# ARM PrimeCell Multimedia Card Interface (PL180)

## Technical Reference Manual

ARM®


ARM DDI0172A

# ARM PrimeCell Multimedia Card Interface (PL180)
## Technical Reference Manual



### Release Information

**Change History**

| Date | Issue | Confidentiality | Change |
|---|---|---|---|
| 21 July 1998 | A | Non-Confidential | First release |

### Proprietary Notice



### Confidentiality Status

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

### Product Status

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

### Web Address

`http://www.arm.com`

# Contents
# ARM PrimeCell Multimedia Card Interface (PL180) Technical Reference Manual

# List of Tables
# ARM PrimeCell Multimedia Card Interface (PL180) Technical Reference Manual

# List of Figures
# ARM PrimeCell Multimedia Card Interface (PL180) Technical Reference Manual

# Chapter 1
# **Introduction**

This chapter describes the ARM PrimeCell Multimedia Card Interface (PL180) and contains the following section:

## 1.1 About the ARM PrimeCell Multimedia Card Interface (PL180)

The PrimeCell *Multimedia Card Interface* (MCI) is an *Advanced Microcontroller Bus Architecture*(AMBA) compliant, *System-on-a-Chip* (SoC) peripheral that is developed, tested, and licensed by ARM.

See Figure 1-1 for a simplified block diagram of the PrimeCell MCI.

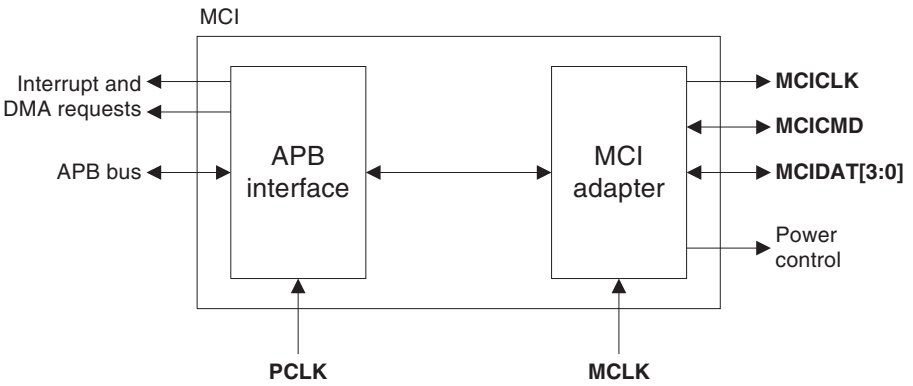



**Figure 1-1 PrimeCell MCI block diagram**

The PrimeCell MCI is an interface between the Advanced Peripheral Bus (APB) system bus and multimedia and/or secure digital memory cards. It consists of two parts:

- The PrimeCell MCI adapter block provides all functions specific to the multimedia/secure digital memory card, such as the clock generation unit, power management control, command and data transfer.
- The APB interface accesses the PrimeCell MCI adapter registers, and generates interrupt and DMA request signals.

### 1.1.1 Features of the PrimeCell MCI

The following features are provided by the PrimeCell MCI:

- Conformance to *Multimedia Card Specification v2.11*.
- Conformance to *Secure Digital Memory Card Physical Layer Specification, v0.96*.
- Use as a multimedia card bus or a secure digital memory card bus host. It can be connected to up to 30 cards as a multimedia card bus, or up to 16 cards as a secure digital memory card bus.

# Chapter 2
# **Functional Overview**

This chapter provides an overview of the PrimeCell *Multimedia Card Interface* (MCI) and its interface with the multimedia card system and the secure digital memory card system. It contains the following sections:

## 2.1 About the ARM PrimeCell MCI (PL180)

The PrimeCell MCI provides an interface between the APB system bus and multimedia and/or secure digital memory cards. It consists of two parts:

- The PrimeCell MCI adapter block provides all functions specific to the multimedia/secure digital memory card. These include the clock generation unit, power management control, command and data transfer (see *PrimeCell MCI adapter* on page 2-7 for more information).
- The APB interface provides access to the PrimeCell MCI adapter registers, and generates interrupt and DMA request signals (see *APB interface* on page 2-23 for more information).

You can use the PrimeCell MCI as a multimedia card bus host (see *Multimedia card system*) or a secure digital memory card bus host (see *Secure digital memory card system* on page 2-4). You can connect up to 30 cards as a multimedia card bus, or up to 16 cards as a secure digital memory card bus, which can include multimedia cards (see *Secure digital memory card bus* on page 2-5 for more information).

The PrimeCell MCI uses two clock signals, each with a maximum frequency of 100 MHz. One or both clocks can be switched off for power saving:

- PrimeCell MCI adapter clock, **MCLK**
- APB bus clock, **PCLK**.

The relationship between **MCLK** and **PCLK** is defined below:

$$fp \geq \frac{3}{8} fm$$

where p = **PCLK** and m = **MCLK**.

### 2.1.1 Multimedia card system

Figure 2-1 on page 2-3 shows the multimedia card system.

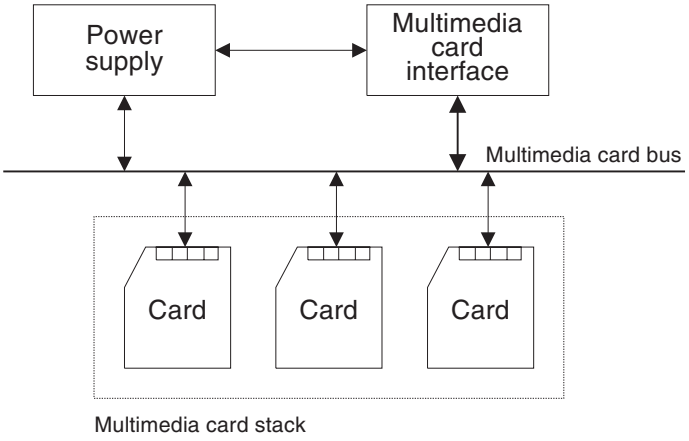



**Figure 2-1 Multimedia card system**

The multimedia card system provides communication and data storage, and consists of:

- A multimedia card stack. This can consist of up to 30 cards on a single physical bus.
- A multimedia card controller: This is the multimedia card master, and provides an interface between a system bus and the multimedia card bus.

Multimedia cards are grouped into three types according to their function:

- *Read Only Memory* (ROM) cards, containing preprogrammed data
- *Read/Write* (R/W) cards, used for mass storage
- *Input/Output* (I/O) cards, used for communication.

The multimedia card system transfers commands and data using three signal lines:

**CLK** One bit is transferred on both command and data lines with each clock cycle. The clock frequency varies between 0 MHz and 20 MHz (for a multimedia card) or 0 MHz and 25 MHz (for a secure digital memory card).

**CMD** Bidirectional command channel that initializes a card and transfers commands. **CMD** has two operational modes:

- Open-drain for initialization
- Push-pull for command transfer.

**DAT** Bidirectional data channel, operating in push-pull mode.

### 2.1.2 Secure digital memory card system

Figure 2-2 shows the secure digital memory card system.

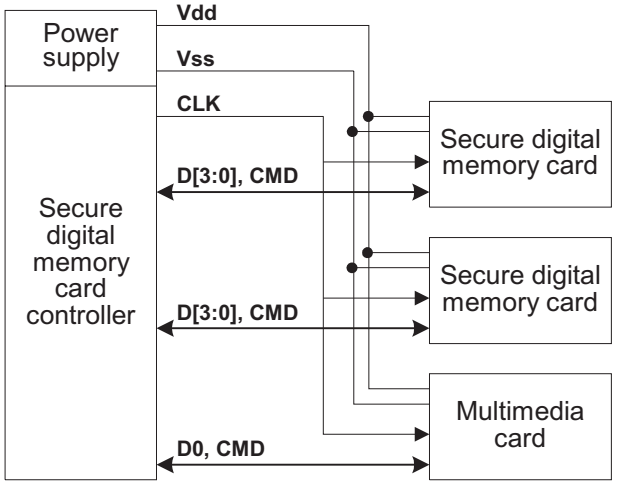



**Figure 2-2 Secure digital memory card system**

The secure digital memory card system consists of the host and cards connected in a star bus topology. The system host contains the secure digital card controller and a power supply.

**Note**

The power supply is not described in this document.

The secure digital memory card system is described in the following sections:

- *Secure digital memory card bus* on page 2-5
- *Secure digital memory card bus signals* on page 2-6.

**Note**

Multimedia cards and secure digital memory cards can be used in the same system, as shown in Figure 2-2.

### Secure digital memory card bus

The secure digital memory card bus is implemented using multiplexing logic, as shown in Figure 2-3. The secure digital memory card select register output (see *Secure digital memory card select register, MCISelect* on page 3-16) controls the output demultiplexers (1 to N) and the input multiplexers (N to 1).

