>	`0x000` + chip configuration base address	RO	`0x0002B3CC`	sram_cycles configuration where: x = interface 0 or 1 n = chip select 0 to 3. See *sram_cycles Register* on page 3-20.
nand_cycles<x>_<n>	`0x000` + chip configuration base address	RO	`0x00092AFF`	nand_cycles configuration where: x = interface 0 or 1 n = chip select 0 to 3. See *nand_cycles Register* on page 3-21.
opmode<x>_<n>	`0x004` + chip configuration base address	RO	Configuration dependent	opmode configuration where: x = interface 0 or 1 n = chip select 0 to 3. See *opmode Register* on page 3-22.
user_status	`0x200`	RO	Configuration dependent	*user_status Register* on page 3-25.
user_config	`0x204`	WO	Configuration dependent	*user_config Register* on page 3-25.
ecc_status	`0x000` + ECC base address	RO	`0x00000000`	*ecc_status Register* on page 3-26.
ecc_memcfg	`0x004` + ECC base address	RW	`0x00000043`	*ecc_memcfg Register* on page 3-27.
ecc_memcommand1	`0x008` + ECC base address	RW	`0x01300080`	*ecc_memcommand1 Register* on page 3-29.
ecc_memcommand2	`0x00C` + ECC base address	RW	`0x01E00585`	*ecc_memcommand2 Register* on page 3-30.
ecc_addr0	`0x010` + ECC base address	RO	`0x00000000`	*ecc_addr0 Register* on page 3-31.
ecc_addr1	`0x014` + ECC base address	RO	`0x00000000`	*ecc_addr1 Register* on page 3-31.
ecc_value0 - ecc_value4	(`0x018`-`0x024`) + ECC base address	RO	`0x00000000`	*ecc_value(0 ... 4) Registers* on page 3-32.

Table 3-1 Register summary (continued)

Name	Base offset	Typ e	Reset value	Description
integration_test	`0xE00`	RW	`0x00000000`	Integration Configuration Register on page 4-2.
periph_id_n	`0xFE0-0xFEC`	RO	Configuration dependent	*Peripheral Identification Registers 0-3* on page 3-33.
pcell_id_n	`0xFF0-0xFFC`	RO	Configuration dependent	*PrimeCell Identification Registers 0-3* on page 3-36.

3.3 Register descriptions

This section describes the SMC registers.

3.3.1 Memory Controller Status Register

The read-only memc_status Register provides information on the configuration of the SMC and also the current state of the SMC. You cannot read this register in the Reset state. Figure 3-6 shows the register bit assignments.

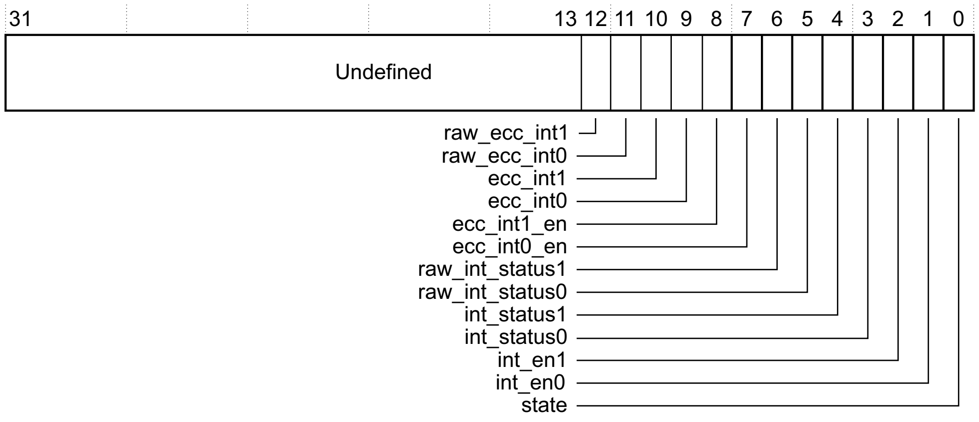


Figure 3-6 memc_status Register bit assignments

Table 3-2 lists the register bit assignments.

Table 3-2 memc_status Register bit assignments

Bits	Name	Function
[31:13]	-	Reserved, read undefined
[12]	raw_ecc_int1	Raw status of the ECC interrupt on interface 1
[11]	raw_ecc_int0	Raw status of the ECC interrupt on interface 0
[10	ecc_int1	Status of the ECC interrupt on interface 1
[9]	ecc_int0	Status of the ECC interrupt on interface 0
[8]	ecc_int1_en	Status of the ECC interrupt enable on interface 1
[7]	ecc_int0_en	Status of the ECC interrupt enable on interface 0
[6]	raw_int_status1	Current raw interrupt status for interface 1

Table 3-2 memc_status Register bit assignments (continued)

Bits	Name	Function
[5]	raw_int_status0	Current raw interrupt status for interface 0
[4]	int_status1	Current interrupt status for interface 1
[3]	int_status0	Current interrupt status for interface 0
[2]	int_en1	Status of memory interface 1 interrupt enable
[1]	int_en0	Status of memory interface 0 interrupt enable
[0]	state	Operating state of the controller: 0 = SMC is in the ready state 1 = SMC is in the low-power state.

3.3.2 Memory Interface Configuration Register

The read-only memif_cfg Register provides information on the configuration of the memory interface. You cannot read this register in the Reset state. Figure 3-7 shows the register bit assignments.

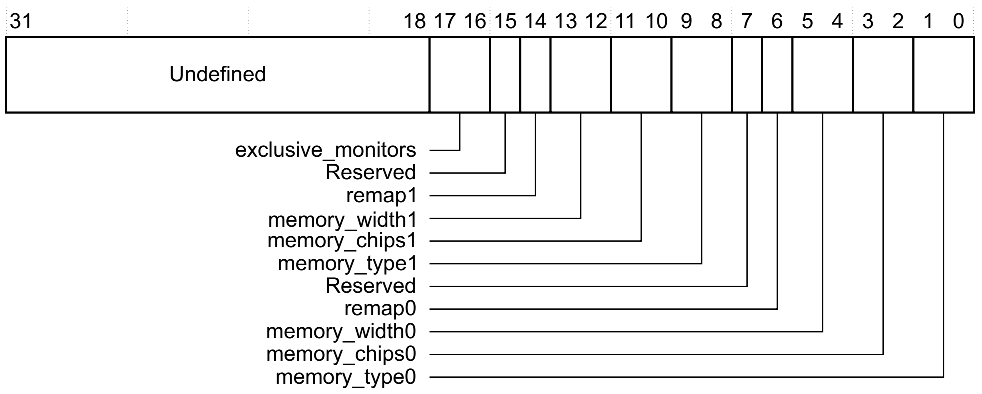


Figure 3-7 memif_cfg Register bit assignments

Table 3-3 lists the register bit assignments.

Table 3-3 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 monitor
		b10 = 2 monitors
		b11 = 4 monitors.
		See *Exclusive accesses* on page 2-15.

[15]	-	Reserved.
[14]	remap1	Returns the value of the **remap_1** input.

[13:12]	memory_width1	Returns the maximum width of the SMC memory data bus for interface 1:
		b00 = 8 bits
		b01 = 16 bits
		b10 = 32 bits
		b11 = reserved.

[11:10]	memory_chips1	Returns the number of different chip selects that the memory interface 1 supports:
		b00 = 1 chip
		b01 = 2 chips
		b10 = 3 chips
		b11 = 4 chips.

[9:8]	memory_type1	Returns the memory interface 1 type:
		b00 = Configuration does not include this memory interface
		b01 = SRAM non-multiplexed
		b10 = NAND
		b11 = SRAM multiplexed.
		If b00, the remaining bit slices for memory interface 1 are always read as 0.

[7]	-	Reserved.
[6]	remap0	Returns the value of the **remap_0** input.

Table 3-3 memif_cfg Register bit assignments (continued)

Bits	Name	Function

[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.

[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 non-multiplexed
		b10 = NAND
		b11 = SRAM multiplexed.

3.3.3 Set Configuration Register

The write-only memc_cfg_set Register enables the memory controller to be changed to low-power state, and interrupts enabled. You cannot write to this register in the Reset state. Figure 3-8 shows the register bit assignments.

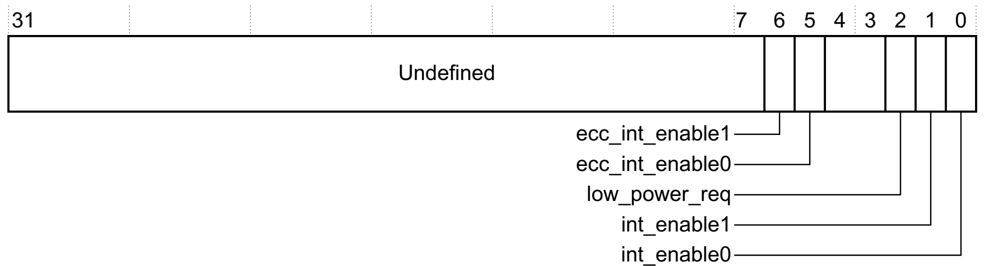


Figure 3-8 memc_cfg_set Register bit assignments

Table 3-4 lists the register bit assignments.

Table 3-4 memc_cfg_set Register bit assignments

Bits	Name	Function
[31:7]	-	Reserved, write as zero.
[6]	ecc_int_enable1	0 = No effect 1 = ECC interrupt enable, memory interface 1.
[5]	ecc_int_enable0	0 = No effect 1 = ECC interrupt enable, memory interface 0.
[4:3]	-	Reserved, write as zero.
[2]	low_power_req	0 = No effect 1 = Request the SMC to enter low-power state when it next becomes idle.
[1]	int_enable1	0 = No effect 1 = Interrupt enable, memory interface 1.
[0]	int_enable0	0 = No effect 1 = Interrupt enable, memory interface 0.

3.3.4 Clear Configuration Register

The write-only memc_cfg_clr Register enables the memory controller to be moved out of the low-power state, and the interrupts disabled. You cannot write to this register in the Reset state. Figure 3-9 shows the register bit assignments.

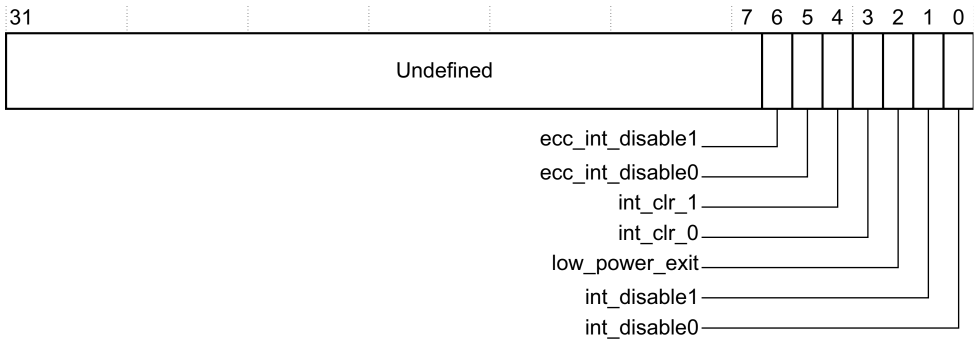


Figure 3-9 memc_cfg_clr Register bit assignments

Table 3-5 lists the register bit assignments.

Table 3-5 memc_cfg_clr Register bit assignments

Bits	Name	Function
[31:7]	-	Reserved, write as zero.
[6]	ecc_int_disable0	0 = No effect 1 = Disable ECC Interrupt on IF1.
[5]	ecc_int_disable0	0 = No effect 1 = Disable ECC Interrupt on IF0.
[4]	int_clr_1	0 = No effect 1 = Clear SMC Interrupt 1 as an alternative to an AXI read.
[3]	int_clr_0	0 = No effect 1 = Clear SMC Interrupt 0 (as an alternative to an AXI read).
[2]	low_power_exit	0 = No effect 1 = Request the SMC to exit low-power state.
[1]	int_disable1	0 = No effect 1 = Interrupt disable, memory interface 1.
[0]	int_disable0	0 = No effect 1 = Interrupt disable, memory interface 0.

3.3.5 Direct Command Register

The write-only 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.

You cannot write to this register in either the Reset or low-power states. Figure 3-10 on page 3-13 shows the register bit assignments.

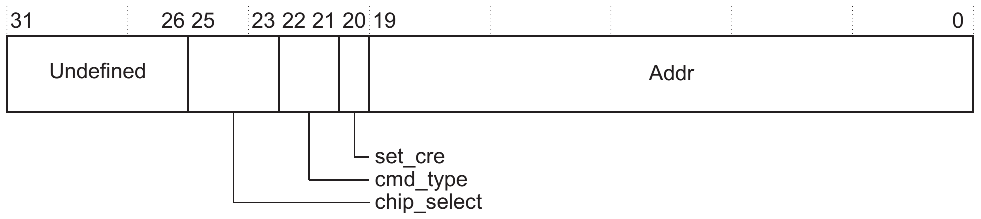


Figure 3-10 direct_cmd Register bit assignments

Table 3-6 lists the register bit assignments.

Table 3-6 direct_cmd 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 1-4 on interface 0
		b100-b111 = Chip selects 1-4 on interface 1.

[22:21]	cmd_type	Determines the current command. The encoding is:
		b00 = UpdateRegs and AXI command
		b01 = ModeReg access
		b10 = UpdateRegs
		b11 = ModeReg and UpdateRegs.

[20]	set_cre	Maps to the configuration register enable signal, **cre**, when a ModeReg command is issued. The encoding is: 0 = **cre** is LOW 1 = **cre** is HIGH when ModeReg write occurs.
[19:0]	addr	When cmd_type = UpdateRegs and AXI command then: • bits [15:0] are used to match **wdata[15:0]** • bits [19:16] are reserved. Write as zero. When cmd_type = ModeReg access or ModeReg and UpdateRegs, these bits map to the external memory address bits [19:0]. When cmd_type = UpdateRegs, these bits are reserved. Write as zero.