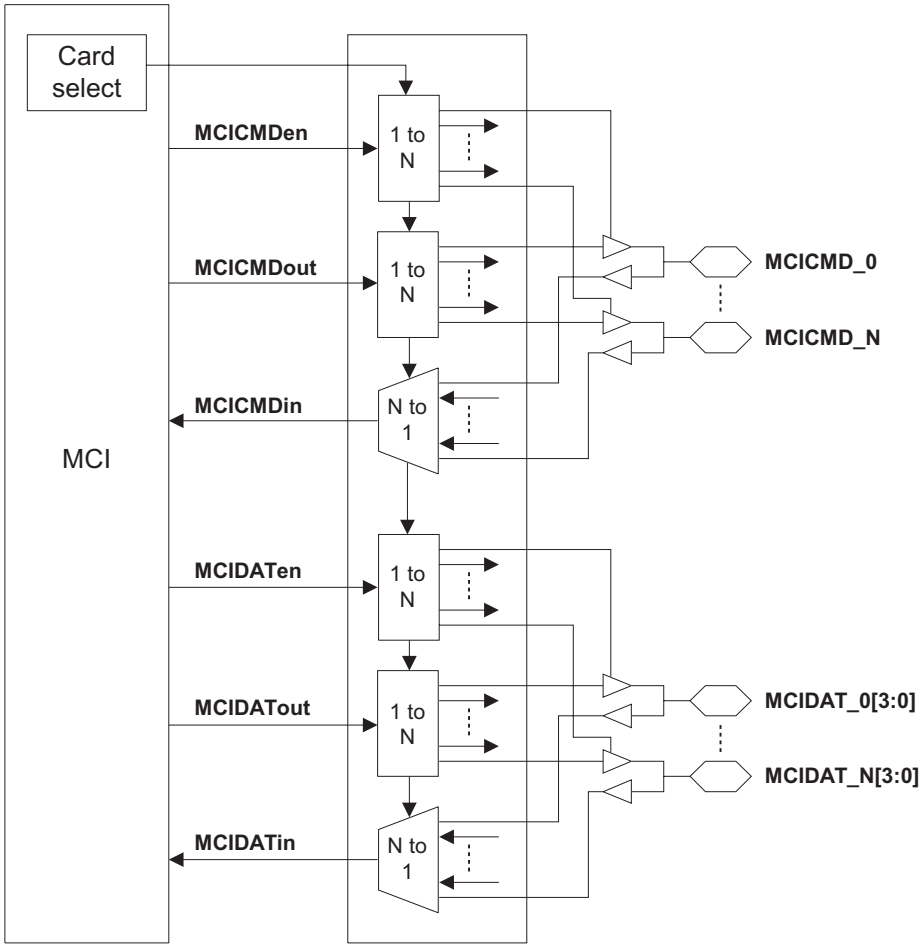



**Figure 2-3 Secure digital memory card bus implementation**

The maximum number of cards that can be installed in a secure digital memory card system depends on the number of data ports on the secure digital card controller. The clock (**CLK**), power (**Vdd**), and ground (**Vss**) are common to all cards, while the command and data (**DAT[3:0]**) signals are dedicated to each card. After power-up, the

secure digital cards only use **DAT0** for data transfer. After initialization, the host can change the data bus width. If a multimedia card is connected to the secure digital card controller, only **DAT0** is used for data transfer.

### Secure digital memory card bus signals

The following signals are used on the secure digital memory card bus:

| | |
|---|---|
| **CLK** | Host to card clock signal. |
| **CMD** | Bidirectional command/response signal (one per card). |
| **DAT[3:0]** | Bidirectional data signals (one per card). |
| **Vdd, Vss1, Vss2** | Power and ground signals. |

**Note**

The PrimeCell MCI does not contain the bus multiplexing logic. You must implement this logic when you integrate the block in a system. The configuration depends on the number of ports required.

## 2.2 PrimeCell MCI adapter

Figure 2-4 shows a simplified block diagram of the PrimeCell MCI adapter.

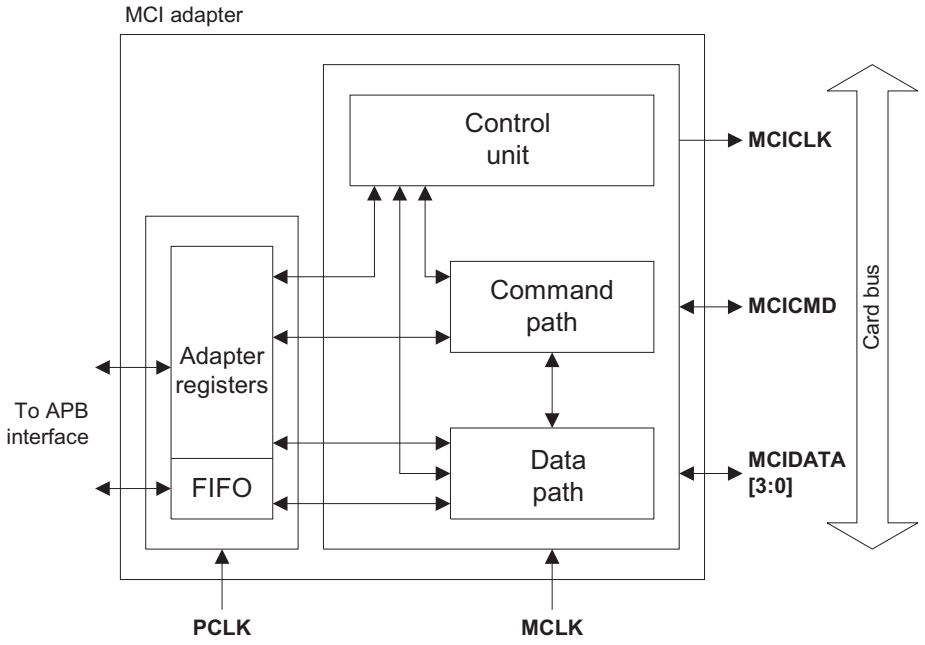



**Figure 2-4 PrimeCell MCI Adapter**

The PrimeCell MCI adapter is a multimedia/secure digital memory card bus master that provides an interface to the multimedia card stack or to the secure digital memory cards. It consists of five subunits:

- *Adapter register block* on page 2-8
- *Control unit* on page 2-8
- *Command path* on page 2-9
- *Data path* on page 2-13
- *Data FIFO* on page 2-19.

**Note**

The adapter registers and FIFO use the APB bus clock domain. The control unit, command path, and data path use the PrimeCell MCI adapter clock domain.

In Figure 2-4, the card select and power connections are not shown for clarity.

### 2.2.1 Adapter register block

The adapter register block contains all system registers. This block also generates the signals that clear the static flags in the multimedia card. The clear signals are generated when 1 is written into the corresponding bit location of the MCIClear register. The clear signal for flags generated in the **MCLK** domain is synchronized to that domain.

### 2.2.2 Control unit

The control unit contains the power management functions and the card bus clock divider. Figure 2-5 shows a block diagram of the control unit.

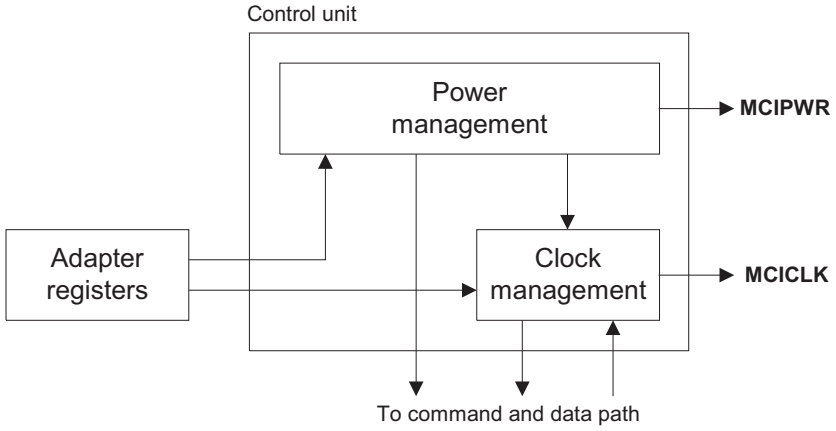



**Figure 2-5 Control unit**

There are three power phases:

- power-off
- power-up
- power-on.

The power management logic controls an external power supply unit, and disables the card bus output signals during the power-off or power-up phases. The power-up phase is a transition phase between the power-off and power-on phases, and allows an external power supply to reach the card bus operating voltage. A device driver is used to ensure that the PrimeCell MCI remains in the power-up phase until the external power supply reaches the operating voltage.

The clock management logic generates and controls the **MCICLK** signal. The **MCICLK** output can use either a clock divide or clock bypass mode. The clock output is inactive:

- after the PrimeCell MCI is reset
- during the power-off or power-up phases
- if the power saving mode is enabled and the card bus is in the IDLE state (eight clock periods after both the command and data path subunits enter the IDLE phase).

### 2.2.3 Command path

The command path subunit sends commands to and receives responses from the cards. Figure 2-6 shows a block diagram of the command path.

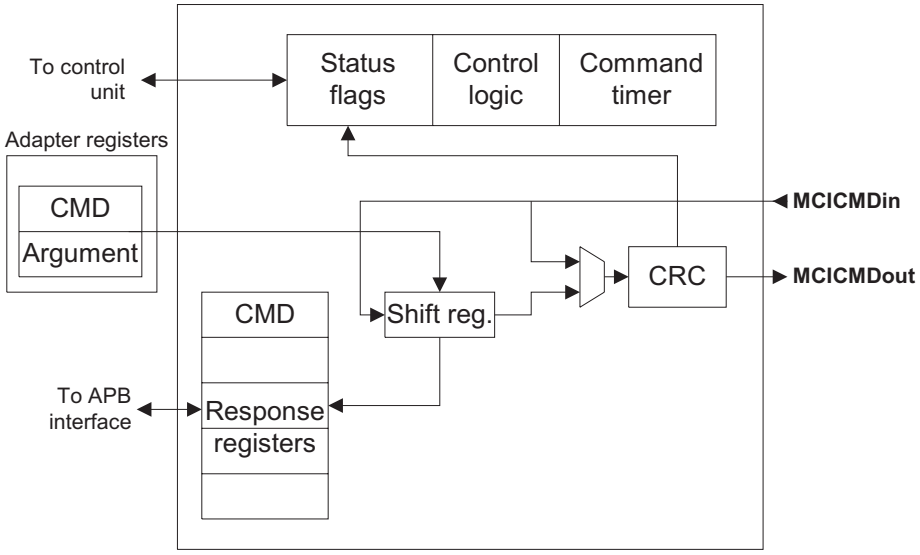



**Figure 2-6 Command path**

#### Command path state machine

When the command register is written to and the enable bit is set, command transfer starts. When the command has been sent, the *Command Path State Machine* (CPSM) sets the status flags and enters the IDLE state if a response is not required. If a response is required, it waits for the response (see Figure 2-7 on page 2-10). When the response is received, the received CRC code and the internally generated code are compared, and the appropriate status flags are set.