The command types referenced in Table 3-6 are:

`ModeReg` programs the memory.

UpdateRegs updates the PL350 holding registers, opmode and *_cycles.

ModeReg and UpdateRegs, used together, update the PL350 holding registers and the memory devices, at the same time. The combined ModeReg and UpdateRegs command aids in programming different memory timing values, while an access to memory is still ongoing. This enables code to be executed from 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, UpdateRegs, from the holding registers, with the dispatch of the memory configuration register write. You can program the relevant command in the register, depending on the simultaneous operations that you want to perform.

The UpdateRegs and AXI command is where the PL350 holding registers are updated. The address match value, bits [19:0] in direct_cmd register, is compared against the AXI command to determine when to program the memory device. This is required for devices like NOR Flash devices, where there is a sequence of commands. Then the write data bus of the AXI can be used to signal when the last AXI command transfer is completed, and it is safe for PL350 to update the chip configuration registers.

3.3.6 set_cycles Register

This write-only register is the holding register for the sram<n>_cycles and nand<n>_cycles Registers. This register enables you to set the time interval for holding registers before writing to certain memory manager specific registers. You cannot write to this register in either the Reset or low-power states. Figure 3-11 shows the register bit assignments.

Note

Table 3-7 on page 3-15 describes register holding, see *Memory manager operation* on page 2-26 for more information.

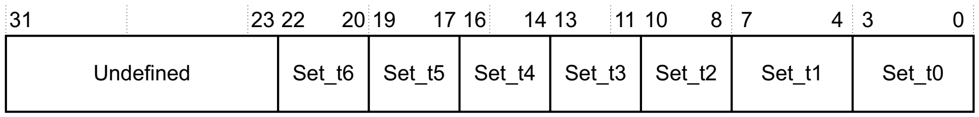


Figure 3-11 set_cycles Register bit assignments

Table 3-7 lists the register bit assignments.

Table 3-7 set_cycles Register bit assignments

Bits	Name	Function
[31:23]	-	Reserved, undefined, write as zero
[22:20]	Set_t6	Holding register for value to be written to the specific NAND chip Register t_{RR} field
[19:17]	Set_t5	Holding register for value to be written to the specific chip Register t_{TR} or t_{AR} fields
[16:14]	Set_t4	Holding register for value to be written to the specific chip Register t_{PC} or t_{CLR} fields
[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} or t_{REA} fields
[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

3.3.7 set_opmode Register

This write-only register is the holding register for the opmode<x>_<n> working registers. You cannot write to it in either the Reset or low-power states. Figure 3-12 on page 3-16 shows the register bit assignments.

Note

Table 3-8 on page 3-16 describes register holding, see *Memory manager operation* on page 2-26 for more information.

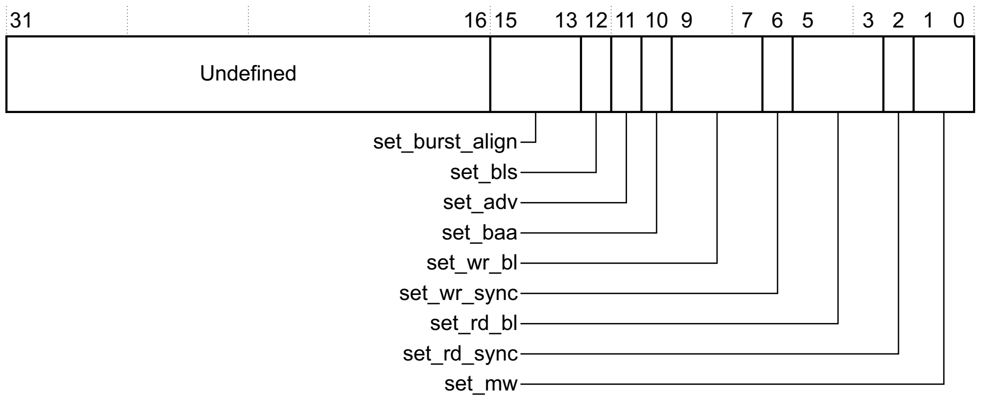


Figure 3-12 set_opmode Register bit assignments

Table 3-8 lists the register bit assignments.

Table 3-8 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. When you configure the controller to perform synchronous transfers[a], these bits determine whether memory bursts are split on memory burst boundaries: b000 = bursts can cross any address boundary b001 = burst split on memory burst boundary, that is, 32 beats for continuous b010 = burst split on 64 beat boundary b011 = burst split on 128 beat boundary b100 = burst split on 256 beat boundary b101 - b111 = reserved. For a NAND memory interface these bits are reserved, and written as zero.
[12]	set_bls	Holding register for value to be written to the specific SRAM chip opmode Register *byte lane strobe* (bls) field. This bit affects the assertion of the byte-lane strobe outputs. 0 = bls timing equals chip select timing. This is the default setting. 1 = bls timing equals **we_n** timing. This setting is used for eight memories that have no **bls_n** inputs. In this case, the **bls_n** output of the memory controller is connected to the **we_n** memory input. For a NAND memory interface this bit is reserved, and written as zero.

Table 3-8 set_opmode Register bit assignments (continued)

Bits	Name	Function
[11]	set_adv	Holding register for value to be written to the specific SRAM chip opmode Register *address valid (*adv) field. The memory uses the address advance signal **adv_n** when set. For a NAND memory interface this bit is reserved, and written as zero.
[10]	set_baa	Holding register for value to be written to the specific SRAM chip opmode Register *burst address advance* (baa) field. The memory uses the **baa_n** signal when set. For a NAND memory interface this bit is reserved, and written as zero.

[9:7]	set_wr_bl	Holding register for value to be written to the specific SRAM chip 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.
		For a NAND memory interface these bits are reserved, and written as zero.

[6]	set_wr_sync	Holding register for value to be written to the specific SRAM chip opmode Register wr_sync field. The memory writes are synchronous when set. For a NAND memory interface this bit is reserved, and written as zero.

Table 3-8 set_opmode Register bit assignments (continued)

Bits	Name	Function

[5:3]	set_rd_bl	Holding register for value to be written to the specific SRAM chip 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.
		For a NAND memory interface these bits are reserved, and written as zero.

[2]	set_rd_sync	Holding register before being written to the specific SRAM chip opmode Register rd_sync field. Memory in sync mode when set. For a NAND memory interface this bit is reserved, and written as zero.

[1:0]	set_mw	Holding register for value to be written to the specific SRAM chip opmode Register *memory width* (mw) field.
		Encodes the memory data bus width:
		b00 = 8 bits[b]
		b01 = 16 bits[b]
		b10 = 32 bits
		b11 = reserved.
		You can program this to the configured width, or half that width. See *Memory Interface Configuration Register* on page 3-8.

a. For asynchronous transfers:
- the controller always aligns read bursts to the memory burst boundary, when set_rd_sync = 0
- the controller always aligns write bursts to the memory burst boundary, when set_wr_sync = 0.

b. For a NAND interface, only 8-bit and 16-bit are valid settings.

3.3.8 refresh_period_0 Register

The read/write refresh_period_0 Register enables the controller to insert idle cycles during consecutive bursts, that enables the PSRAM devices on memory interface 0, to initiate a refresh cycle. You cannot access this register in either the Reset or low-power states. Figure 3-13 on page 3-19 shows the register bit assignments.

Note

You can only access this register when you are using an SRAM memory interface.

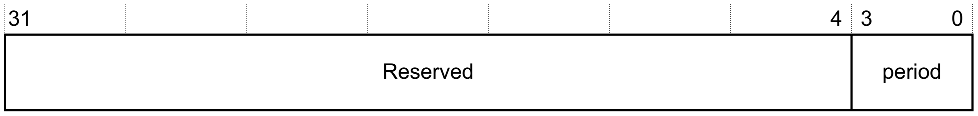


Figure 3-13 refresh_period_0 Register bit assignments

Table 3-9 lists the register bit assignments.

Table 3-9 refresh_period_0 Register bit assignments

Bits	Name	Function
[31:4]	-	Reserved, read undefined.
[3:0]	period	Sets the number of consecutive memory bursts[a] that are permitted, prior to the controller 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.

a. In continuous mode the memory bursts are limited to 32 beats.

3.3.9 refresh_period_1 Register

The read/write refresh_period_1 Register enables the controller to insert idle cycles during consecutive bursts, that enables the PSRAM devices on memory interface 1, to initiate a refresh cycle. Figure 3-14 shows the register bit assignments.

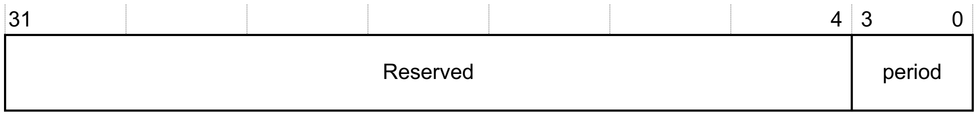


Figure 3-14 refresh_period_1 Register bit assignments

Note

You can only access this register when you are using an SRAM memory interface.

Table 3-10 lists the register bit assignments.

Table 3-10 refresh_period_1 Register bit assignments

Bits	Name	Function
[31:4]	-	Reserved, read undefined.
[3:0]	period	Sets the number of consecutive memory bursts[a] that are permitted, prior to the controller 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.

a. In continuous mode the memory bursts are limited to 32 beats.

3.3.10 sram_cycles Register

There is an instance of this register for each SRAM chip supported. You cannot read the read-only sram_cycles Register in the Reset state. Figure 3-15 shows the register bit assignments.

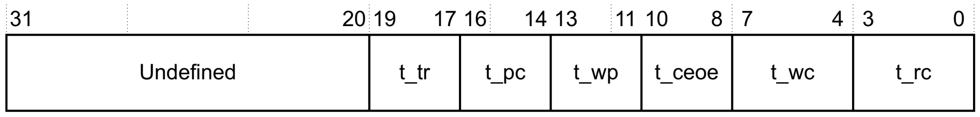


Figure 3-15 sram_cycles Register bit assignments

Table 3-11 lists the register bit assignments.

Table 3-11 sram_cycles Register bit assignments

Bits	Name	Function
[31:20]	-	Reserved, read undefined.
[19:17]	t_tr	Turnaround time for SRAM chip configurations. Minimum permitted value = 1.
[16:14]	t_pc	Page cycle time for SRAM chip configurations. Minimum permitted value = 1.
[13:11]	t_wp	**we_n** assertion delay. Minimum permitted value = 1.
[10:8]	t_ceoe	**oe_n** assertion delay for SRAM chip configurations. Minimum permitted value = 1.
[7:4]	t_wc	Write cycle time. Minimum permitted value = 2.
[3:0]	t_rc	Read cycle time. Minimum permitted value = 2.

3.3.11 nand_cycles Register

There is an instance of this register for each NAND chip supported. You cannot read the read-only nand_cycles Register in the Reset state. Figure 3-16 shows the register bit assignments.

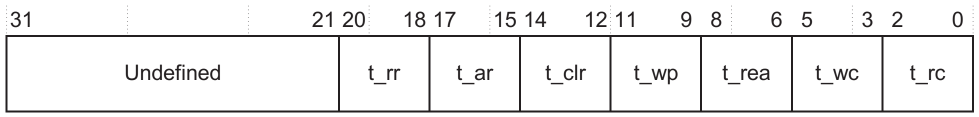


Figure 3-16 nand_cycles Register bit assignments

Table 3-12 lists the register bit assignments.

Table 3-12 nand_cycles Register bit assignments

Bits	Name	Function
[31:21]	-	Reserved, read undefined.
[20:18]	t_rr	**busy** to **re_n** for NAND chip configurations. Minimum permitted value = 0.
[17:15]	t_ar	ID read time for NAND chip configurations. Minimum permitted value = 0.
[14:12]	t_clr	Status read time for NAND chip configurations. Minimum permitted value = 0.
[11:9]	t_wp	**we_n** de-assertion delay. Minimum permitted value = 1.
[8:6]	t_rea	**re_n** assertion delay for NAND chip configurations. Minimum permitted value = 1.
[5:3]	t_wc	Write cycle time. Minimum permitted value = 2.
[2:0]	t_rc	Read cycle time. Minimum permitted value = 2.

3.3.12 opmode Register

There is an instance of the opmode Register for each chip supported. This register is read-only and you cannot read it 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. Tie-offs at the top-level set the address_match and address_mask reset values. The reset value of burst_align is 3'b001, and all other fields reset to 0.

Figure 3-17 on page 3-23 shows the register bit assignments.

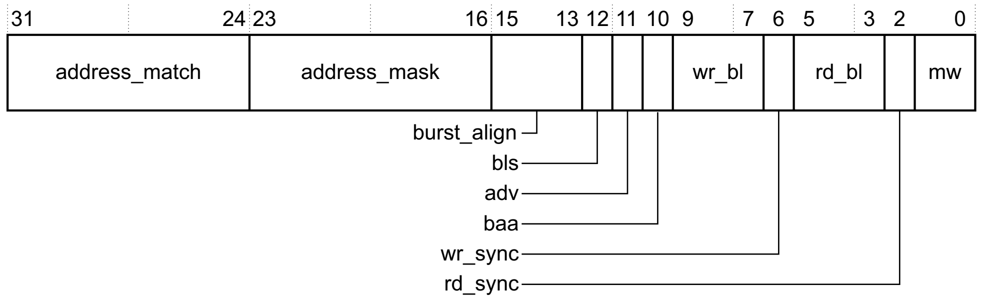


Figure 3-17 opmode Register bit assignments

Table 3-13 lists the register bit assignments.