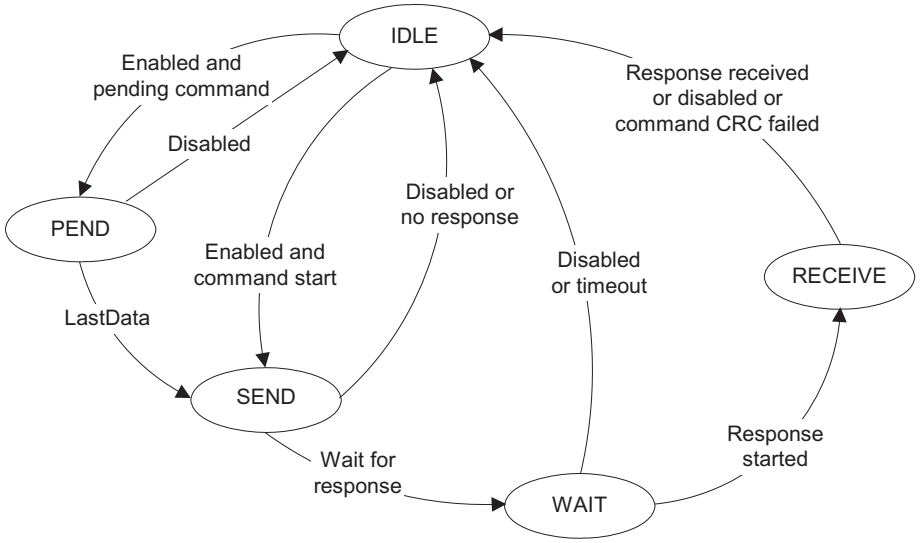



**Figure 2-7 Command path state machine**

When the WAIT state is entered, the command timer starts running. If the timeout is reached before the CPSM moves to the RECEIVE state, the timeout flag is set and the IDLE state is entered.

**Note**

The timeout period has a fixed value of 64 **MCICLK** clock periods.

If the interrupt bit is set in the command register, the timer is disabled and the CPSM waits for an interrupt request from one of the cards. If a pending bit is set in the command register, the CPSM enters the PEND state, and waits for a **CmdPend** signal from the data path subunit. When **CmdPend** is detected, the CPSM moves to the SEND state. This enables the data counter to trigger the stop command transmission.

**Note**

The CPSM remains in the IDLE state for at least eight **MCICLK** periods to meet Ncc and Nrc timing constraints.

Figure 2-8 on page 2-11 shows the PrimeCell MCI command transfer.

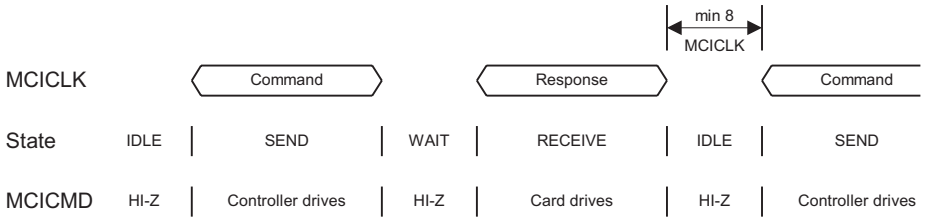



**Figure 2-8 PrimeCell MCI command transfer**

### Command format

The command path operates in a half-duplex mode, so that commands and responses can either be sent or received. If the CPSM is not in the SEND state, the **MCICMD** output is in HI-Z state, as shown in Figure 2-8. Data on **MCICMD** is synchronous to the rising **MCICLK** edge. All commands have a fixed length of 48 bits. Table 2-1 shows the command format.

**Table 2-1 Command format**

| Bit position | Width | Value | Description |
|---|---|---|---|
| 47 | 1 | 0 | Start bit |
| 46 | 1 | 1 | Transmission bit |
| [45:40] | 6 | - | Command index |
| [39:8] | 32 | - | Argument |
| [7:1] | 7 | - | CRC7 |
| 0 | 1 | 1 | End bit |

The PrimeCell MCI adapter supports two response types. Both use CRC error checking:

- 48 bit short response (see Table 2-2 on page 2-12)
- 136 bit long response (see Table 2-3 on page 2-12).

**Note**

If the response does not contain CRC (CMD1 response), the device driver must ignore the CRC failed status.

**Table 2-2 Short response format**

| Bit position | Width | Value | Description |
|---|---|---|---|
| 47 | 1 | 0 | Start bit |
| 46 | 1 | 0 | Transmission bit |
| [45:40] | 6 | - | Command index |
| [39:8] | 32 | - | Argument |
| [7:1] | 7 | - | CRC7 (or 1111111) |
| 0 | 1 | 1 | End bit |

**Table 2-3 Long response format**

| Bit position | Width | Value | Description |
|---|---|---|---|
| 135 | 1 | 0 | Start bit |
| 134 | 1 | 1 | Transmission bit |
| [133:128] | 6 | 111111 | Reserved |
| [127:1] | 127 | - | CID or CSD (including internal CRC7) |
| 0 | 1 | 1 | End bit |

The command register contains the command index (six bits sent to a card) and the command type. These determine whether the command requires a response, and whether the response is 48 or 136 bits long, see *Command register, MCICommand* on page 3-7 for more information. The command path implements the status flags shown in Table 2-4 , see *Status register, MCIStatus* on page 3-12 for more information.

**Table 2-4 Command path status flags**

| Flag | Description |
|---|---|
| CmdRespEnd | Set if response CRC is OK |
| CmdCrcFail | Set if response CRC fails |

**Table 2-4 Command path status flags (continued)**

| Flag | Description |
|---|---|
| CmdSent | Set when command (that does not require response) is sent |
| CmdTimeOut | Response timeout |
| CmdActive | Command transfer in progress |

The CRC generator calculates the CRC checksum for all bits before the CRC code. This includes the start bit, transmitter bit, command index, and command argument (or card status). The CRC checksum is calculated for the first 120 bits of CID or CSD for the long response format. Note that the start bit, transmitter bit and the six reserved bits are not used in the CRC calculation. The CRC checksum is a 7-bit value:

```
CRC[6:0] = Remainder [(M(x) *x7) / G(x)]

G(x) = x7 + x3 + 1

M(x) = (start bit) * x39 + ... + (last bit before CRC) * x0, or

M(x) = (start bit) * x119 + ... + (last bit before CRC) * x0
```

### 2.2.4 Data path

The data path subunit transfers data to and from cards. Figure 2-9 on page 2-14 shows a block diagram of the data path.

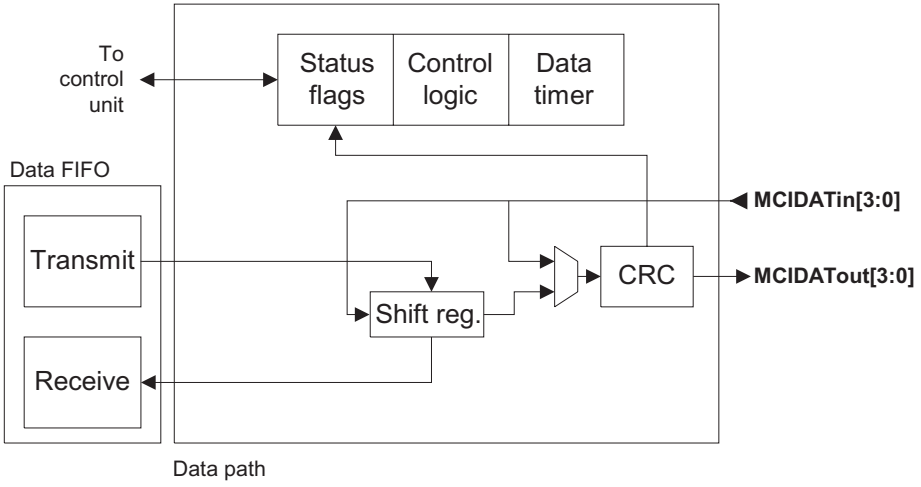



**Figure 2-9 Data path**

You can program the card data bus width using the clock control register. If the wide bus mode is enabled, data is transferred at four bits per clock cycle over all four data signals (**MCIDAT[3:0]**). If the wide bus mode is not enabled, only one bit per clock cycle is transferred over **MCIDAT0**.

Depending on the transfer direction (send or receive), the *Data Path State Machine* (DPSM) moves to the WAIT_S or WAIT_R state when it is enabled:

**Send** The DPSM moves to the WAIT_S state. If there is data in the send FIFO, the DPSM moves to the SEND state, and the data path subunit starts sending data to a card.

**Receive** The DPSM moves to the WAIT_R state and waits for a start bit. When it receives a start bit, the DPSM moves to the RECEIVE state, and the data path subunit starts receiving data from a card

### Data path state machine

The DPSM operates at **MCICLK** frequency. Data on the card bus signals is synchronous to the rising edge of **MCICLK**. The DPSM has six states, as shown in Figure 2-10 on page 2-15.

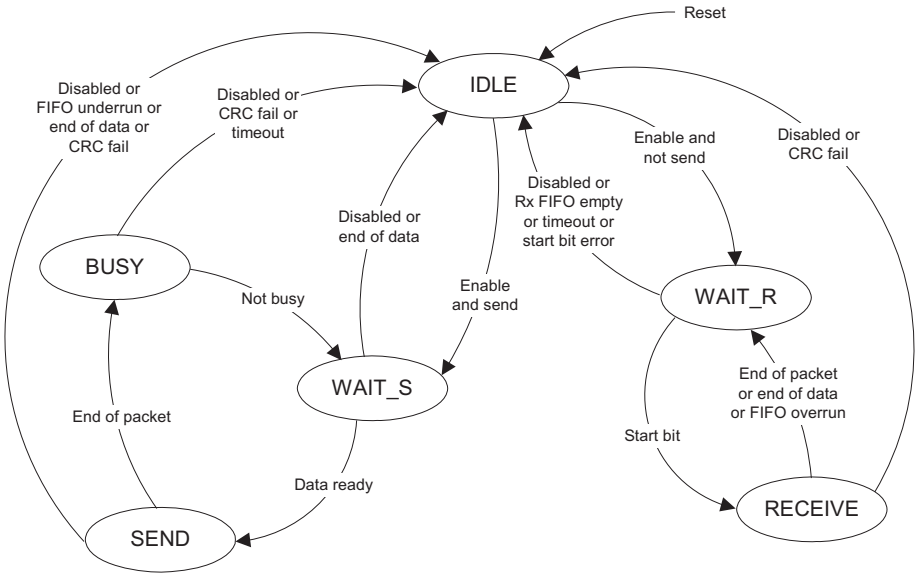



**Figure 2-10 Data path state machine**

**IDLE** The data path is inactive, and the **MCIDAT[3:0]** outputs are in HI-Z. When the data control register is written and the enable bit is set, the DPSM loads the data counter with a new value and, depending on the data direction bit, moves to either the WAIT_S or WAIT_R state.

**WAIT_R** If the data counter equals zero, the DPSM moves to the IDLE state when the receive FIFO is empty. If the data counter is not zero, the DPSM waits for a start bit on **MCIDAT**.