Table 3-13 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	When you configure the controller to perform synchronous transfers[a], these bits determine whether memory bursts are split on memory burst boundaries:
		b000 = bursts can cross any address boundary
		b001 = burst split on memory burst boundary, that is, 32 beats for continuous
		b010 = burst split on 64 beat boundary
		b011 = burst split on 128 beat boundary
		b100 = burst split on 256 beat boundary
		b101-b111 = reserved.
		For a NAND memory interface these bits are reserved.

[12]	bls	This bit affects the assertion of the byte-lane strobe outputs: 0 = bls timing equals chip select timing. This is the default setting. 1 = bls timing equals **we_n** timing. This setting is used for 8-bit memories that have no bls inputs. In this case, the **bls_n** output of the memory controller is connected to the **we_n** memory input. For a NAND memory interface this bit is reserved.
[11]	adv	The memory uses the address advance signal, **adv_n**, when set. For a NAND memory interface this bit is reserved.

Table 3-13 opmode Register bit assignments (continued)

Bits	Name	Function
[10]	baa	The memory uses the burst address advance signal, **baa_n**, when set. For a NAND memory interface this bit is reserved.

[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.
		For a NAND memory interface these bits are reserved.

[6]	wr_sync	When set, the memory operates in write sync mode. For a NAND memory interface this bit is reserved.

[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.
		For a NAND memory interface these bits are reserved.

[2]	rd_sync	When set, the memory operates in read sync mode. For a NAND memory interface this bit is reserved.

[1:0]	mw	Determines the SMC memory data bus width:
		b00 = 8 bits
		b01 = 16 bits
		b10 = 32 bits
		b11 = reserved.

a. For asynchronous transfers:
- the controller always aligns read bursts to the memory burst boundary, when rd_sync = 0
- the controller always aligns write bursts to the memory burst boundary, when wr_sync = 0.

3.3.13 user_status Register

This is the general purpose I/O read-only register that returns the value of **user_status[7:0]**. You can read the user_status Register in all states. Figure 3-18 shows the register bit assignments.

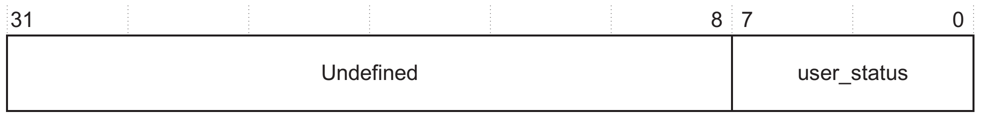


Figure 3-18 user_status Register bit assignments

Table 3-14 lists the register bit assignments.

Table 3-14 user_status Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	user_status	This value returns the state of the **user_status[7:0]** inputs

3.3.14 user_config Register

This is a general purpose write-only I/O register. The value of this register is output on **user_config[7:0]**. You can write to the user_config Register in all states. Figure 3-19 shows the register bit assignments.

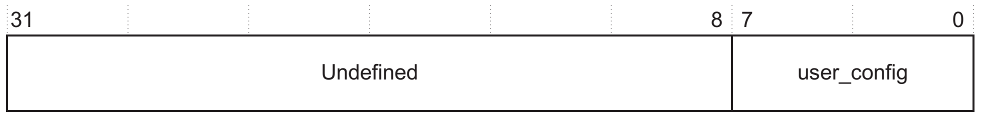


Figure 3-19 user_config Register bit assignments

Table 3-15 lists the register bit assignments.

Table 3-15 user_config Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, undefined, write as zero
[7:0]	user_config	This value sets the state of the **user_config[7:0]** outputs

3.3.15 ecc_status Register

ECC Status is a general purpose read-only I/O register. It contains status information for the ECC.

Although this is a read-only register, the bottom five bits can be written to clear the corresponding interrupts.

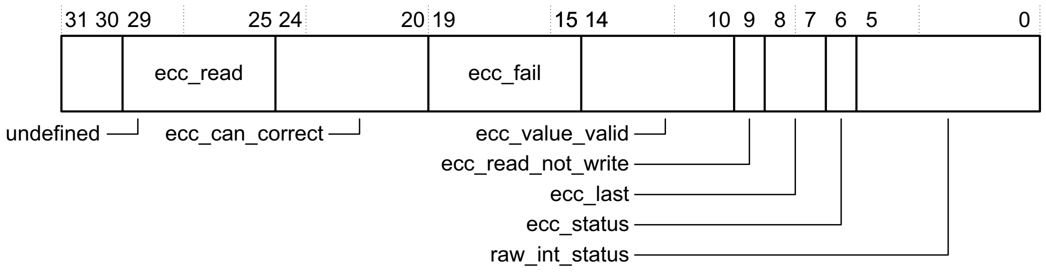


Figure 3-20 ecc_status Register bit assignments

Table 3-16 lists the register bit assignments.

Table 3-16 ecc_status Register bit assignments

Bits	Name	Function
[31:29]	-	Reserved, read undefined.
[29:25]	ecc_read	Read flags for ECC blocks. Indicates whether the stored ECC value for each block has been read from memory 0 = not read 1 = read.
[24:20]	ecc_can_correct	Correctable flag for each ECC block. Indicates if the detected error is correctable. 0 = not correctable 1 = correctable.
[19:15]	ecc_fail	Pass/fail flag for each ECC block.
[14:10]	ecc_value_valid	Valid_flag for each ECC block.
[9]	ecc_read_not_write	0 = write 1 = read.

Table 3-16 ecc_status Register bit assignments (continued)

Bits	Name	Function
[8:7]	ecc_last_status	00 = Completed successfully 01 = Unaligned Address, or `out-of-range` 10 = Data stop after incomplete block 11 = Data stopped but not values not read/written because of `ecc_jump` value. **Note** The ecc_last_status bit is only updated at the completion of an ECC calculation.
[6]	ecc_status	Describes the status of the ECC block 0 = idle 1 = busy.
[5:0]	raw_int_status	Flags: 0 = Block 0 1 = Block 1 2 = Block 2 3 = Block 3 4 = Extra block_flag (if used) 5 = Abort interrupt flag. **Note** Writing to the appropriate bit clears the interrupt.

3.3.16 ecc_memcfg Register

Memory Configuration Register is a general purpose read-write I/O register. It contains information on the structure of the memory. See *Error Correction Code operation* on page 2-48 for more details about the settings in this register.

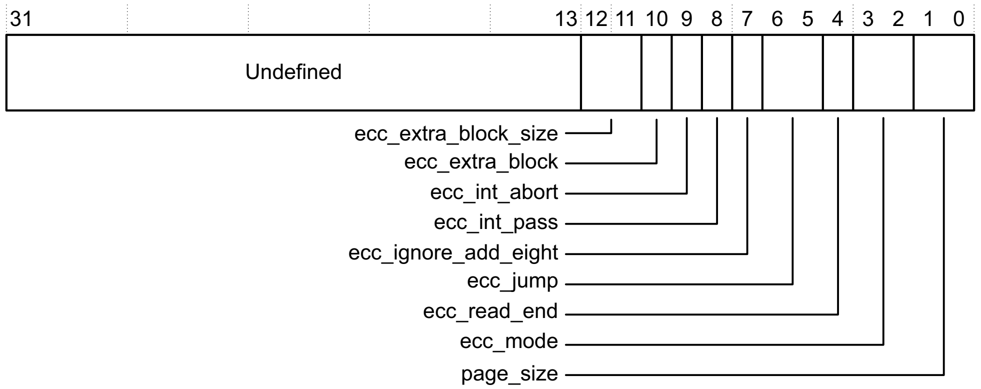


Figure 3-21 ecc_memcfg Register bit assignments

Table 3-17 lists the register bit assignments.

Table 3-17 ecc_memcfg Register bit assignments

Bits	Name	Function
[31:13]	-	Reserved, read undefined
[12:11]	ecc_extra_block_size	The size of the extra block in memory after the last 512 block: 00 = 4 bytes 01 = 8 bytes 10 = 16 bytes 11 =32 bytes. **Note** These bits are only present if the ecc_extra_block configuration option is set.
[10]	ecc_extra_block	If configured, this enables a small block for extra information after the last 512 bytes block in the page. **Note** These bits are only present if the ecc_extra_block configuration option is set.
[9]	ecc_int_abort	Interrupt on ECC abort
[8]	ecc_int_pass	Interrupt when a correct ECC value is read from memory

Table 3-17 ecc_memcfg Register bit assignments (continued)

Bits	Name	Function
[7]	ecc_ignore_add_eight	This bit is used to indicate if A8 is output with the address, required to find the aligned start of blocks: 0 =A8 is output 1 =A8 is not output. See *Secondary mode addressing* on page 2-51.
[6:5]	ecc_jump	Indicates that the memory supports col change address commands: 00 =No jumping. Reads and writes only occur at end of page 01 =Jump using column change commands 10 =jump using full command 11 =Reserved.
[4]	ecc_read_end	Indicates when ECC values are read from memory: 0 = the ECC value for a block must be read immediately after the block. Data access must stop on a 512 byte boundary. 1 = ECC values for all blocks are read at the end of the page.
[3:2]	ecc_mode	This specifies the mode of the ECC block: 00 = bypassed 01 = ECC values are calculated and made available on the apb interface. But they are not read to or written from memory. 10 = ECC values and calculated and read/written to memory. For a read, the ECC value is checked and the result of the check is made available on the APB interface. 11 = Reserved.
[1:0]	page_size	The number of 512 byte blocks in a page: 00 = No 512 byte blocks. Reserved if an ecc_extra_block is not configured and enabled 01 = One 512 byte block 10 = Two 512 byte blocks 11 = Four 512 byte blocks.

Note

You must not write to this register while the ECC block is busy. You can read the current ECC block status from the `ecc_status` reg.

3.3.17 ecc_memcommand1 Register

Memory Commands Register 1 is a general purpose read-write I/O register.

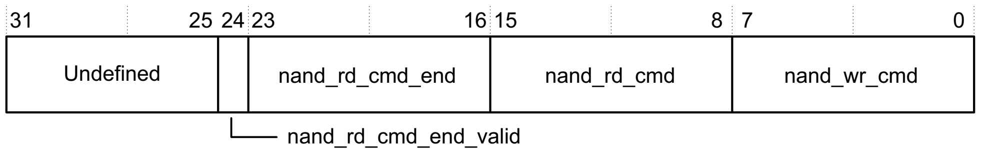


Figure 3-22 ecc_memcommand1 Register bit assignments

Table 3-18 lists the register bit assignments.

Table 3-18 ecc_memcommand1 Register bit assignments

Bits	Name	Function
[31:25]	-	Reserved, undefined, write as zero
[24]	nand_rd_cmd_end_valid	Use end command
[23:16]	nand_rd_cmd_end	The NAND command used to initiate a write (`0x30`)
[15:8]	nand_rd_cmd	The NAND command used to initiate a read (`0x00`)
[7:0]	nand_wr_cmd	The NAND command used to initiate a write (`0x80`)

3.3.18 ecc_memcommand2 Register

Memory Commands Register 2 is a general purpose read-write I/O register. This must not be changed for ONFI 1.0 compliant devices.

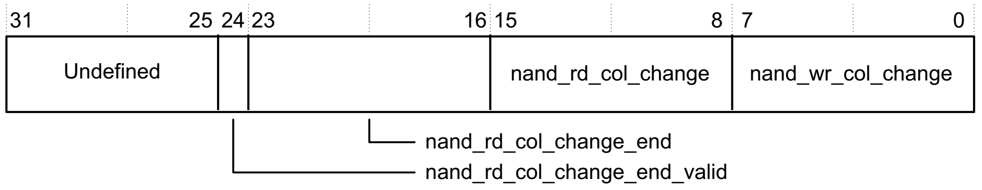


Figure 3-23 ecc_memcommand2 Register bit assignments

Table 3-19 lists the register bit assignments.

Table 3-19 ecc_memcommand2 Register bit assignments

Bits	Name	Function
[31:25]	-	Reserved, undefined, write as zero
[24]	nand_rd_col_change_end_valid	Use end command
[23:16]	nand_rd_col_change_end	The NAND command used to initiate a write
[15:8]	nand_rd_col_change	The NAND command used to initiate a read or Spare bits pointer command
[7:0]	nand_wr_col_change	The NAND command used to initiate a write

3.3.19 ecc_addr0 Register

ECC Address 0 is a general purpose read-only I/O register. It contains the lower 32 bits of the ECC Address.

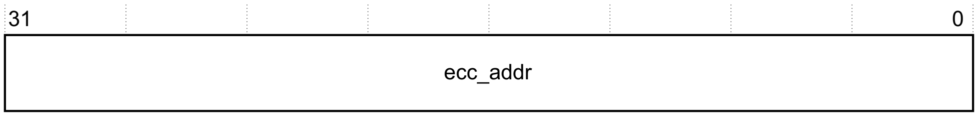


Figure 3-24 ecc_addr0 Register bit assignments

Table 3-20 lists the register bit assignments.

Table 3-20 ecc_addr0 Register bit assignments

Bits	Name	Function
[31:0]	ecc_addr	Address bits 31 to 0

3.3.20 ecc_addr1 Register

ECC Address 1 is a general purpose read-only I/O register. It contains the upper 24 bits of the ECC Address.

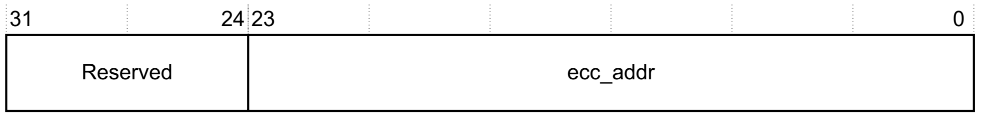


Figure 3-25 ecc_addr1 Register bit assignments

Table 3-21 lists the register bit assignments.