The DPSM moves to the RECEIVE state if it receives a start bit before a timeout, and loads the data block counter. If it reaches a timeout before it detects a start bit, or a start bit error occurs, it moves to the IDLE state and sets the timeout status flag.

**RECEIVE** Serial data received from a card is packed in bytes and written to the data FIFO. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:

- In *block mode*, when the data block counter reaches zero, the DPSM waits until it receives the CRC code. If the received code matches the internally generated CRC code, the DPSM moves to the WAIT_R state. If not, the CRC fail status flag is set and the DPSM moves to the IDLE state.
- In *stream mode*, the DPSM receives data while the data counter is not zero. When the counter is zero, the remaining data in the shift register is written to the data FIFO, and the DPSM moves to the WAIT-R state.

  If a FIFO overrun error occurs, the DPSM sets the FIFO error flag and moves to the WAIT_R state.

**WAIT_S** The DPSM moves to the IDLE state if the data counter is zero. If not, it waits until the data FIFO empty flag is deasserted, and moves to the SEND state.

**Note**

The DPSM remains in the WAIT_S state for at least two clock periods to meet Nwr timing constraints.

**SEND** The DPSM starts sending data to a card. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:

- In *block mode*, when the data block counter reaches zero, the DPSM sends an internally generated CRC code and end bit, and moves to the BUSY state.
- In *stream mode*, the DPSM sends data to a card while the enable bit is HIGH and the data counter is not zero. It then moves to the IDLE state.

If a FIFO underrun error occurs, the DPSM sets the FIFO error flag and moves to the IDLE state.

**BUSY** The DPSM waits for the CRC status flag:

- If it does not receive a positive CRC status, it moves to the IDLE state and sets the CRC fail status flag.
- If it receives a positive CRC status, it moves to the WAIT_S state if **MCIDAT0** is not LOW (the card is not busy).

If a timeout occurs while the DPSM is in the BUSY state, it sets the data timeout flag and moves to the IDLE state.

The data timer is enabled when the DPSM is in the WAIT_R or BUSY state, and generates the data timeout error:

- When transmitting data, the timeout occurs if the DPSM stays in the BUSY state for longer than the programmed timeout period.
- When receiving data, the timeout occurs if the end of the data is not true, and if the DPSM stays in the WAIT_R state for longer than the programmed timeout period.

### Data counter

The data counter has two functions:

- To stop a data transfer when it reaches zero. This is the end of the data condition.
- To start transferring a pending command (see Figure 2-11). This is used to send the stop command for a stream data transfer.

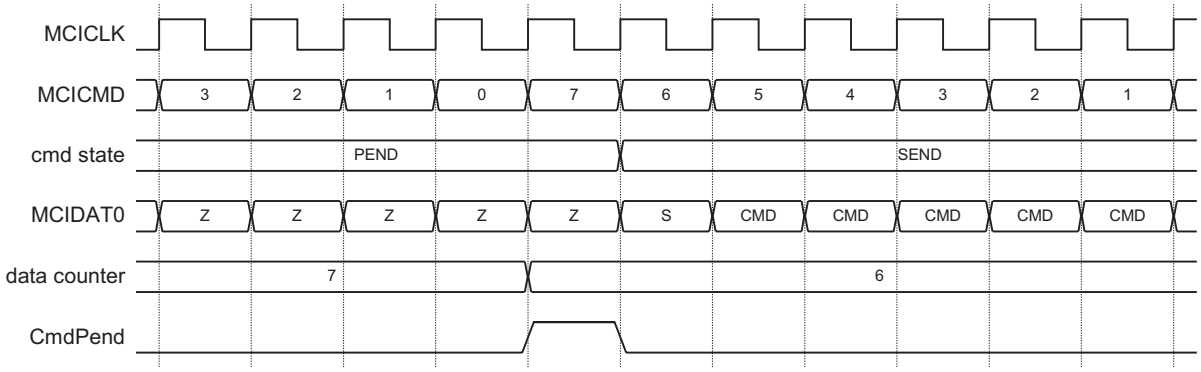



**Figure 2-11 Pending command start**

The data block counter determines the end of a data block. If the counter is zero, the end-of-data condition is TRUE, see *Data control register, MCIDataCtrl* on page 3-11 for more information.

### Bus mode

In wide bus mode, all four data signals (**MCIDAT[3:0]**) are used to transfer data, and the CRC code is calculated separately for each data signal. While transmitting data blocks to a card, only **MCIDAT0** is used for the CRC token and busy signalling. The

start bit must be transmitted on all four data signals at the same time (during the same clock period). If the start bit is not detected on all data signals on the same clock edge while receiving data, the DPSM sets the start bit error flag and moves to the IDLE state.

The data path also operates in half-duplex mode, where data is either sent to a card or received from a card. While not being transferred, **MCIDAT[3:0]** are in the HI-Z state. Data on these signals is synchronous to the rising edge of the clock period.

If you select wide mode, both **nMCIDAT0EN** and **nMCIDATEN** outputs are driven low at the same time. If not, the **MCIDAT[3:1]** outputs are always in HI-Z state (**nMCIDATEN)** is driven HIGH), and only the **MCIDAT0** output is driven LOW when data is transmitted.

### CRC token status

The CRC token status follows each write data block, and determines whether a card has received the data block correctly. When the token has been received, the card asserts a busy signal by driving **MCIDAT0** LOW. Table 2-5 shows the CRC token status values.

**Table 2-5 CRC token status**

| Token | Description |
|---|---|
| 010 | Card has received error-free data block |
| 101 | Card has detected a CRC error |

### Status flags

Table 2-6 lists the data path status flags, see *Status register, MCIStatus* on page 3-12 for more information.

**Table 2-6 Data path status flags**

| Flag | Description |
|---|---|
| TxFifoFull | Transmit FIFO is full |
| TxFifoEmpty | Transmit FIFO is empty |
| TxFifoHalfEmpty | Transmit FIFO is half full |
| TxDataAvlbl | Transmit FIFO data available |
| TxUnderrun | Transmit FIFO underrun error |
| RxFifoFull | Receive FIFO is full |

**Table 2-6 Data path status flags (continued)**

| Flag | Description |
|---|---|
| RxFifoEmpty | Receive FIFO is empty |
| RxFifoHalfFull | Receive FIFO is half full |
| RxDataAvlbl | Receive FIFO data available |
| RxOverrun | Receive FIFO overrun error |
| DataBlockEnd | Data block sent/received |
| StartBitErr | Start bit not detected on all data signals in wide bus mode |
| DataCrcFail | Data packet CRC failed |
| DataEnd | Data end (data counter is zero) |
| DataTimeOut | Data timeout |
| TxActive | Data transmission in progress |
| RxActive | Data reception in progress |

### CRC generator

The CRC generator calculates the CRC checksum only for the data bits in a single block, and is bypassed in data stream mode. The checksum is a 16-bit value:

```
CRC[15:0] = Remainder [(M(x) *x15) / G(x)]

G(x) = x16 + x12 + x5 + 1

M(x) - (first data bit) *xn + ... + (last data bit) *x0
```

## 2.2.5 Data FIFO

The data FIFO (first-in-first-out) subunit is a data buffer with transmit and receive logic. Figure 2-12 on page 2-20 shows a block diagram of the FIFO.

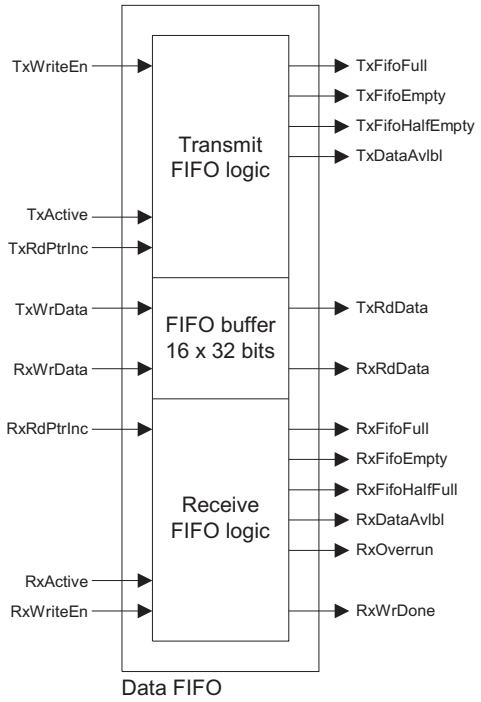



**Figure 2-12 Data FIFO**

The FIFO contains a 32-bit wide, 16-word deep data buffer, and transmit and receive logic. Because the data FIFO operates in the APB clock domain (**PCLK**), all signals from the subunits in the PrimeCell MCI clock domain (**MCLK**) are resynchronized.

Depending on **TxActive** and **RxActive**, the FIFO can be disabled, transmit enabled, or receive enabled. **TxActive** and **RxActive** are driven by the data path subunit and are mutually exclusive:

- The transmit FIFO refers to the transmit logic and data buffer when **TxActive** is asserted (see *Transmit FIFO* on page 2-21)
- The receive FIFO refers to the receive logic and data buffer when **RxActive** is asserted (see *Receive FIFO* on page 2-21).

### Transmit FIFO

Data is written to the transmit FIFO through the APB interface once the MCI is enabled for transmission. When the write signal (**TxWriteEn**) is asserted, data can be written (on the rising edge of **PCLK**) into the FIFO location specified by the current value of the data pointer. The pointer is incremented after every FIFO write.

The transmit FIFO contains a data output register. This holds the data word pointed to by the read pointer. When the data path subunit has loaded its shift register, the data path logic asserts **TxRdPtrInc**. This signal is synchronized with **PCLK**, and increments the read pointer and drives new data on the **TxRdData** output.

If the transmit FIFO is disabled, all status flags are deasserted, and the read and write pointers are reset. The data path subunit asserts **TxActive** when it transmits data. Table 2-7 lists the transmit FIFO status flags.

**Table 2-7 Transmit FIFO status flags**

| Flag | Description |
|---|---|
| TxFifoFull | Set to HIGH when all 16 transmit FIFO words contain valid data. |
| TxFifoEmpty | Set to HIGH when the transmit FIFO does not contain valid data. |
| TxHalfEmpty | Set to HIGH when 8 or more transmit FIFO words are empty. This flag can be used as a DMA request. |
| TxDataAvlbl | Set to HIGH when the transmit FIFO contains valid data. This flag is the inverse of the **TxFifoEmpty** flag. |
| TxUnderrun | Set to HIGH when an underrun error occurs. This flag is cleared by writing to the MCIClear register. |

### Receive FIFO

When the data path subunit receives a word of data, it drives data on the write data bus and asserts the write enable signal. This signal is synchronized to the **PCLK** domain. The write pointer is incremented after the write is completed, and the receive FIFO control logic asserts **RxWrDone**, that then deasserts the write enable signal.

On the read side, the content of the FIFO word pointed to by the current value of the read pointer is driven on the read data bus. The read pointer is incremented when the APB bus interface asserts **RxRdPrtInc**.

If the receive FIFO is disabled, all status flags are deasserted, and the read and write pointers are reset. The data path subunit asserts **RxActive** when it receives data. Table 2-8 lists the receive FIFO status flags.

**Table 2-8 Receive FIFO status flags**

| Flag | Description |
|---|---|
| RxFifoFull | Set to HIGH when all 16 receive FIFO words contain valid data. |
| RxFifoEmpty | Set to HIGH when the receive FIFO does not contain valid data. |
| RxHalfFull | Set to HIGH when 8 or more receive FIFO words contain valid data. This flag can be used as a DMA request. |
| RxDataAvlbl | Set to HIGH when the receive FIFO is not empty. This flag is the inverse of the **RxFifoEmpty** flag. |
| RxOverrun | Set to HIGH when an overrun error occurs. This flag is cleared by writing to the MCIClear register. |

## 2.3 APB interface

Figure 2-13 shows a block diagram of the APB interface.

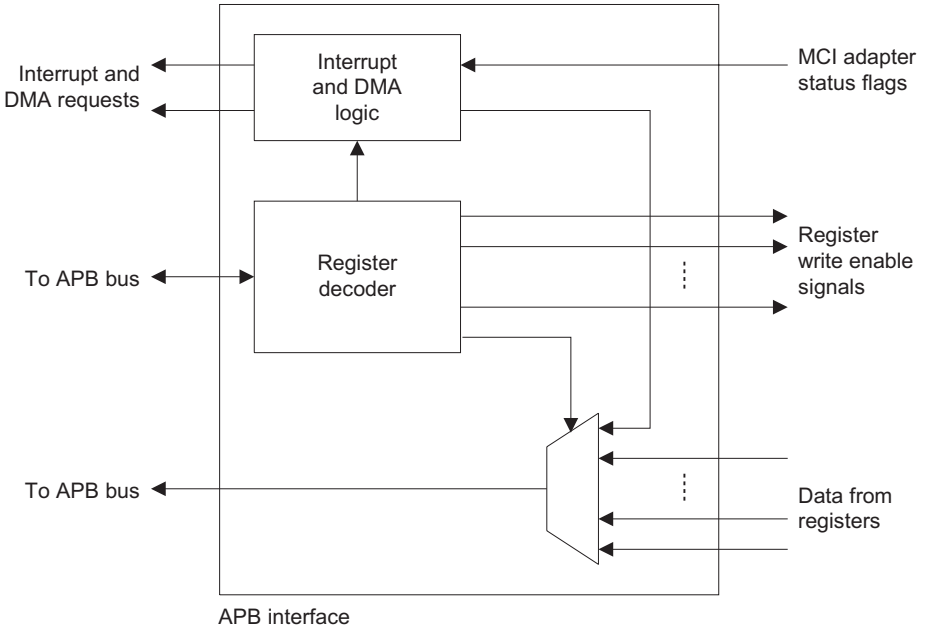



**Figure 2-13 APB Interface**

The APB interface generates the interrupt and DMA requests, and accesses the PrimeCell MCI adapter registers and the data FIFO. It consists of a data path, register decoder, and interrupt/DMA logic.

### 2.3.1 Interrupt logic

The interrupt logic (see Figure 2-14 on page 2-24) generates two interrupt request signals, that are asserted when at least one of the selected status flags is HIGH. A status flag generates the interrupt request if a corresponding mask flag is set. You can assert the interrupt request even if **PCLK** is disabled.

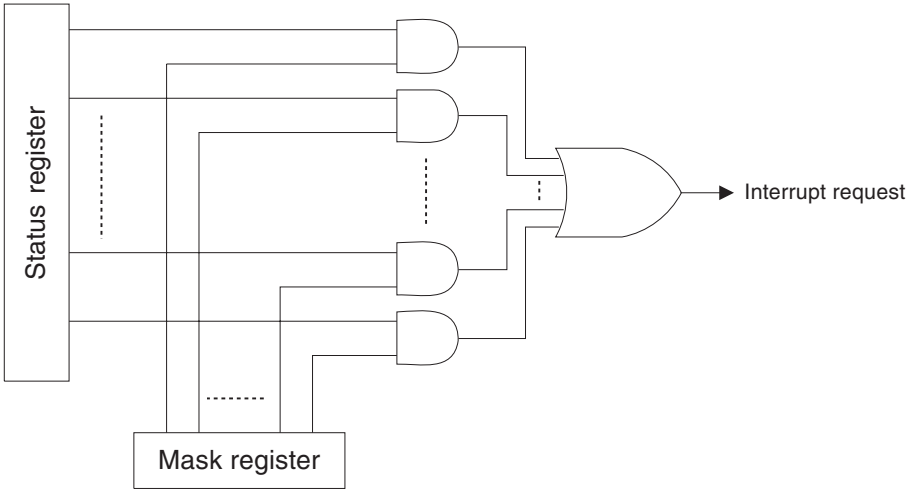



**Figure 2-14 Interrupt request logic**

**Note**

A separate mask register is provided for each interrupt request signal (see *Interrupt mask registers, MCIMask0-1* on page 3-14 for more information).

### 2.3.2 DMA

The interface to the DMA controller includes the signals described in Table 2-9.

**Table 2-9 DMA controller interface signals**

| Signal | Type | Description |
|---|---|---|
| **DMASREQ** | Single word DMA transfer request, asserted by PrimeCell MCI | For receive: Asserted if data counter is zero and receive FIFO contains more than one and fewer than eight words.<br>For transmit: Asserted if fewer than eight and more than one word remain for transfer to FIFO. |
| **DMABREQ** | Burst DMA transfer request, asserted by PrimeCell MCI | For receive: Asserted if FIFO contains eight words and data counter is not zero, or if FIFO contains more than eight words.<br>For transmit: Asserted if more than eight words remain for transfer to FIFO. |
| **DMALSREQ** | Last single word DMA transfer request, asserted by PrimeCell MCI | For receive: Asserted if data counter is zero and FIFO contains only one word.<br>For transmit: Asserted if only one word remains for transfer to FIFO. |
| **DMALBREQ** | Last burst DMA transfer request, asserted by PrimeCell MCI | For receive: Asserted if data counter is zero and FIFO contains eight words.<br>For transmit: Asserted if only eight words remain for transfer to FIFO. |
| **DMACLR** | DMA request clear, asserted by DMA controller to clear request signals | Asserted during transfer of last data in burst if DMA burst transfer is requested. |

Because the four request signals are mutually exclusive, only one signal is asserted at a time. The signal remains asserted until **DMACLR** is asserted. After this, a request signal can be active again, depending on the conditions described in Table 2-9. When the Enable bit in the Data Control register is cleared, the data path is disabled and all request signals are de-asserted.

The DMA signals are synchronous with **PCLK**. Figure 2-15 on page 2-26 shows the DMA transfer of the last three words.

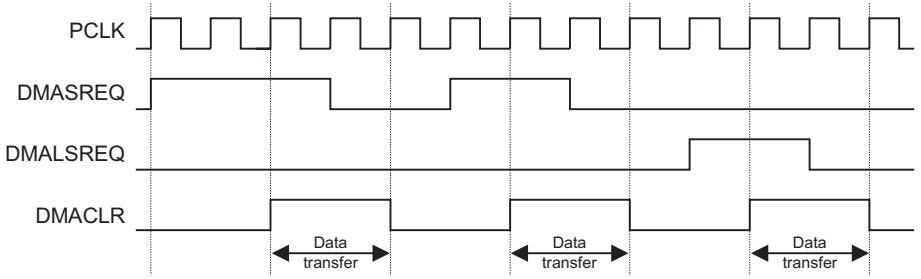



**Figure 2-15 DMA interface**

## 2.4 Timing requirements

The clock output is routed back to the PrimeCell MCI and is used to clock the output registers, to meet the hold time requirements of **MCICMD** and **MCIDATx**. Figure 2-16 shows a block diagram of the clock output routing.

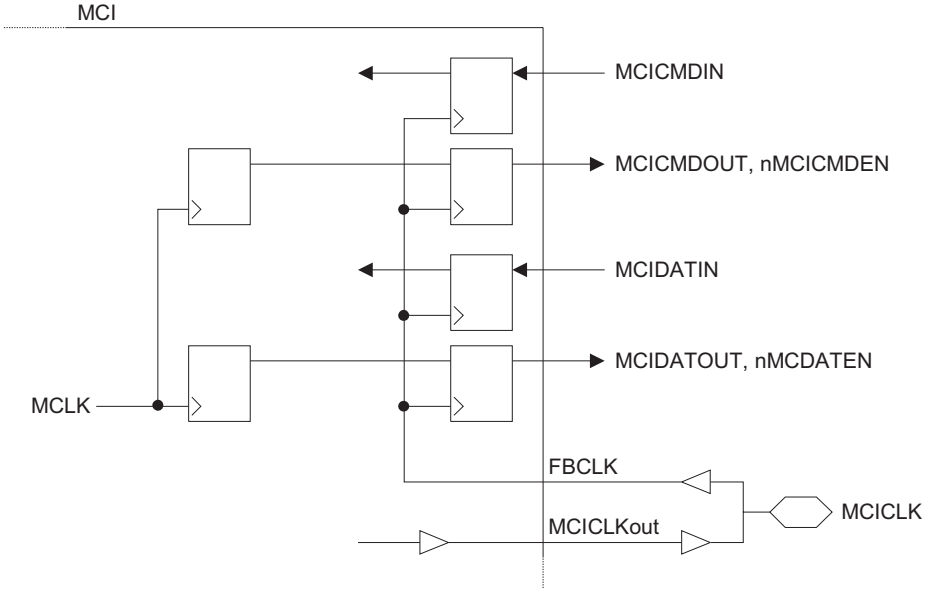



**Figure 2-16 Clock output retiming logic**

Ensure that you meet the input and timing requirements when integrating the PrimeCell MCI in a system. Figure 2-17 shows the signal relationship.