Table 3-21 ecc_addr1 Register bit assignments

Bits	Name	Function
[31:24]	-	Reserved, read undefined
[23:0]	ecc_addr	Address bits 55 to 32

3.3.21 ecc_value(0 ... 4) Registers

The five ecc_value registers are general purpose read-only I/O registers. They contain block information for the ECC.

Although these are read-only registers, you can write to the registers to clear the interrupt.

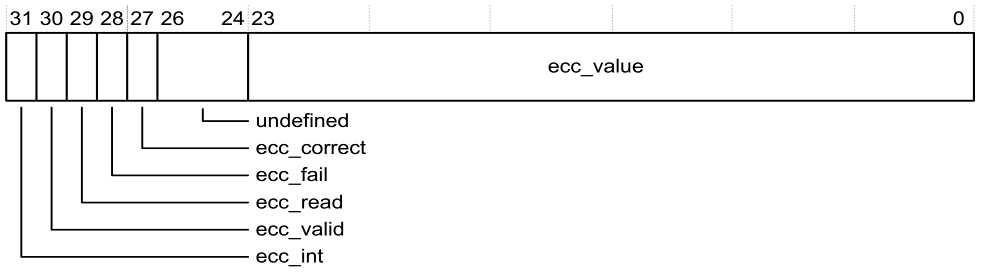


Figure 3-26 ecc_value Register bit assignments

Table 3-22 lists the register bit assignments.

Table 3-22 ecc_value Register bit assignments

Bits	Name	Function
[31]	ecc_int	Interrupt flag for this value
[30]	ecc_valid	Indicates if this value is valid
[29]	ecc_read	Indicates if the ECC value has been read from memory
[28]	ecc_fail	Indicates if this value has failed
[27]	ecc_correct	Indicates if this block is correctable
[26:24]	-	Reserved, read undefined
[23:0]	ecc_value	ECC value of check result for block, depending on ECC configuration

3.3.22 Peripheral Identification Registers 0-3

The 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. An external master reads them to determine the version of the device. None of the registers 0-3 can be read in the Reset state.

Table 3-23 lists the register bit assignments.

Table 3-23 periph_id Register bit assignments

Bits	Name	Description
[31:25]	-	Reserved, read undefined.
[24]	integration_cfg	Configuration options are peripheral-specific. See *Peripheral Identification Register 3* on page 3-36.
[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 numbers for the SMC are: `0x351` for SMC (PL351) `0x352` for SMC (PL352) `0x353` for SMC (PL353) `0x354` for SMC (PL354).

Figure 3-27 shows the correspondence between bits of the periph_id registers and the conceptual 32-bit Peripheral ID Register.

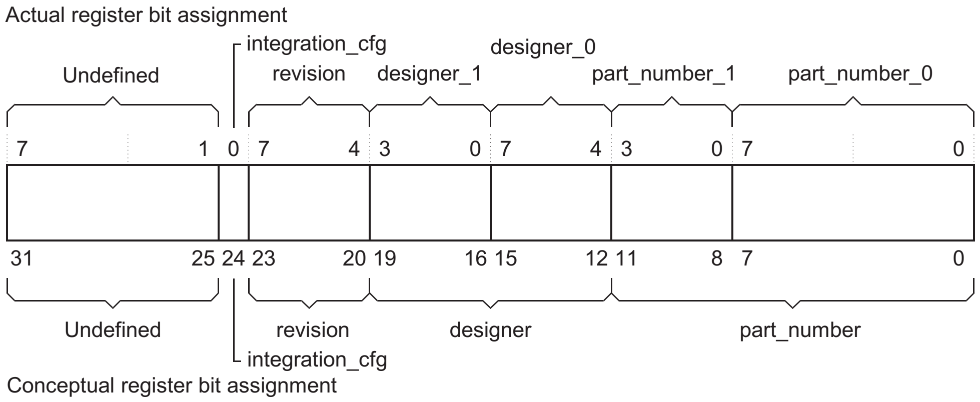


Figure 3-27 periph_id Register bit assignments

The following subsections describe the periph_id Registers:

- *Peripheral Identification Register 0*
- *Peripheral Identification Register 1* on page 3-35
- *Peripheral Identification Register 2* on page 3-35
- *Peripheral Identification Register 3* on page 3-36.

Peripheral Identification Register 0

The periph_id_0 Register is hard-coded and the fields within the register determine the reset value. Table 3-24 lists the register bit assignments.

Table 3-24 periph_id_0 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	part_number_0	These bits read back as `0x5x`, see Table 3-23 on page 3-33

Peripheral Identification Register 1

The periph_id_1 Register is hard-coded and the fields within the register determine the reset value. Table 3-25 lists the register bit assignments.

Table 3-25 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`

Peripheral Identification Register 2

The periph_id_2 Register is hard-coded and the fields within the register determine the reset value. Table 3-26 lists the register bit assignments.

Table 3-26 periph_id_2 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:4]	revision	These bits read back as: • `0x1` for r1p0 • `0x2` for1p1 • `0x3` for r1p2 • `0x4` for r2p0.
[3:0]	designer_1	These bits read back as `0x4`

Peripheral Identification Register 3

The periph_id_3 Register is hard-coded and the fields within the register determine the reset value. Table 3-27 lists the register bit assignments.

Table 3-27 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.

3.3.23 PrimeCell Identification Registers 0-3

The 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 pcell_id Register is set to `0xB105F00D`. You can access the register with one wait state. Table 3-28 lists the register bit assignments.

Table 3-28 pcell_id Register bit assignments

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

Figure 3-28 on page 3-37 shows the register bit assignments.

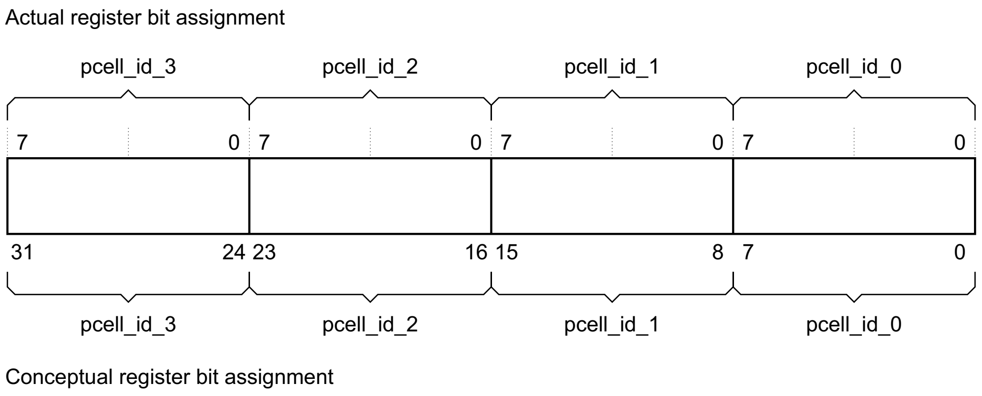


Figure 3-28 pcell_id Register bit assignments

The following subsections describe the pcell_id Registers:

- *PrimeCell Identification Register 0*
- *PrimeCell Identification Register 1* on page 3-38
- *PrimeCell Identification Register 2* on page 3-38
- *PrimeCell Identification Register 3* on page 3-38.

Note

You cannot read these registers in the Reset state.

PrimeCell Identification Register 0

The pcell_id_0 Register is hard-coded and the fields within the register determine the reset value. Table 3-29 lists the register bit assignments.

Table 3-29 pcell_id_0 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	pcell_id_0	These bits read back as `0x0D`

PrimeCell Identification Register 1

The pcell_id_1 Register is hard-coded and the fields within the register determine the reset value. Table 3-30 lists the register bit assignments.

Table 3-30 pcell_id_1 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	pcell_id_1	These bits read back as `0xF0`

PrimeCell Identification Register 2

The pcell_id_2 Register is hard-coded and the fields within the register determine the reset value. Table 3-31 lists the register bit assignments.

Table 3-31 pcell_id_2 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	pcell_id_2	These bits read back as `0x5`

PrimeCell Identification Register 3

The pcell_id_3 Register is hard-coded and the fields within the register determine the reset value. Table 3-32 lists the register bit assignments.

Table 3-32 pcell_id_3 Register bit assignments

Bits	Name	Function
[31:8]	-	Reserved, read undefined
[7:0]	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:

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

4.1 Integration test registers

Test registers are provided for integration testing.

Figure 4-1 shows the Integration Test Register map.

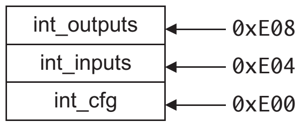


Figure 4-1 Integration test register map

Table 4-1 lists the integration test registers.

Table 4-1 SMC test register summary

Name	Base offset	Typ e	Reset value	Description
int_cfg	`0xE00`	R/W	`0x0`	*Integration Configuration Register*
int_inputs	`0xE04`	RO	-	*Integration Inputs Register* on page 4-3
int_outputs	`0xE08`	WO	-	*Integration Outputs Register* on page 4-4

Note

Signals that this section describes that end in zero are always valid. These signals reference interface 0.

Signals that this section describes that end in a one are only valid if configured for variants SMC (PL353) and SMC (PL354). These signals reference interface 1.

4.1.1 Integration Configuration Register

The read/write int_cfg Register selects the integration test registers. This register is only for test. You cannot read or write to this register in the Reset state.

Figure 4-2 on page 4-3 shows the register bit assignments.

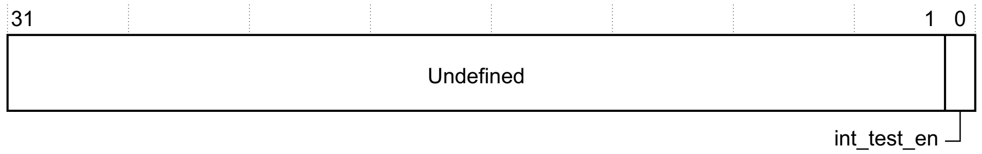


Figure 4-2 int_cfg Register bit assignments

Table 4-2 lists the register bit assignments.

Table 4-2 int_cfg Register bit assignments

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

4.1.2 Integration Inputs Register

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

Figure 4-3 shows the register bit assignments.

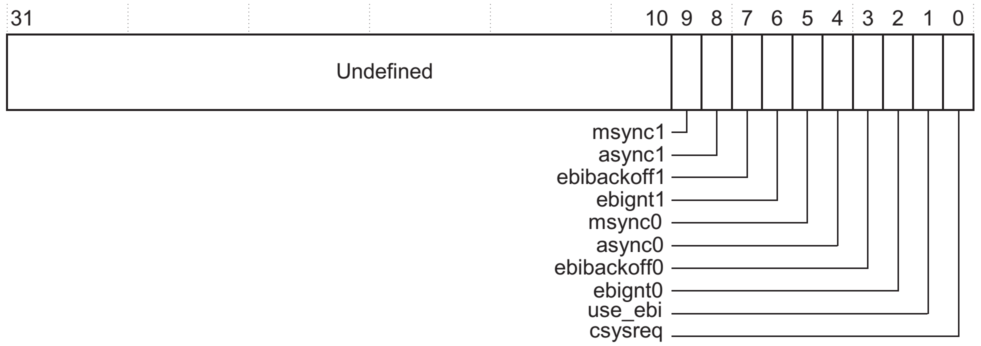


Figure 4-3 Int_inputs Register bit assignments

Table 4-3 lists the register bit assignments.

Table 4-3 Int_inputs Register bit assignments

Bits	Name	Function
[31:10]	-	Read undefined
[9]	msync1	Returns the value of this top-level tie-off
[8]	async1	Returns the value of this top-level tie-off
[7]	ebibackoff1	Returns the value of **ebibackoff1**
[6]	ebignt1	Returns the value of **ebigrant1**
[5]	msync0	Returns the value of this top-level tie-off
[4]	async0	Returns the value of this top-level tie-off
[3]	ebibackoff0	Returns the value of **ebibackoff0**
[2]	ebignt0	Returns the value of **ebigrant0**
[1]	use_ebi	Returns the value of **use_ebi**
[0]	csysreq	Returns the value of **csysreq**

4.1.3 Integration Outputs Register

The write-only int_outputs Register enables an external master to access the outputs of the SMC using the APB interface. You cannot read this register in the Reset state.

Figure 4-4 shows the register bit assignments.

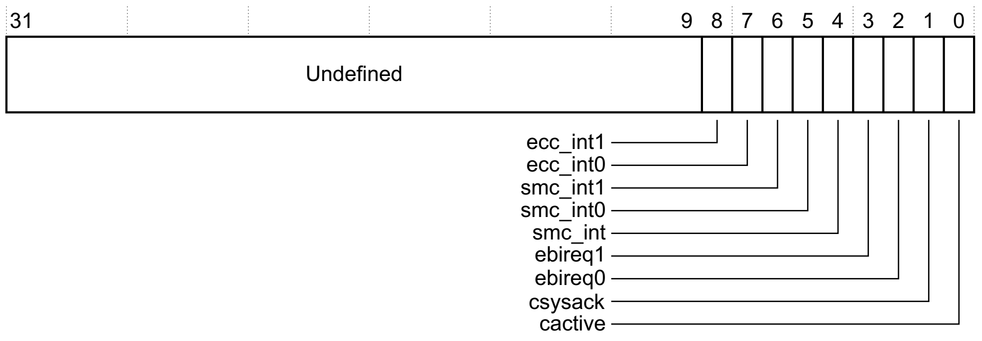


Figure 4-4 int_outputs Register bit assignments

Table 4-4 lists the register bit assignments.

Table 4-4 int_outputs Register bit assignments

Bits	Name	Function
[31:9]	-	Undefined
[8]	ecc_int1	Set the value of **ecc_int1** when in integration test mode
[7]	ecc_int0	Set the value of **ecc_int0** when in integration test mode
[6]	smc_int1	Set the value of **smc_int1** when in integration test mode
[5]	smc_int0	Set the value of **smc_int0** when in integration test mode
[4]	smc_int	Set the value of **smc_int** when in integration test mode
[3]	ebireq1	Set the value of **ebireq1** when in integration test mode
[2]	ebireq0	Set the value of **ebireq0** when in integration test mode
[1]	csysack	Set the value of **csysack** when in integration test mode
[0]	cactive	Set the value of **cactive** when in integration test mode

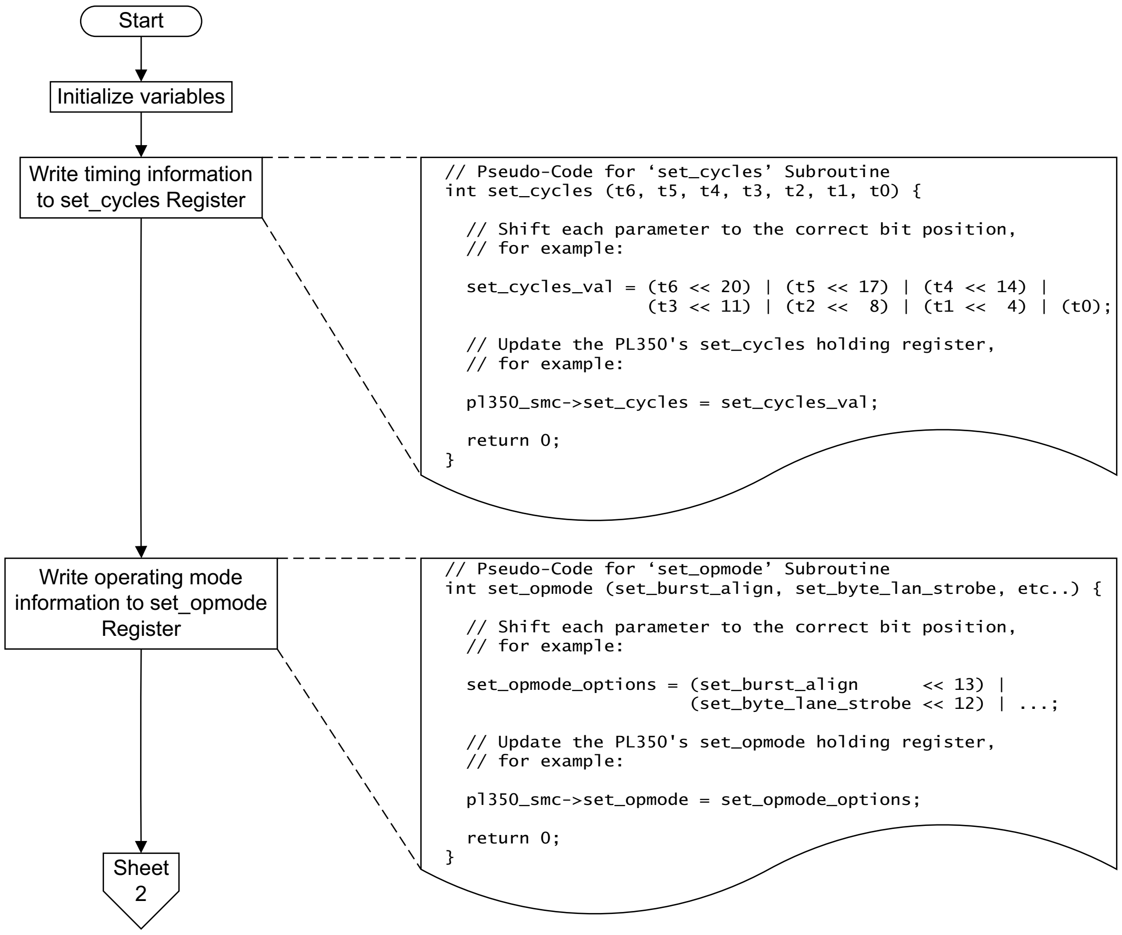
Chapter 5
Device Driver Requirements

This chapter contains flow diagrams to aid in the development of a software driver for the SMC. It contains the following sections:

5.1 Memory initialization

Figure 5-1 and Figure 5-2 on page 5-3 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.

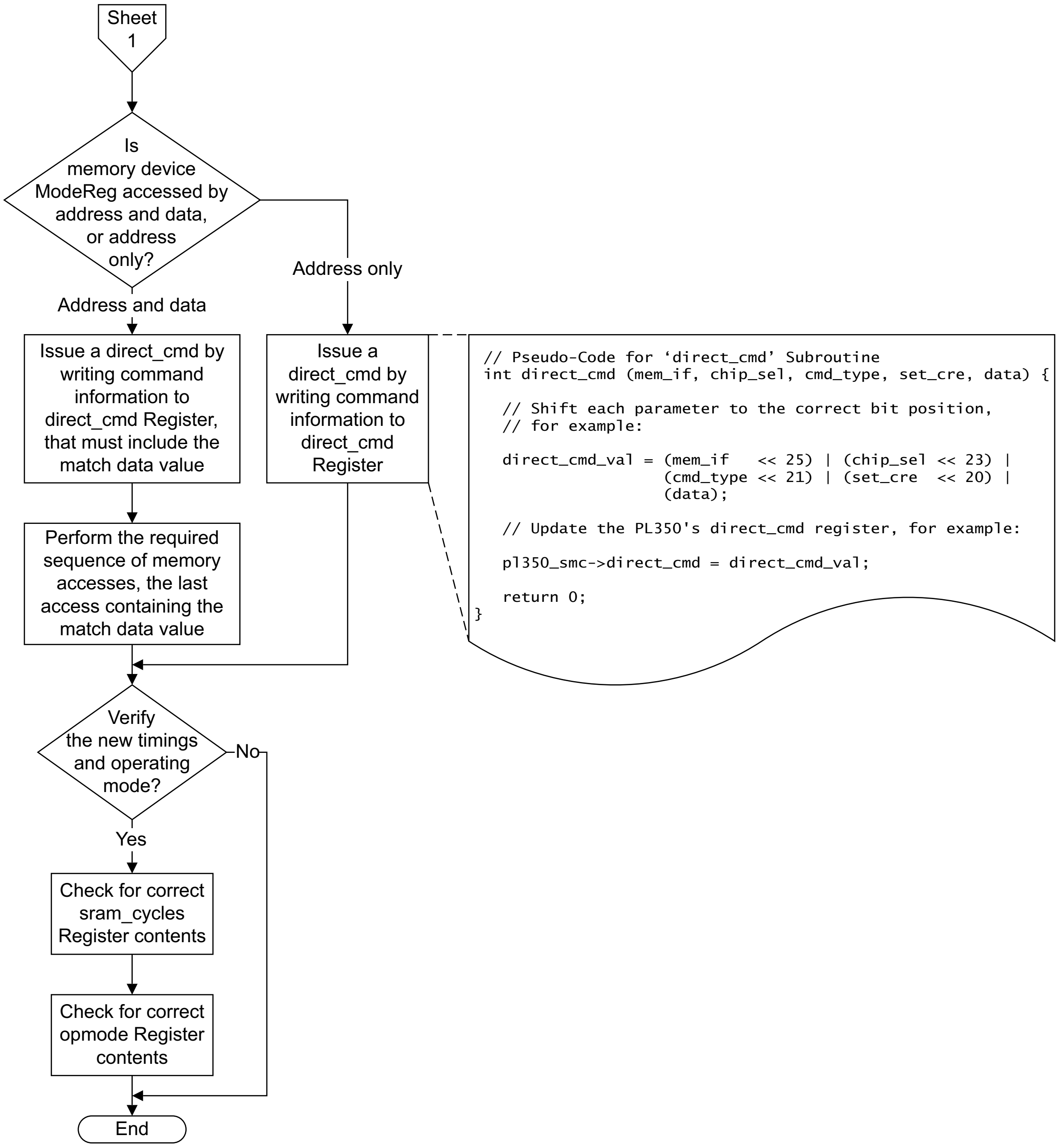
Typically, PSRAM devices can have the mode register programmed using the address bus only. NOR flash memory devices are examples of memory that require mode register accesses to be carried out using a sequence of accesses using the address and data buses. Check the data sheet for the specific memory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


Start
Initialize variables
Write timing information to set_cycles Register
Write operating mode information to set_opmode Register
Sheet 2

```
// Pseudo-Code for ‘set_cycles’ Subroutine
int set_cycles (t6, t5, t4, t3, t2, t1, t0) {

  // Shift each parameter to the correct bit position,
  // for example:

  set_cycles_val = (t6 << 20) | (t5 << 17) | (t4 << 14) |
                   (t3 << 11) | (t2 <<  8) | (t1 <<  4) | (t0);

  // Update the PL350's set_cycles holding register,
  // for example:

  pl350_smc->set_cycles = set_cycles_val;

  return 0;
}
```

```
// Pseudo-Code for ‘set_opmode’ Subroutine
int set_opmode (set_burst_align, set_byte_lan_strobe, etc..) {

  // Shift each parameter to the correct bit position,
  // for example:

  set_opmode_options = (set_burst_align      << 13) |
                       (set_byte_lane_strobe << 12) | ...;

  // Update the PL350's set_opmode holding register,
  // for example:

  pl350_smc->set_opmode = set_opmode_options;

  return 0;
}
```

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


Sheet 1
Is memory device ModeReg accessed by address and data, or address only?
Address and data
Address only
Issue a direct_cmd by writing command information to direct_cmd Register, that must include the match data value
Issue a direct_cmd by writing command information to direct_cmd Register
Perform the required sequence of memory accesses, the last access containing the match data value
Verify the new timings and operating mode?
No
Yes
Check for correct sram_cycles Register contents
Check for correct opmode Register contents
End

```
// Pseudo-Code for 'direct_cmd' Subroutine
int direct_cmd (mem_if, chip_sel, cmd_type, set_cre, data) {

  // Shift each parameter to the correct bit position,
  // for example:

  direct_cmd_val = (mem_if   << 25) | (chip_sel << 23) |
                   (cmd_type << 21) | (set_cre  << 20) |
                   (data);

  // Update the PL350's direct_cmd register, for example:

  pl350_smc->direct_cmd = direct_cmd_val;

  return 0;
}
```

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

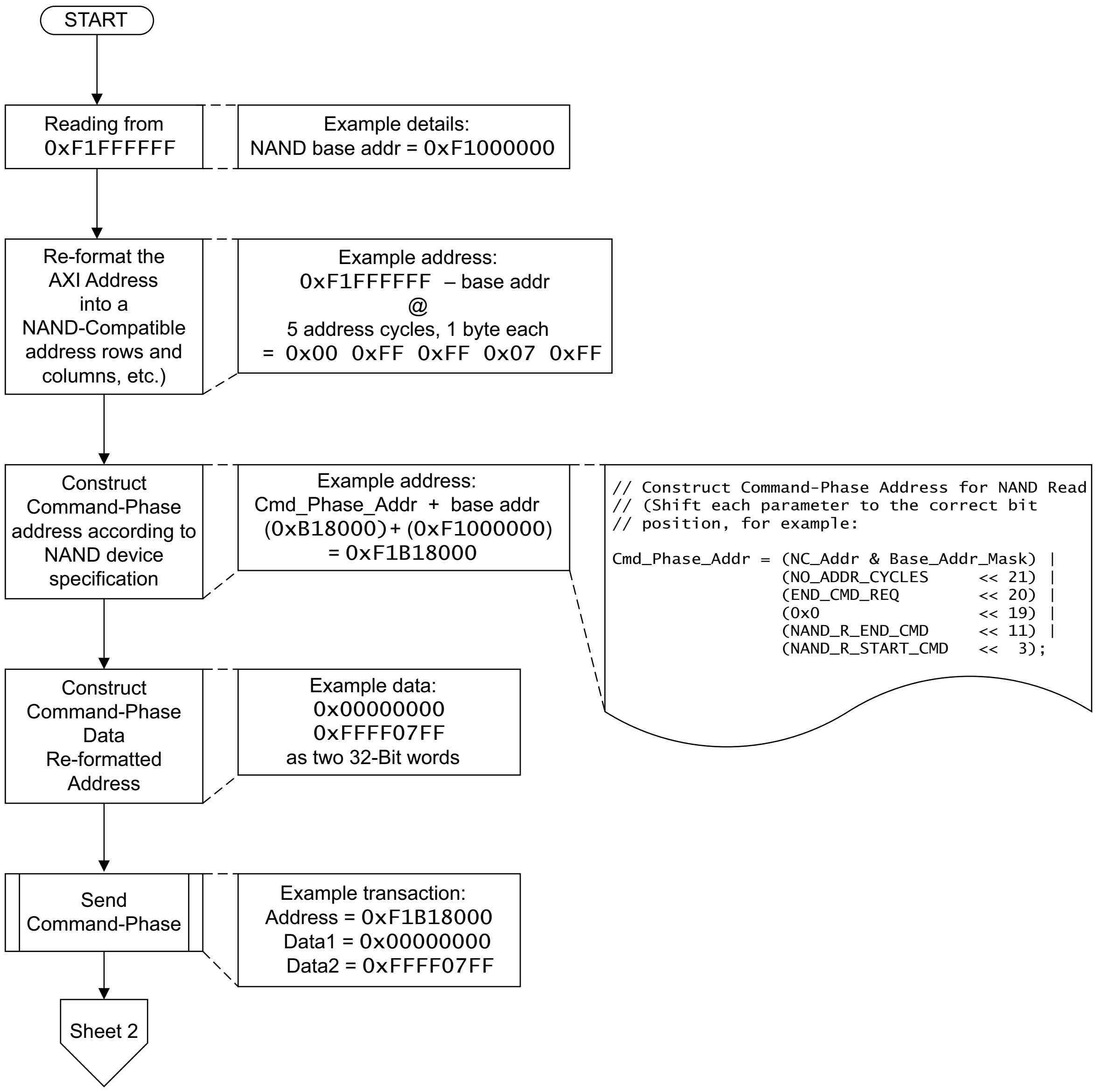
5.2 NAND transactions

The transactions for NAND devices require some specific manipulation to format the transaction into the correct format for the memory controller to map the transaction correctly to the NAND device.