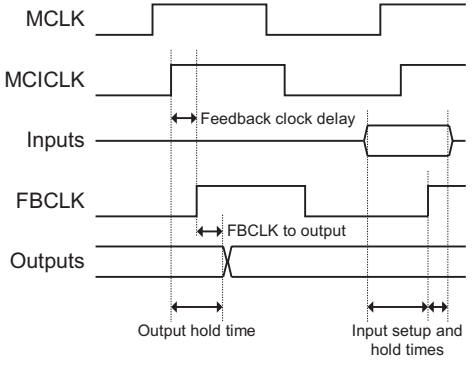



**Figure 2-17 MCICMD and MCIDAT timing**

# Chapter 3
# **Programmer's Model**

This chapter describes the ARM PrimeCell MCI (PL180) registers, and provides details needed when programming the microcontroller. It contains the following sections:

## 3.1 About the programmer's model

The base address of the PrimeCell MCI is not fixed, and can be different for each particular system implementation.

## 3.2 Summary of PrimeCell MCI registers

The PrimeCell MCI registers are shown in Table 3-1.

**Table 3-1 PrimeCell MCI register summary**

| Address | Type | Width | Reset value | Name | Description |
|---|---|---|---|---|---|
| MCI Base + 0x000 | Read/write | 8 | 0x00 | MCIPower | Power control register. |
| MCI Base + 0x004 | Read/write | 12 | 0x000 | MCIClock | Clock control register. |
| MCI Base + 0x008 | Read/write | 32 | 0x00000000 | MCIArgument | Argument register. |
| MCI Base + 0x00C | Read/write | 11 | 0x000 | MMCCommand | Command register. |
| MCI Base + 0x010 | Read only | 6 | 0x00 | MCIRespCmd | Response command register. |
| MCI Base + 0x014 | Read only | 32 | 0x00000000 | MCIResponse0 | Response register. |
| MCI Base + 0x018 | Read only | 32 | 0x00000000 | MCIResponse1 | Response register. |
| MCI Base + 0x01C | Read only | 32 | 0x00000000 | MCIResponse2 | Response register. |
| MCI Base + 0x020 | Read only | 31 | 0x00000000 | MCIResponse3 | Response register. |
| MCI Base + 0x024 | Read/write | 32 | 0x00000000 | MCIDataTimer | Data timer. |
| MCI Base + 0x028 | Read/write | 16 | 0x0000 | MCIDataLength | Data length register. |
| MCI Base + 0x02C | Read/write | 8 | 0x00 | MCIDataCtrl | Data control register. |
| MCI Base + 0x030 | Read only | 16 | 0x0000 | MCIDataCnt | Data counter. |
| MCI Base + 0x034 | Read only | 22 | 0x000000 | MCIStatus | Status register. |
| MCI Base + 0x038 | Write only | 11 | - | MCIClear | Clear register. |
| MCI Base + 0x03C | Read/write | 22 | 0x000000 | MCIMask0 | Interrupt 0 mask register. |
| MCI Base + 0x040 | Read/write | 22 | 0x000000 | MCIMask1 | Interrupt 1 mask register. |
| MCI Base + 0x044 | Read/write | 4 | 0x0 | MCISelect | Secure digital memory card select register. |
| MCI Base + 0x048 | Read only | 15 | 0x0000 | MCIFifoCnt | FIFO counter. |
| MCI Base + 0x04C - 0x07C | - | - | - | Reserved | - |
| MCI Base + 0x080 - 0x0BC | Read/write | 32 | 0x00000000 | MCIFIFO | Data FIFO register. |

**Table 3-1 PrimeCell MCI register summary (continued)**

| Address | Type | Width | Reset value | Name | Description |
|---|---|---|---|---|---|
| MCI Base + 0xFE0 | Read only | 8 | 0x80 | MCIPeriphID0 | Peripheral identification register bits 7:0. |
| MCI Base + 0xFE4 | Read only | 8 | 0x11 | MCIPeriphID1 | Peripheral identification register bits 15:8. |
| MCI Base + 0xFE8 | Read only | 8 | 0x04 | MCIPeriphID2 | Peripheral identification register bits 23:16. |
| MCI Base + 0xFEC | Read only | 8 | 0x00 | MCIPeriphID3 | Peripheral identification register bits 31:24. |
| MCI Base + 0xFF0 | Read only | 8 | 0x0D | MCIPCellID0 | PrimeCell identification register bits 7:0. |
| MCI Base + 0xFF4 | Read only | 8 | 0xF0 | MCIPCellID1 | PrimeCell identification register bits 15:8. |
| MCI Base + 0xFF8 | Read only | 8 | 0x05 | MCIPCellID2 | PrimeCell identification register bits 23:16. |
| MCI Base + 0xFFC | Read only | 8 | 0xB1 | MCIPCellID3 | PrimeCell identification register bits 31:24. |

## 3.3 Register descriptions

The following PrimeCell MCI registers are described in this section:

- *Power control register, MCIPower*
- *Clock control register, MCIClock* on page 3-6
- *Argument register, MCIArgument* on page 3-7
- *Command register, MCICommand* on page 3-7
- *Command response register, MCIRespCommand* on page 3-9
- *Response registers, MCIResponse0-3* on page 3-9
- *Data timer register, MCIDataTimer* on page 3-10
- *Data length register, MCIDataLength* on page 3-10
- *Data control register, MCIDataCtrl* on page 3-11
- *Data counter register, MCIDataCnt* on page 3-12
- *Status register, MCIStatus* on page 3-12
- *Clear register, MCIClear* on page 3-14
- *Interrupt mask registers, MCIMask0-1* on page 3-14
- *Secure digital memory card select register, MCISelect* on page 3-16
- *FIFO counter register, MCIFifoCnt* on page 3-16
- *Data FIFO register, MCIFIFO* on page 3-17
- *Peripheral identification registers, MCIPeriphID0-3* on page 3-17
- *PrimeCell identification registers, MCIPCellID0-3* on page 3-20.

### 3.3.1 Power control register, MCIPower

The MCIPower register controls an external power supply. You can switch the power on and off, and adjust the output voltage. Table 3-2 shows the bit assignment of the MCIPower register.

**Table 3-2 MCIPower register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 1:0 | Ctrl | Read/write | 00 = Power-off<br>01 = Reserved<br>10 = Power-up<br>11 = Power-on |
| 5:2 | Voltage | Read/write | Output voltage |

**Table 3-2 MCIPower register (continued)**

| Bit | Name | Type | Function |
|---|---|---|---|
| 6 | OpenDrain | Read/write | MCICMD output control |
| 7 | Rod | Read/write | Rod control |
| 31:8 | Reserved | - | - |

When you switch the external power supply on, the software first enters the power-up phase, and waits until the supply output is stable before moving to the power-on phase. During the power-up phase, **MCIPWR** is set HIGH. The card bus outlets are disabled during both phases.

You can set the supply output voltage using the voltage value on the **MCIVDD** outputs. Because the operating voltage range can be any value between 2.0 and 3.6 volts, the encoding of the voltage bits in the power control register is application-specific.

**Note**

After a data write, data cannot be written to this register for three MCLK clock periods plus two PCLK clock periods.

### 3.3.2 Clock control register, MCIClock

The MCIClock register controls the **MCICLK** output. Table 3-3 shows the bit assignment of the clock control register.

**Table 3-3 MCIClock register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 7:0 | ClkDiv | Read/write | PrimeCell MCI bus clock period:<br>**MCLCLK** frequency = **MCLK** / [2x(ClkDiv+1)]. |
| 8 | Enable | Read/write | Enable PrimeCell MCI bus clock:<br>0 = Clock disabled<br>1 = Clock enabled. |
| 9 | PwrSave | Read/write | Disable PrimeCell MCI clock output when bus is idle:<br>0 = Always enabled<br>1 = Clock enabled when bus is active. |

**Table 3-3 MCIClock register (continued)**

| Bit | Name | Type | Function |
|---|---|---|---|
| 10 | Bypass | Read/write | Enable bypass of clock divide logic:<br>0 = Disable bypass<br>1 = Enable bypass (**MCLK** driven to card bus output (**MCICLK**)). |
| 11 | WideBus | Read/write | Enable wide bus mode:<br>0 = Standard bus mode (only **MCIDAT0** used)<br>1 = Wide bus mode (**MCIDAT3:0** used). |
| 31:12 | Reserved | - | - |

While the PrimeCell MCI is in identification mode, the **MCICLK** frequency must be less than 400 kHz. You can change the clock frequency to the maximum card bus frequency when relative card addresses are assigned to all cards.

**Note**

After a data write, data cannot be written to this register for three MCLK clock periods plus two PCLK clock periods.

### 3.3.3 Argument register, MCIArgument

The MCIArgument register contains a 32-bit command argument, which is sent to a card as part of a command message. Table 3-4 shows the bit assignment of the MCIArgument register.

**Table 3-4 MCIArgument register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 31:0 | CmdArg | Read/write | Command argument |

If a command contains an argument, it must be loaded into the argument register before writing a command to the command register.

### 3.3.4 Command register, MCICommand

The MCICommand register contains the command index and command type bits:

- The command index is sent to a card as part of a command message

- The command type bits control the *Command Path State Machine* (CPSM). Writing 1 to the enable bit starts the command send operation, while clearing the bit disables the CPSM.

Table 3-5 shows the bit assignment of the MCICommand register.

**Table 3-5 MCICommand register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 5:0 | CmdIndex | Read/write | Command index |
| 6 | Response | Read/write | If set, CPSM waits for a response |
| 7 | LongRsp | Read/write | If set, CPSM receives a 136-bit long response |
| 8 | Interrupt | Read/write | If set, CPSM disables command timer and waits for interrupt request |
| 9 | Pending | Read/write | If set, CPSM waits for CmdPend before it starts sending a command |
| 10 | Enable | Read/write | If set, CPSM is enabled |
| 31:11 | Reserved | - | - |

**Note**

After a data write, data cannot be written to this register for three MCLK clock periods plus two PCLK clock periods.