The following figures show how a device driver is required to format the NAND command for the controller. See Table 2-1 on page 2-16 for a definition of how the address must be formatted:

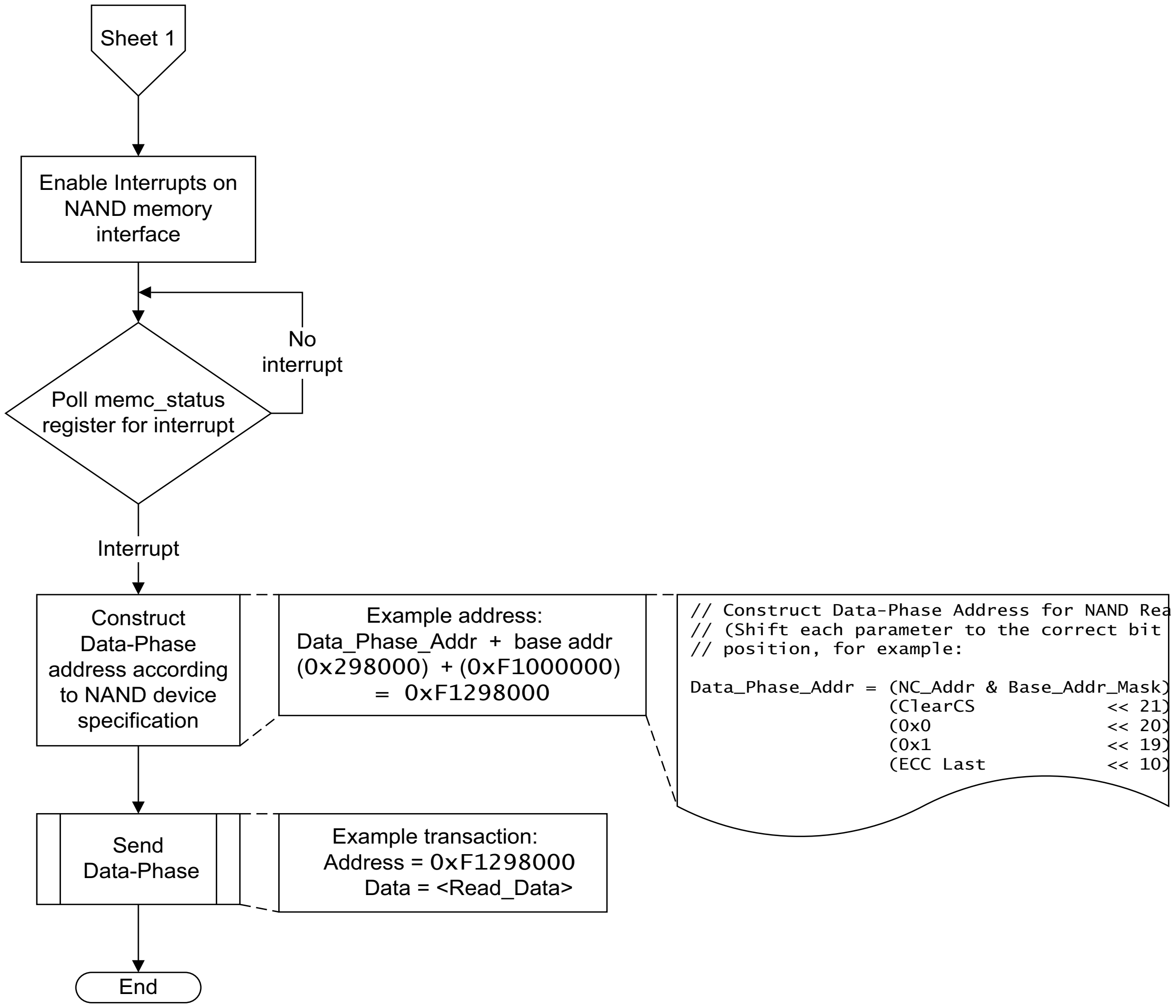
- Figure 5-3 on page 5-5 to Figure 5-4 on page 5-6
- Figure 5-5 on page 5-7 to Figure 5-6 on page 5-8.

For a command phase access, the NAND memory address is passed as data. For a data phase access, the data is passed on the data bus.



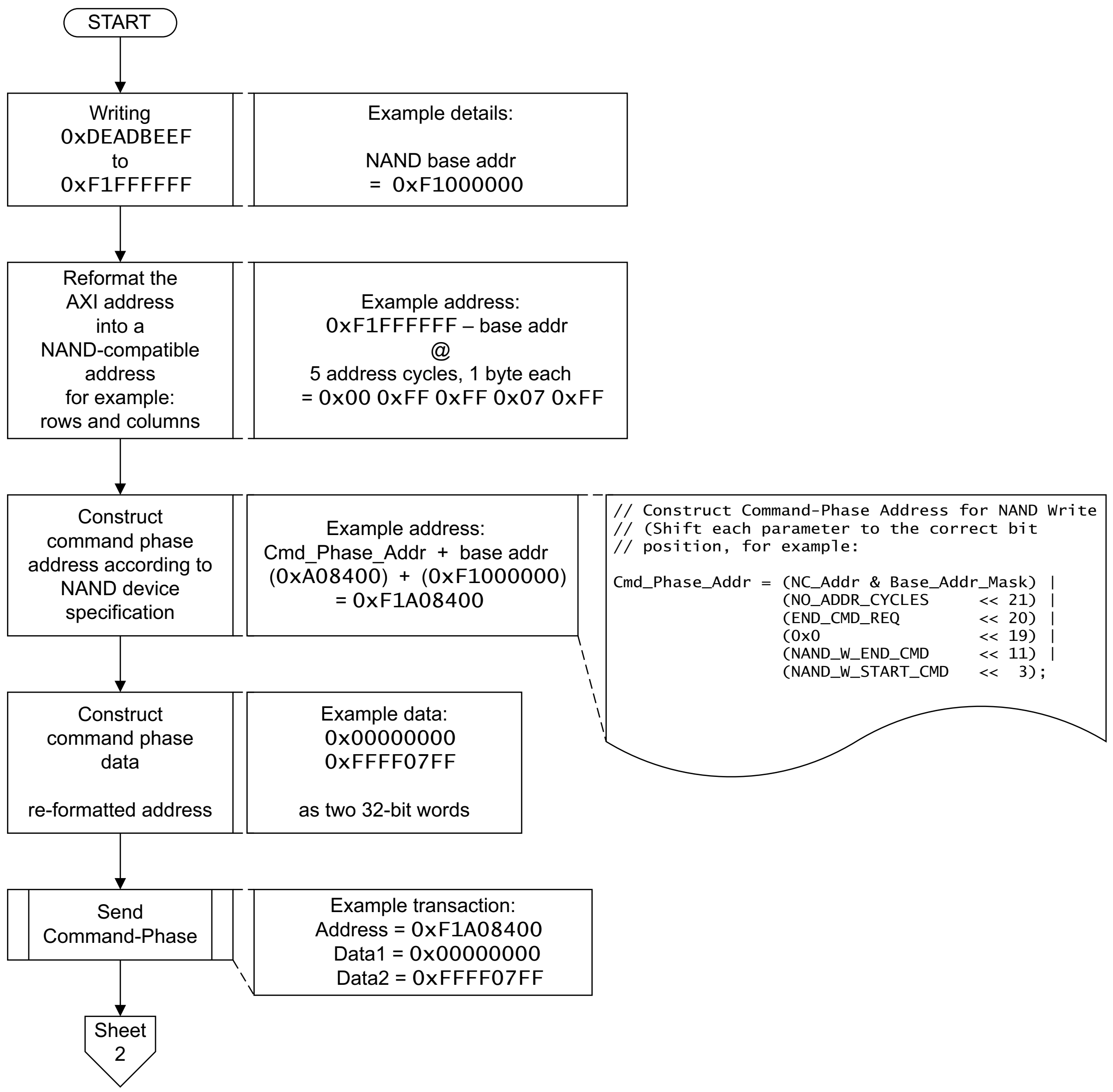
```
// Construct Command-Phase Address for NAND Read
// (Shift each parameter to the correct bit
// position, for example:

Cmd_Phase_Addr = (NC_Addr & Base_Addr_Mask) |
                 (NO_ADDR_CYCLES     << 21) |
                 (END_CMD_REQ        << 20) |
                 (0x0                << 19) |
                 (NAND_R_END_CMD     << 11) |
                 (NAND_R_START_CMD   <<  3);
```

Figure 5-3 NAND read sheet 1 of 2


Sheet 1
Enable Interrupts on NAND memory interface
Poll memc_status register for interrupt
No interrupt
Interrupt
Construct Data-Phase address according to NAND device specification
Example address:
Data_Phase_Addr + base addr
(0x298000) + (0xF1000000)
= 0xF1298000
Send Data-Phase
Example transaction:
Address = 0xF1298000
Data = <Read_Data>
End

```
// Construct Data-Phase Address for NAND Rea
// (Shift each parameter to the correct bit
// position, for example:

Data_Phase_Addr = (NC_Addr & Base_Addr_Mask)
                  (ClearCS             << 21)
                  (0x0                 << 20)
                  (0x1                 << 19)
                  (ECC Last            << 10)
```

Figure 5-4 NAND read sheet 2 of 2



```
// Construct Command-Phase Address for NAND Write
// (Shift each parameter to the correct bit
// position, for example:

Cmd_Phase_Addr = (NC_Addr & Base_Addr_Mask) |
                 (NO_ADDR_CYCLES     << 21) |
                 (END_CMD_REQ        << 20) |
                 (0x0                << 19) |
                 (NAND_W_END_CMD     << 11) |
                 (NAND_W_START_CMD   <<  3);
```

Figure 5-5 NAND write sheet 1 of 2

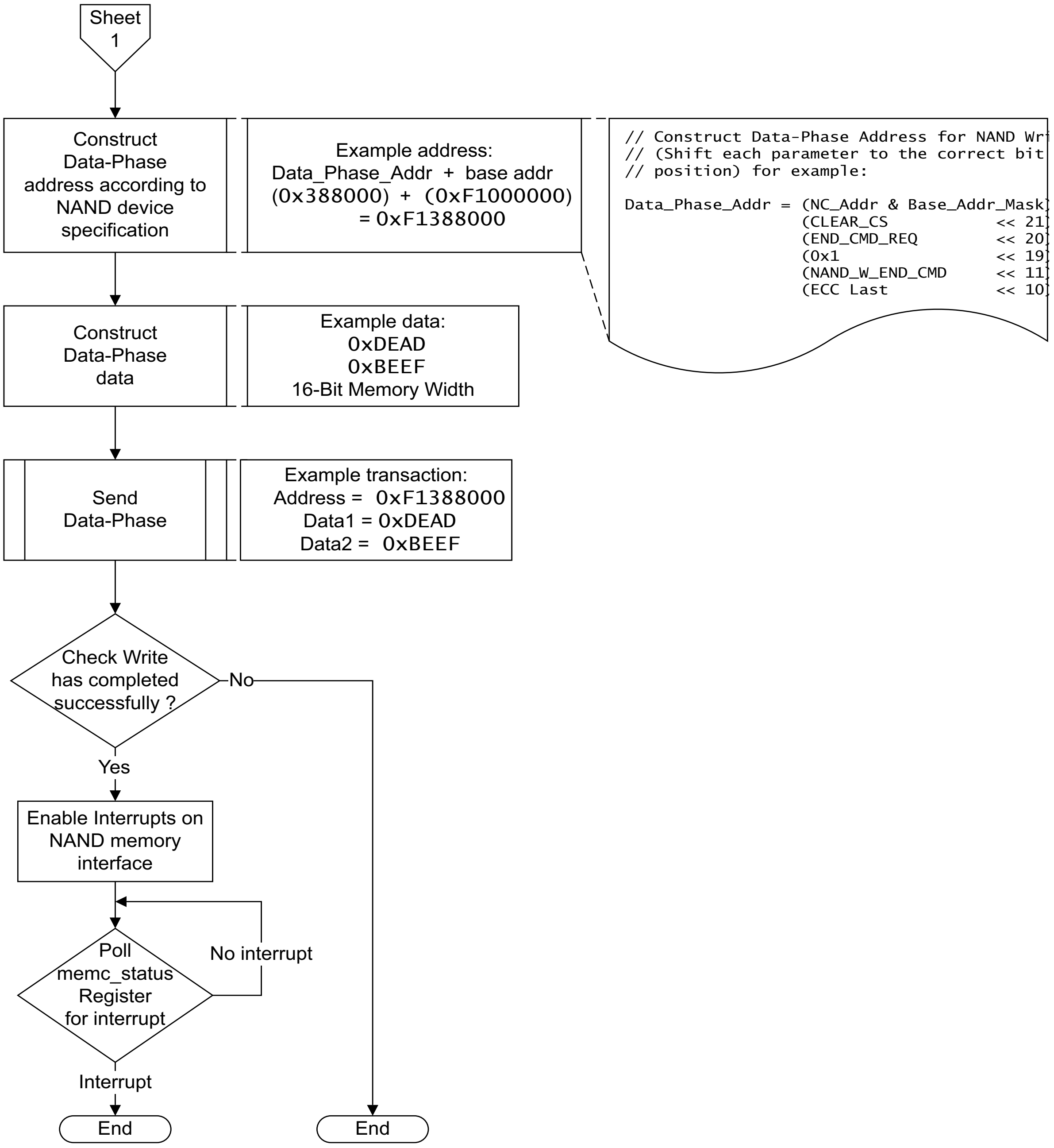

Sheet 1
Construct Data-Phase address according to NAND device specification
Example address:
Data_Phase_Addr + base addr
(0x388000) + (0xF1000000)
= 0xF1388000
// Construct Data-Phase Address for NAND Wr
// (Shift each parameter to the correct bit
// position) for example:
Data_Phase_Addr = (NC_Addr & Base_Addr_Mask
(CLEAR_CS << 21
(END_CMD_REQ << 20
(0x1 << 19
(NAND_W_END_CMD << 11
(ECC Last << 10
Construct Data-Phase data
Example data:
0xDEAD
0xBEEF
16-Bit Memory Width
Send Data-Phase
Example transaction:
Address = 0xF1388000
Data1 = 0xDEAD
Data2 = 0xBEEF
Check Write has completed successfully ?
No
Yes
Enable Interrupts on NAND memory interface
Poll memc_status Register for interrupt
No interrupt
Interrupt
End
End

Figure 5-6 NAND write sheet 2 of 2

Chapter 6
Configurations

This chapter describes the four possible configurations of the SMC (PL350 series). The configurations are fixed in the number of memory interfaces and memory types supported, but configurable to support different numbers of chip selects and data bus widths. It contains the following sections:

- *SMC (PL351)* on page 6-2
- *SMC (PL352)* on page 6-4
- *SMC (PL353)* on page 6-6
- *SMC (PL354)* on page 6-8.