Table 3-6 shows the response types.

**Table 3-6 Command response types**

| Response | LongRsp. | Description |
|---|---|---|
| 0 | 0 | No response, expect CmdSent flag |
| 0 | 1 | No response, expect CmdSent flag |
| 1 | 0 | Short response, expect CmdRespEnd or CmdCrcFail flag |
| 1 | 1 | Long response, expect CmdRespEnd or CmdCrcFail flag |

### 3.3.5 Command response register, MCIRespCommand

The MCIRespCommand register contains the command index field of the last command response received. Table 3-7 shows the bit assignment of the MCIRespCommand register.

**Table 3-7 MCIRespCommand register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 5:0 | RespCmd | Read | Response command index |
| 31:6 | Reserved | - | - |

If the command response transmission does not contain the command index field (long response), the RespCmd field is unknown, although it must contain 111111 (the value of the reserved field from the response).

### 3.3.6 Response registers, MCIResponse0-3

The MCIResponse0-3 registers contain the status of a card, which is part of the received response. Table 3-8 shows the bit assignment of the MCIResponse0-3 registers.

**Table 3-8 MCIResponse0-3 registers**

| Bit | Name | Type | Function |
|---|---|---|---|
| 31:0 | Status | Read | Card status |

The card status size can be 32 or 127 bits, depending on the response type. see Table 3-8.

**Table 3-9 Response register type**

| Description | Short response | Long response |
|---|---|---|
| MCIResponse0 | Card status [31:0] | Card status [127:96] |
| MCIResponse1 | Unused | Card status [95:64] |
| MCIResponse2 | Unused | Card status [63:32] |
| MCIResponse3 | Unused | Card status [31:1] |

The most significant bit of the card status is received first. The MCIResponse3 register LSBit is always 0.

### 3.3.7 Data timer register, MCIDataTimer

The MCIDataTimer register contains the data timeout period, in card bus clock periods. Table 3-10 shows the bit assignment of the MCIDataTimer register.

**Table 3-10 MCIDataTimer register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 31:0 | DataTime | Read/write | Data timeout period |

A counter loads the value from the data timer register, and starts decrementing when the *Data Path State Machine* (DPSM) enters the WAIT_R or BUSY state. If the timer reaches 0 while the DPSM is in either of these states, the timeout status flag is set.

A data transfer must be written to the data timer register and the data length register before being written to the data control register.

### 3.3.8 Data length register, MCIDataLength

The MCIDataLength register contains the number of data bytes to be transferred. The value is loaded into the data counter when data transfer starts. Table 3-11 shows the bit assignment of the MCIDataLength register.

**Table 3-11 MCIDataLength register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 15:0 | DataLength | Read/write | Data length value |
| 31:16 | Reserved | - | - |

For a block data transfer, the value in the data length register must be a multiple of the block size (see *Data control register, MCIDataCtrl* on page 3-11).

A data transfer must be written to the data timer register and the data length register before being written to the data control register.

### 3.3.9 Data control register, MCIDataCtrl

The MCIDataCtrl register controls the DPSM. Table 3-12 shows the bit assignment of the MCIDataCtrl register.

**Table 3-12 MCIDataCtrl register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 0 | Enable | Read/write | Data transfer enabled |
| 1 | Direction | Read/write | Data transfer direction:<br>0 = From controller to card<br>1 = From card to controller |
| 2 | Mode | Read/write | Data transfer mode:<br>0 = Block data transfer<br>1 = Stream data transfer |
| 3 | DMAEnable | Read/write | Enable DMA:<br>0 = DMA disabled<br>1 = DMA enabled. |
| 7:4 | BlockSize | Read/write | Data block length |
| 31:8 | Reserved | - | - |

**Note**

After a data write, data cannot be written to this register for three MCLK clock periods plus two PCLK clock periods.

Data transfer starts if 1 is written to the enable bit. Depending on the direction bit, the DPSM moves to the WAIT_S or WAIT_R state. You do not need to clear the enable bit after data transfer. Table 3-13 shows the data block length if block data transfer mode is selected.

**Table 3-13 Data block length**

| Block size | Block length |
|---|---|
| 0 | $2^0$ = 1 byte |
| 1 | $2^1$ = 2 bytes |

**Table 3-13 Data block length (continued)**

| Block size | Block length |
|---|---|
| ... | - |
| 11 | $2^{11}$ = 2048 bytes |
| 12:15 | Reserved |

### 3.3.10 Data counter register, MCIDataCnt

The MCIDataCnt register loads the value from the data length register (see *Data length register, MCIDataLength* on page 3-10) when the DPSM moves from the IDLE state to the WAIT_R or WAIT_S state. As data is transferred, the counter decrements the value until it reaches 0. The DPSM then moves to the IDLE state and the data status end flag is set. Table 3-14 shows the bit assignment of the MCIDataCnt register.

**Table 3-14 MCIDataCnt register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 15:0 | DataCount | Read | Remaining data |
| 31:16 | Reserved | - | - |

**Note**

This register should be read only when the data transfer is complete.

### 3.3.11 Status register, MCIStatus

The MCIStatus register is a read-only register. It contains two types of flag:

**Static [10:0]** These remain asserted until they are cleared by writing to the Clear register (see *Clear register, MCIClear* on page 3-14).

**Dynamic [21:11]** These change state depending on the state of the underlying logic (for example, FIFO full and empty flags are asserted and deasserted as data while written to the FIFO).

Table 3-15 shows the bit assignment of the MCIStatus register.

**Table 3-15 MCIStatus register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 0 | CmdCrcFail | Read | Command response received (CRC check failed) |
| 1 | DataCrcFail | Read | Data block sent/received (CRC check failed) |
| 2 | CmdTimeOut | Read | Command response timeout |
| 3 | DataTimeOut | Read | Data timeout |
| 4 | TxUnderrun | Read | Transmit FIFO underrun error |
| 5 | RxOverrun | Read | Receive FIFO overrun error |
| 6 | CmdRespEnd | Read | Command response received (CRC check passed) |
| 7 | CmdSent | Read | Command sent (no response required) |
| 8 | DataEnd | Read | Data end (data counter is zero) |
| 9 | StartBitErr | Read | Start bit not detected on all data signals in wide bus mode |
| 10 | DataBlockEnd | Read | Data block sent/received (CRC check passed) |
| 11 | CmdActive | Read | Command transfer in progress |
| 12 | TxActive | Read | Data transmit in progress |
| 13 | RxActive | Read | Data receive in progress |
| 14 | TxFifoHalfEmpty | Read | Transmit FIFO half empty |
| 15 | RxFifoHalfFull | Read | Receive FIFO half full |
| 16 | TxFifoFull | Read | Transmit FIFO full |
| 17 | RxFifoFull | Read | Receive FIFO full |
| 18 | TxFifoEmpty | Read | Transmit FIFO empty |
| 19 | RxFifoEmpty | Read | Receive FIFO empty |
| 20 | TxDataAvlbl | Read | Data available in transmit FIFO |
| 21 | RxDataAvlbl | Read | Data available in receive FIFO |
| 31:22 | Reserved | - | - |

### 3.3.12 Clear register, MCIClear

The MCIClear register is a write-only register. The corresponding static status flags can be cleared by writing a 1 to the corresponding bit in the register. Table 3-16 shows the bit assignment of the MCIClear register.

**Table 3-16 MCIClear register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 0 | CmdCrcFailClr | Write | Clears CmdCrcFail flag |
| 1 | DataCrcFailClr | Write | Clears DataCrcFail flag |
| 2 | CmdTimeOutClr | Write | Clears CmdTimeOut flag |
| 3 | DataTimeOutClr | Write | Clears DataTimeOut flag |
| 4 | TxUnderrunClr | Write | Clears TxUnderrun flag |
| 5 | RxOverrunClr | Write | Clears RxOverrun flag |
| 6 | CmdRespEndClr | Write | Clears CmdRespEnd flag |
| 7 | CmdSentClr | Write | Clears CmdSent flag |
| 8 | DataEndClr | Write | Clears DataEnd flag |
| 9 | StartBitErrClr | Write | Clears StartBitErr flag |
| 10 | DataBlockEndClr | Write | Clears DataBlockEnd flag |
| 31:11 | Reserved | - | - |

### 3.3.13 Interrupt mask registers, MCIMask0-1

There are two interrupt mask registers, MCIMask0-1, one for each interrupt request signal. Table 3-17 shows the bit assignment of the MCIMask0-1 registers.

**Table 3-17 MCIMask0-1 registers**

| Bit | Name | Type | Function |
|---|---|---|---|
| 0 | Mask0 | Read/write | Mask CmdCrcFail flag |
| 1 | Mask1 | Read/write | Mask DataCrcFail flag |
| 2 | Mask2 | Read/write | Mask CmdTimeOut flag |
| 3 | Mask3 | Read/write | Mask DataTimeOut flag |
| 4 | Mask4 | Read/write | Mask TxUnderrun flag |

**Table 3-17 MCIMask0-1 registers (continued)**

| Bit | Name | Type | Function |
|---|---|---|---|
| 5 | Mask5 | Read/write | Mask RxOverrun flag |
| 6 | Mask6 | Read/write | Mask CmdRespEnd flag |
| 7 | Mask7 | Read/write | Mask CmdSent flag |
| 8 | Mask8 | Read/write | Mask DataEnd flag |
| 9 | Mask9 | Read/write | Mask StartBitErr flag |
| 10 | Mask10 | Read/write | Mask DataBlockEnd flag |
| 11 | Mask11 | Read/write | Mask CmdActive flag |
| 12 | Mask12 | Read/write | Mask TxActive flag |
| 13 | Mask13 | Read/write | Mask RxActive flag |
| 14 | Mask14 | Read/write | Mask TxFifoHalfEmpty flag |
| 15 | Mask15 | Read/write | Mask RxFifoHalfFull flag |
| 16 | Mask16 | Read/write | Mask TxFifoFull flag |
| 17 | Mask17 | Read/write | Mask RxFifoFull flag |
| 18 | Mask18 | Read/write | Mask TxFifoEmpty flag |
| 19 | Mask19 | Read/write | Mask RxFifoEmpty flag |
| 20 | Mask20 | Read/write | Mask TxDataAvlbl flag |
| 21 | Mask21 | Read/write | Mask RxDataAvlbl flag |
| 31:22 | Reserved | - | - |

The interrupt mask registers determine which status flags generate an interrupt request by setting the corresponding bit to 1.