——— **Note** ———

References are made to chapters and the signals appendix of this manual.

———————————

6.1 SMC (PL351)

The SMC (PL351) supports NAND flash memories.

This configuration supports a single memory interface, with the following configurable options:

- 32-bit or 64-bit AXI data width
- 8-bit or 16-bit memory data width
- 1-4 chip selects
- command FIFO depth
- read data FIFO depth
- write data FIFO depth.

See *Features of the SMC (PL350 series)* on page 1-4 for a complete list of configuration options.

6.1.1 Functional overview

AXI reads are completed in the order they are accepted by the AXI interface, as are writes. The SMC (PL351) can prioritize between AXI reads and writes.

Only two clock domains exist:

- **aclk**
- **mclk0**.

6.1.2 Programmer's model

This section describes the following registers:

- *direct_cmd*
- *set_opmode*
- *opmode* on page 6-3.

direct_cmd

Certain direct_cmd register commands have no effect for this configuration because NAND memories do not require mode-register operations. The only cmd_type supported in the Direct Command Register is b10. See *Direct Command Register* on page 3-12.

set_opmode

Only the memory width fields, bits[1:0], are relevant for NAND memories.

opmode

There is one register per chip select. Only the memory width field, bits[1:0], are relevant for NAND memories.

6.1.3 Programmer's model for test

This section describes the following registers:

- *int_inputs*
- *int_outputs*.

int_inputs

Only bits [5:0] that apply to the memory interface 0 are implemented.

int_outputs

Only bits [2:0] and bit [5] that apply to the memory interface 0 are implemented.

6.1.4 Signal descriptions

The pad interface only implements the NAND interface signals, see *NAND* on page A-16.

6.2 SMC (PL352)

The SMC (PL352) static memory controller supports a single SRAM memory interface type with the following configurable options:

- 32-bit or 64-bit AXI data width
- 8-bit, 16-bit, or 32-bit memory data width
- 1-4 chip selects
- command FIFO depth
- read data FIFO depth
- write data FIFO depth
- number of exclusive monitors.

See *Features of the SMC (PL350 series)* on page 1-4 for a complete list of configuration options.

6.2.1 Functional overview

Because the SMC (PL352) supports a single memory interface, AXI reads are completed in the order they are accepted by the AXI interface, as are writes. The SMC SMC (PL352) can prioritize between AXI reads and writes.

Only two clock domains exist:

- **aclk**
- **mclk0**.

6.2.2 Programmer's model

The SMC (PL352) implements the full functionality that Chapter 3 *Programmer's Model* describes.

6.2.3 Programmer's model for test

This section describes the following registers:

- *int_inputs*
- *int_outputs* on page 6-5.

int_inputs

Only bits [5:0] that apply to memory interface 0 are implemented.

int_outputs

Only bits [2:0] and bit [5] that apply to memory interface 0 are implemented.

6.2.4 Signal descriptions

The pad interface only implements the SRAM interface signals. See *SRAM* on page A-15.

6.3 SMC (PL353)

The SMC (PL353) static memory controller supports two memory interfaces:

Interface 0	type SRAM.
Interface 1	type NAND.

This configuration supports the following configurable options:

- 32-bit or 64-bit AXI data width
- 8-bit, 16-bit, or 32-bit memory data width for interface 0
- 8-bit, or 16-bit memory data width for interface 1
- 1-4 chip selects on each interface
- command FIFO depth
- read data FIFO depth
- write data FIFO depth
- number of exclusive monitors
- optional entry block pipeline stage.

See *Features of the SMC (PL350 series)* on page 1-4 for a complete list of configuration options.

6.3.1 Functional overview

The SMC (PL353) implements the full functionality that Chapter 2 *Functional Overview* describes.

6.3.2 Programmer's model

Only the direct_cmd Register is applicable.

direct_cmd

Certain direct_cmd register commands have no effect for this configuration, because NAND memories do not require mode-register operations. The only cmd_type supported in the Direct Command Register is b10. See *Direct Command Register* on page 3-12.

The SRAM memory interface supports all options for this field.

6.3.3 Programmer's model for test

The SMC (PL353) implements the full functionality that Chapter 3 *Programmer's Model* describes.

6.3.4 Signal descriptions

Interface 0 implements the SRAM interface signals, see *SRAM* on page A-15, with 0 appended, and Interface 1 implements the NAND interface signals, see *NAND* on page A-16, with 1 appended.

6.4 SMC (PL354)

The SMC (PL354) static memory controller supports two memory interfaces, both are of type SRAM.

This configuration supports the following configurable options:

- 32-bit or 64-bit AXI data width
- 8-bit, 16-bit, or 32-bit memory data width for interface 0
- 8-bit, 16-bit, or 32-bit memory data width for interface 1
- 1-4 chip selects on each interface
- command FIFO depth
- read data FIFO depth
- write data FIFO depth
- number of exclusive monitors
- optional entry block pipeline stage.

See *Features of the SMC (PL350 series)* on page 1-4 for a complete list of configuration options.

6.4.1 Functional overview

The SMC (PL354) implements the full functionality that Chapter 2 *Functional Overview* describes.

6.4.2 Programmer's model

The SMC (PL354) implements the full functionality that Chapter 3 *Programmer's Model* describes.

6.4.3 Programmer's model for test

The SMC (PL354) implements the full functionality that Chapter 4 *Programmer's Model for Test* describes.

6.4.4 Signal descriptions

Both memory interfaces implement the SRAM interface signals. See *SRAM* on page A-15.

Interface 0 signals have a 0 appended and interface 1 signals have a 1 appended.

Appendix A
Signal Descriptions

This appendix describes the signals that the SMC uses. It contains the following sections:

A.1 Clock and reset signals

Table A-1 lists the AXI domain clock and reset signals.

Table A-1 AXI domain clock and reset signals

Signal	Typ e	Source	Description
aclk	Input	Clock source	Clock for the **AXI** domain.
aresetn	Input	Reset source	AXI domain reset signal. This signal is active LOW.

Table A-2 lists the memory interface 0 clock and reset signals.

Table A-2 Memory interface 0 clock and reset signals

Signal	Typ e	Source	Description
mclk0	Input	Clock source	Memory clock domain 0 clock.
mclk0n	Input	Clock source	Memory clock domain 0 clock inverted.
async0	Input	Clock source	When HIGH, indicates to **aclk** domain that **aclk** is synchronous to **mclk0**. When LOW, synchronizing logic is enabled.
msync0	Input	Clock source	When HIGH, indicates to **mclk0** domain that **mclk0** is synchronous to **aclk**. When LOW, synchronizing logic is enabled.
mreset0n	Input	Reset source	Reset for **mclk0** domain 0. This signal is active LOW.
a_gt_m0_sync	Input	Clock source	When HIGH, indicates that **aclk** is faster than and synchronous to **mclk0**.

Table A-3 lists the memory interface 1 clock and reset signals.

Table A-3 Memory interface 1 clock and reset signals

Signal	Typ e	Source	Description
mclk1	Input	Clock source	Memory clock domain 1 clock.
mclk1n	Input	Clock source	Memory clock domain 1 clock inverted.
async1	Input	Clock source	When HIGH, indicates to aclk domain that **aclk** is synchronous to **mclk1**. When LOW, synchronizing logic is enabled.

Table A-3 Memory interface 1 clock and reset signals (continued)

Signal	Typ e	Source	Description
msync1	Input	Clock source	When HIGH, indicates to **mclk1** domain that **mclk1** is synchronous to **aclk**. When LOW, synchronizing logic is enabled.
mreset1n	Input	Reset source	Reset for **mclk1** domain 1.
a_gt_m1_sync	Input	Clock source	When HIGH, indicates that **aclk** is faster than and synchronous to **mclk1**.

A.2 Miscellaneous signals

The following sections describe the miscellaneous signals:

- *SRAM miscellaneous signals*
- *NAND miscellaneous signals*
- *Interrupt miscellaneous reset signals* on page A-5
- *Tie-off miscellaneous signals* on page A-5
- *User programmable signals* on page A-7.

A.2.1 SRAM miscellaneous signals

Table A-4 lists the SRAM miscellaneous signals.

Table A-4 SRAM miscellaneous signals

Signal	Type	Source	Description
remap_0	Input	SRAM or NAND	Remap SRAM or NAND chip select 0 to address `0x0`.

sram_mw_<x>[1:0]	Input	SRAM	Chip 0 memory width at power-on reset. The encoding is:
			b00 = 8-bit
			b01 = 16-bit
			b10 = 32-bit
			b11 = Reserved.

A.2.2 NAND miscellaneous signals

Table A-5 lists the NAND miscellaneous signals.

Table A-5 NAND miscellaneous signals

Signal	Type	Source	Description
nand_booten_<x>[a]	Input	Tie-off pin	Enable NAND booting functionality.
remap_1	Input	NAND or SRAM	Remap SRAM or NAND, memory interface, chip select 0 to address `0x0`.

a. The **<x>** notation represents memory interface 0 or 1.

A.2.3 Interrupt miscellaneous reset signals

Table A-6 lists the interrupt miscellaneous signals.

Table A-6 Interrupt miscellaneous reset signals

Signal	Type	Destination	Description
smc_int	Output	External	Combined interrupt output
smc_int<x>[a]	Output	External	Individual memory interrupt outputs
ecc_int<x>[b]	Output	External	Individual ECC interrupt outputs

a. The **<x>** notation represents memory interface 0 or 1. For single memory interface configurations without ECC, only the **smc_int** signal is present.
b. **ecc_int<x>** is only present for NAND interfaces with ECC configured

A.2.4 Tie-off miscellaneous signals

Table A-7 lists tie-off miscellaneous signals.

Table A-7 Tie-off miscellaneous signals

Signal	Type	Source	Description
addr_mask<x>_<n>[7:0][a]	Input	NAND/SRAM	Address mask tie-off, eight bits, 1 per chip select. A mask applied to the AXI address bits [31:24] before the comparison with the address_match value.
addr_match<x>_<n>[7:0][a]	Input	NAND/SRAM	Address match tie-off, eight bits, 1 per chip select. The comparison value that determines the chip select base address.
dft_en_clk_out	Input	Tie-off pin	Used to force the **clk_out[ ]** outputs to be the same as **mclk<x>**. This signal is used for ATPG testing.
mux_mode_<x>	Input	Tie-off pin	Tie-off to 1 to enable multiplexor mode.
nand_csl_1	Input	Tie-off pin	When tied HIGH, the chip-select remains asserted between the address phase and data phase of a transfer on the NAND interface.
rst_bypass	Input	Tie-off pin	Used to by-pass synchronization of external resets. This signal is used for ATPG testing.
use_ebi	Input	Tie-off pin	When HIGH, indicates that the SMC must operate with a *PrimeCell EBI (PL220)*.

a. The **<x>** notation represents memory interface 0 or 1. The **<n>** notation indicates that you can use chip select 0 to 3.

A.2.5 User programmable signals

Table A-8 lists user programmable miscellaneous signals.

Table A-8 User programmable miscellaneous signals

Signal	Type	Source/ destination	Description
user_config[7:0]	Output	External control logic	General purpose APB accessible output pins
user_status[7:0]	Input	External control logic	General-purpose APB accessible input pin

A.3 AXI interface signals

The following sections describe the AXI signals:

- *Write address (AXI-AW) channel signals*
- *Write data (AXI-W) channel signals* on page A-9
- *Buffered write (AXI-B) response channel signals* on page A-10
- *Read address (AXI-AR) channel signals* on page A-11
- *Read data (AXI-R) channel signals* on page A-12
- *AXI low-power interface signals* on page A-13.

A.3.1 Write address (AXI-AW) channel signals

Table A-9 lists the AXI-AW channel signals.

Table A-9 AXI-AW channel signals

Signal	Type	Source/ destination	Description
awid[7:0]	Input	Master	Address ID. This signal is the ID tag of the write transaction.
awaddr[31:0]	Input	Master	Address. The address bus gives the address of the first transfer in a burst. The associated control signals determine the addresses of the remaining transfers in the burst.
awlen[3:0]	Input	Master	Burst length. This signal indicates the number of transfers in a burst.
awsize[2:0]	Input	Master	Burst size. This signal indicates the size of each transfer in a burst. For write transfers, byte lane strobes indicate the byte lanes to update in memory.
awburst[1:0]	Input	Master	Burst type. This signal indicates a fixed, incrementing, or wrapping burst. The AXI slave uses **awburst[1:0]** and **awsize[2:0]** to calculate the address for successive transfers in the burst.
awlock[1:0]	Input	Master	Lock type. This signal indicates a normal, exclusive, or locked transaction.
awcache[3:0]	Input	Master	The controller does not use the cache information that these signals provide.

Table A-9 AXI-AW channel signals (continued)

Signal	Type	Source/ destination	Description
awprot[2:0]	Input	Master	The controller does not use the protection information that these signals provide.
awvalid	Input	Master	Address valid. This signal indicates that valid address and control information are available: 0 = address and control information not available 1 = address and control information available. The address and control information remain stable until the address acknowledge signal, **awready**, goes HIGH.
awready	Output	Slave	Address ready. This signal indicates that the slave is ready to accept an address: 0 = slave not ready 1 = slave ready.

A.3.2 Write data (AXI-W) channel signals

Table A-10 lists the AXI-W data channel signals.

Table A-10 AXI-W data channel signals

Signal	Type	Source/ destination	Description
wid[7:0]	Input	Master	Write ID tag. This signal is the ID tag of the write transfer. The **wid** value must match the **awid** value of the write transaction.
wdata[PORTWIDTH-1:0][a]	Input	Master	Write data. The write data bus can be 32 or 64 bits wide.
wstrb[PORTBYTES-1:0][b]	Input	Master	Write strobes. This signal indicates the byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, **wstrb[n]** corresponds to **wdata[(8 × n) + 7:(8 × n)]**.

Table A-10 AXI-W data channel signals (continued)

Signal	Type	Source/ destination	Description
wlast	Input	Master	Write last. This signal indicates the last transfer in a write burst.
wvalid	Input	Master	Write valid. This signal indicates that valid write data and strobes are available: 0 = write data and strobes not available 1 = write data and strobes available.
wready	Output	Slave	Write ready. This signal indicates that the slave can accept the write data: 0 = slave not ready 1 = slave ready.

a. PORTWIDTH is the data width of the AXI port in bits.
b. PORTBYTES is the data width of the AXI port in bytes.

A.3.3 Buffered write (AXI-B) response channel signals

Table A-11 lists the AXI-B response channel signals.

Table A-11 AXI-B response channel signals

Signal	Type	Source/ destination	Description
bid[7:0]	Output	Slave	Response ID. The identification tag of the write response. The **bid** value must match the **awid** value of the write transaction to which the slave is responding.
bresp[1:0]	Output	Slave	Write response. This signal indicates the status of the write transaction. The permitted responses are OKAY and EXOKAY.
bvalid	Output	Slave	Write response valid. This signal indicates that a valid write response is available: 0 = write response not available 1 = write response available.
bready	Input	Master	Response ready. This signal indicates that the master can accept the response information. 0 = master not ready 1 = master ready.

A.3.4 Read address (AXI-AR) channel signals

Table A-12 lists the AXI-AR channel signals.

Table A-12 AXI-AR channel signals

Signal	Type	Source/ destination	Description
arid[7:0]	Input	Master	Address ID. This signal is the ID tag of the read transaction.
araddr[31:0]	Input	Master	Address. The address bus gives the address of the first transfer in a burst. The associated control signals determine the addresses of the remaining transfers in the burst.
arlen[3:0]	Input	Master	Burst length. This signal indicates the number of transfers in a burst.
arsize[2:0]	Input	Master	Burst size. This signal indicates the size of each transfer in a burst. For write transfers, byte lane strobes indicate the byte lanes to update in memory.
arburst[1:0]	Input	Master	Burst type. This signal indicates a fixed, incrementing, or wrapping burst. The AXI slave uses **arburst[1:0]** and **arsize[2:0]** to calculate the address for successive transfers in the burst.
arlock[1:0]	Input	Master	Lock type. This signal indicates a normal, exclusive, or locked transaction.
arcache[3:0]	Input	Master	The controller does not use the cache information that these signals provide.
arprot[2:0]	Input	Master	The controller does not use the protection information that these signals provide.
arvalid	Input	Master	Address valid. This signal indicates that valid address and control information are available: 0 = Address and control information not available 1 = Address and control information available. The address and control information remain stable until the address acknowledge signal, **arready**, goes HIGH.
arready	Output	Slave	Address ready. This signal indicates that the slave is ready to accept an address: 0 = slave not ready 1 = slave ready.

A.3.5 Read data (AXI-R) channel signals

Table A-13 lists the AXI-R channel signals.

Table A-13 AXI-R channel signals

Signal	Type	Source/ destination	Description
rid[7:0]	Output	Slave	Read ID tag. This signal is the ID tag of the read transfer. The **rid** value must match the **arid** value of the read transaction to which the slave is responding.
rdata[PORTWIDTH-1:0][a]	Output	Slave	Read data. The read data bus can be 32 or 64 bits wide.
rresp[1:0]	Output	Slave	Read response. This signal indicates the status of the read transfer. The permitted responses are OKAY and EXOKAY.
rlast	Output	Slave	Read last. This signal indicates the last transfer in a read burst.
rready	Input	Master	Read ready. This signal indicates that the master can accept the read data and response information: 0 = master not ready 1= master ready.
rvalid	Output	Slave	Read valid. This signal indicates that valid read data is available: 0 = read data not available 1 = read data available.

a. PORTWIDTH is the data width of the AXI port in bits.

A.3.6 AXI low-power interface signals

Table A-14 lists the optional low-power interface signals.

Table A-14 AXI low-power interface signals

Signal	Type	Source/ destination	Description
cclken	Input	Bus clock	Clock enable.
csysreq	Input	Memory manager	System low-power request. This signal is a request from the system clock controller for the peripheral to enter a low-power state.
csysack	Output	Peripheral device	Low-power request acknowledgement. This signal is the acknowledgement from a peripheral of a system low-power state request.
cactive	Output	Peripheral device	Clock active. This signal indicates that the peripheral requires its clock signal: 0 = peripheral clock not required 1 = peripheral clock required.

A.4 APB signals

Table A-15 lists the APB signals.

Table A-15 APB signals

Signal	Type	Source/ destination	Description
paddr[31:0]	Input	APB address bus	This is the APB address bus. The full 32-bits of APB address is included for completeness, but only **paddr[11:2]** are used in the SMC.
pclken	Input	Bus clock	Clock enable for **clk** in APB domain.
penable	Input	APB strobe	This strobe signal times all accesses on the peripheral bus. The enable signal indicates the second cycle of an APB transfer. The rising edge of **penable** occurs in the middle of the APB transfer.
prdata[31:0]	Output	APB read data bus	The read data bus is driven by the selected slave during read cycles, when **pwrite** is LOW. The read data bus can be up to 32-bits wide.
pready	Output	APB	APB transfer wait signal.
psel	Input	APB select	A signal from the secondary decoder, within the peripheral bus bridge unit, to each peripheral bus slave x. This signal indicates that the slave device is selected, and that a data transfer is required. There is a **psel** signal for each bus slave.
pslverr	Output	APB transfer error	This is tied off within the SMC. Included for completeness.
pwdata[31:0]	Input	APB write data bus	The write data bus is driven by the peripheral bus bridge unit during write cycles, when **pwrite** is HIGH. The write data bus can be up to 32-bits wide.
pwrite	Input	APB transfer direction	When HIGH, this signal indicates an APB write access and when LOW, indicates an APB read access.

A.5 Pad interface signals

This following section describes the SRAM and NAND pad interface signals:

- *SRAM*
- *NAND* on page A-16.

A.5.1 SRAM

Table A-16 lists the SRAM pad interface signals.

Table A-16 SRAM pad interface signals

Signal	Type	Source/ destination	Description
add_<x>[31:0]	Input	External memory	Address
adv_n_<x>	Output	External memory	Address valid
baa_n_<x>	Output	External memory	Burst address advance
bls_n_<x>[(MEMWIDTH/8)-1:0][a]	Output	External memory	Byte lane strobe
clk_out_<x>[MEMORIES-1:0][b]	Output	External memory	Memory clock
cre_<x>	Output	External memory	Configuration register write
cs_n_<x>[MEMORIES-1:0][b]	Output	External memory	Chip select
data_en_<x>	Output	External memory	Data bus tristate enable
data_in_<x>[MEMWIDTH-1:0][a]	Input	External memory	Data in
data_out_<x>[MEMWIDTH-1:0][a]	Output	External memory	Data out
fbclk_in_<x>	Input	External memory	Feedback clock
int_<x>	Input	External memory	Interrupt
oe_n_<x>	Output	External memory	Output enable
wait_<x>	Input	External memory	Wait
we_n_<x>	Output	External memory	Write enable

a. Memory data bus width.
b. MEMORIES is the number of chip selects.

A.5.2 NAND

Table A-17 lists the NAND pad interface signals.

Table A-17 NAND pad interface signals

Signal	Type	Source/ destination	Description
ale_<x>	Output	External memory	Address latch enable
busy_<x>	Input	External memory	Busy
cle	Output	External memory	Command latch enable
cs_n_<x>[MEMORIES-1:0][a]	Output	External memory	Chip select
data_en_<x>	Output	External memory	Data bus tristate enable
data_in_<x>[MEMWIDTH-1:0][b]	Input	External memory	Data in
data_out_<x>[MEMWIDTH-1:0][b]	Output	External memory	Data out
re_n_<x>	Output	External memory	Read enable
we_n_<x>	Output	External memory	Write enable

a. MEMORIES is the number of chip selects.
b. Memory data bus width.

A.6 EBI signals

Table A-18 lists the EBI signals.

Table A-18 EBI signals

Signal[a]	Type	Source/ destination	Description
ebibackoff<x>	Input	SMC	External memory bus access backoff. The EBI backoff signal goes active HIGH when the EBI wants to remove the SMC from the memory bus so that another memory controller can be granted the memory bus.
ebigrant<x>	Input	SMC	External memory bus grant. The EBI grant signal goes HIGH when the EBI grants the external memory bus.
ebireq<x>	Output	External	External memory bus request. The EBI request signal goes HIGH when the SMC requires the memory bus.

a. The **<x>** notation represents memory interface 0 or 1.

Glossary

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

Advanced eXtensible Interface (AXI)

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

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

Advanced 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 details 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 AXI and 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.

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.

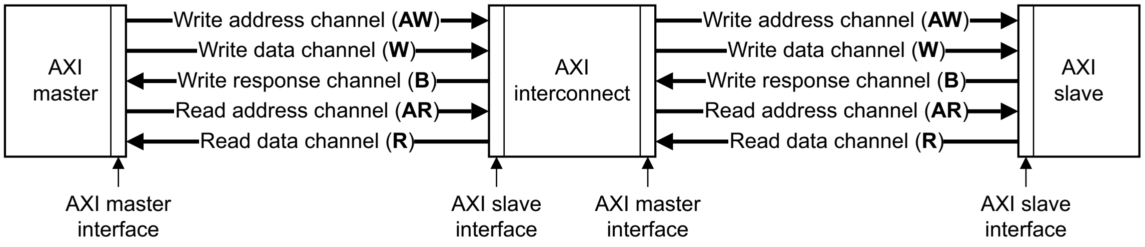
APB *See* Advanced Peripheral Bus.

AXI *See* Advanced eXtensible Interface.

AXI channel order and interfaces

The block diagram shows:

- the order in which AXI channel signals are described
- the master and slave interface conventions for AXI components.



AXI terminology The following AXI terms are general. They apply to both masters and slaves:

Active read transaction

A transaction for which the read address has transferred, but the last read data has not yet transferred.

Active transfer

A transfer for which the **xVALID**[1] handshake has asserted, but for which **xREADY** has not yet asserted.

Active write transaction

A transaction for which the write address or leading write data has transferred, but the write response has not yet transferred.

Completed transfer

A transfer for which the **xVALID/xREADY** handshake is complete.

Payload The non-handshake signals in a transfer.

Transaction An entire burst of transfers, comprising an address, one or more data transfers and a response transfer (writes only).

Transmit An initiator driving the payload and asserting the relevant **xVALID** signal.

Transfer A single exchange of information. That is, with one **xVALID/xREADY** handshake.

The following AXI terms are master interface attributes. To obtain optimum performance, they must be specified for all components with an AXI master interface:

Combined issuing capability

The maximum number of active transactions that a master interface can generate. This is specified instead of write or read issuing capability for master interfaces that use a combined storage for active write and read transactions.

Read ID capability

The maximum number of different **ARID** values that a master interface can generate for all active read transactions at any one time.

Read ID width

The number of bits in the **ARID** bus.

Read issuing capability

The maximum number of active read transactions that a master interface can generate.

Write ID capability

The maximum number of different **AWID** values that a master interface can generate for all active write transactions at any one time.

1. The letter **x** in the signal name denotes an AXI channel as follows:

AW	Write address channel.
W	Write data channel.
B	Write response channel.
AR	Read address channel.
R	Read data channel.

Write ID width

The number of bits in the **AWID** and **WID** buses.

Write interleave capability

The number of active write transactions for which the master interface is capable of transmitting data. This is counted from the earliest transaction.

Write issuing capability

The maximum number of active write transactions that a master interface can generate.

The following AXI terms are slave interface attributes. To obtain optimum performance, they must be specified for all components with an AXI slave interface:

Combined acceptance capability

The maximum number of active transactions that a slave interface can accept. This is specified instead of write or read acceptance capability for slave interfaces that use a combined storage for active write and read transactions.

Read acceptance capability

The maximum number of active read transactions that a slave interface can accept.

Read data reordering depth

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

Write acceptance capability

The maximum number of active write transactions that a slave interface can accept.

Write interleave depth

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

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

See also Burst.

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 which byte lanes are active in a transfer. One bit of this signal corresponds to eight bits of the data bus.
Direct Memory Access (DMA)	An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.
DMA	*See* Direct Memory Access.
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.