### 3.3.14 Secure digital memory card select register, MCISelect

The MCISelect register selects one of the secure digital memory cards on the bus. If the system supports more than one secure digital memory card, the cards are selected (or addressed) by writing the card address to the register. This address corresponds to the physical connection of the card to the controller. Table 3-18 shows the bit assignment of the MCISelect register.

**Table 3-18 MCISelect register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 3:0 | SDCard | Read/write | Secure digital memory card address |
| 31:4 | Reserved | - | - |

**Note**

Use this register only if the controller is used in the secure digital memory card system.

### 3.3.15 FIFO counter register, MCIFifoCnt

The MCIFifoCnt register contains the remaining number of words to be written to or read from the FIFO. The FIFO counter loads the value from the data length register (see *Data length register, MCIDataLength* on page 3-10) when the Enable bit is set in the data control register. If the data length is not word aligned (multiple of 4), the remaining 1 to 3 bytes are regarded as a word. Table 3-19 shows the bit assignment of the MCIFifoCnt register.

**Table 3-19 MCIFifoCnt register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 14:0 | DataCount | Read | Remaining data |
| 31:15 | Reserved | - | - |

### 3.3.16 Data FIFO register, MCIFIFO

The receive and transmit FIFOs can be read or written as 32-bit wide registers. The FIFOs contain 16 entries on 16 sequential addresses. This allows the microprocessor to use its load and store multiple operands to read/write to the FIFO. Table 3-20 shows the bit assignment of the MCIFIFO register.

**Table 3-20 MCIFIFO register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 31:0 | Data | Read/write | FIFO data |

### 3.3.17 Peripheral identification registers, MCIPeriphID0-3

The MCIPeriphID0-3 registers are four 8-bit registers, that span address locations `0xFE0-0xFEC`. The registers can conceptually be treated as a single 32-bit register. The read-only registers provide the following options of the peripheral:

**Part number [11:0]** This is used to identify the peripheral. The three digit product code 180 is used for the PrimeCell MCI.

**Designer [19:12]** This is the identification of the designer. ARM is `0x4` (ASCII A).

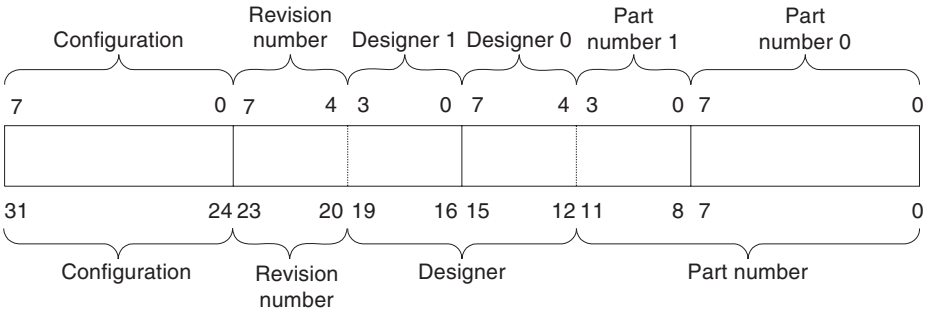
**Revision number [23:20]**

This is the revision number of the peripheral. The revision number starts from 0.

**Configuration [31:24]**

This is the configuration option of the peripheral. The configuration value is 0.

Figure 3-1 on page 3-18 shows the bit assignment for the MCIPeriphID0-3 registers.




Actual register bit assignment

Conceptual register bit assignment

**Figure 3-1 Peripheral identification register bit assignment**

**Note**

When you design a systems memory map you must remember that the register has a 4KB-memory footprint.

The 4-bit revision number is implemented by instantiating a component called RevisionAnd four times with its inputs tied off as appropriate, and the output sent to the read multiplexor.

All memory accesses to the peripheral identification registers must be 32-bit, using the LDR and STR instructions.

The four, 8-bit peripheral identification registers are described in the following subsections:

- *MCIPeriphID0 register* on page 3-19
- *MCIPeriphID1 register* on page 3-19
- *MCIPeriphID2 register* on page 3-19
- *MCIPeriphID3 register* on page 3-20.

### MCIPeriphID0 register

The MCIPeriphID0 register is hard coded and the fields within the registers determine the reset value. Table 3-21 shows the bit assignment of the MCIPeriphID0 register.

**Table 3-21 MCIPeriphID0 register**

| Bits | Name | Type | Function |
|---|---|---|---|
| 31:8 | - | - | Reserved, read undefined, must be written as zeros. |
| 7:0 | Partnumber0 | Read | These bits read back as `0x80`. |

### MCIPeriphID1 register

The MCIPeriphID1 register is hard coded and the fields within the registers determine the reset value. Table 3-22 shows the bit assignment of the MCIPeriphID1 register.

**Table 3-22 MCIPeriphID1 register**

| Bits | Name | Type | Function |
|---|---|---|---|
| 31:8 | - | - | Reserved, read undefined, must be written as zeros. |
| 7:4 | Designer0 | Read | These bits read back as `0x1`. |
| 3:0 | Partnumber1 | Read | These bits read back as `0x1`. |

### MCIPeriphID2 register

The MCIPeriphID2 register is hard coded and the fields within the registers determine the reset value. Table 3-23 shows the bit assignment of the MCIPeriphID2 register.

**Table 3-23 MCIPeriphID2 register**

| Bits | Name | Type | Function |
|---|---|---|---|
| 31:8 | - | - | Reserved, read undefined, must be written as zeros. |
| 7:4 | Revision | Read | These bits read back as `0x0`. |
| 3:0 | Designer1 | Read | These bits read back as `0x4`. |

### MCIPeriphID3 register

The MCIPeriphID3 register is hard coded and the fields within the registers determine the reset value. Table 3-24 shows the bit assignment of the MCIPeriphID3 register.

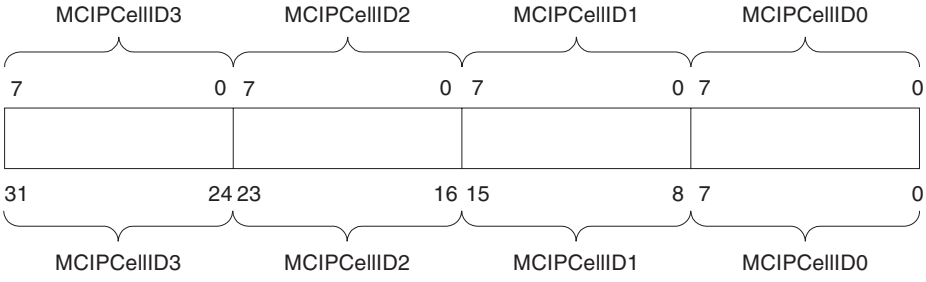
**Table 3-24 MCIPeriphID3 register**

| Bits | Name | Type | Function |
|---|---|---|---|
| 31:8 | - | - | Reserved, read undefined, must be written as zeros. |
| 7:0 | Configuration | Read | These bits read back as `0x0`. |

## 3.3.18 PrimeCell identification registers, MCIPCellID0-3

The MCIPCellID0-3 registers are four 8-bit registers, that span address locations `0xFF0-0xFFC`. The read-only registers can conceptually be treated as a single 32-bit register. The register is used as a standard cross-peripheral identification system. Figure 3-2 shows the bit assignment for the MCIPCellID0-3 registers.

Actual register bit assignment




Conceptual register bit assignment

**Figure 3-2 PrimeCell identification register bit assignment**

The four, 8-bit registers are described in the following subsections:

- *MCIPCellID0 register* on page 3-21
- *MCIPCellID1 register* on page 3-21
- *MCIPCellID2 register* on page 3-21
- *MCIPCellID3 register* on page 3-22.

### MCIPCellID0 register

The MCIPCellID0 register is hard coded and the fields within the registers determine the reset value. Table 3-25 shows the bit assignment of the MCIPCellID0 register.

**Table 3-25 MCIPCellID0 register**

| Bits | Name | Type | Function |
|---|---|---|---|
| 31:8 | - | - | Reserved, read undefined, must be written as zeros. |
| 7:0 | MCIPCellID0 | Read | These bits read back as `0x0D`. |

### MCIPCellID1 register

The MCIPCellID1 register is hard coded and the fields within the registers determine the reset value. Table 3-26 shows the bit assignment of the MCIPCellID1 register.

**Table 3-26 MCIPCellID1 register**

| Bits | Name | Type | Function |
|---|---|---|---|
| 31:8 | - | - | Reserved, read undefined, must be written as zeros. |
| 7:0 | MCIPCellID1 | Read | These bits read back as `0xF0`. |

### MCIPCellID2 register

The MCIPCellID2 register is hard coded and the fields within the registers determine the reset value. Table 3-27 shows the bit assignment of the MCIPCellID2 register.

**Table 3-27 MCIPCellID2 register**

| Bits | Name | Type | Function |
|---|---|---|---|
| 31:8 | - | - | Reserved, read undefined, must be written as zeros. |
| 7:0 | MCIPCellID2 | Read | These bits read back as `0x05`. |

### MCIPCellID3 register

The MCIPCellID3 register is hard coded and the fields within the registers determine the reset value. Table 3-28 shows the bit assignment of the MCIPCellID3 register.

**Table 3-28 MCIPCellID3 register**

| Bits | Name | Type | Function |
|---|---|---|---|
| 31:8 | - | - | Reserved, read undefined, must be written as zeros. |
| 7:0 | MCIPCellID3 | Read | These bits read back as `0xB1`. |

# Chapter 4
# **Programmer's Model for Test**

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

## 4.1 PrimeCell MCI test harness overview

The additional logic for functional verification and integration vectors allows:

- capture of input signals to the block
- stimulation of the output signals.

The integration vectors provide a way of verifying that the PrimeCell MCI is correctly wired into a system. This is done by separately testing three groups of signals:

**AMBA signals** These are tested by checking the connections of all the address and data bits.

**Primary input/output signals**

These are tested using a simple trickbox that can demonstrate the correct connection of the input/output signals to external pads.

**Intra-chip signals (such as interrupt sources)**

The tests for these signals are system-specific, and enable you to write the necessary tests. Additional logic is implemented allowing you to read and write to each intra-chip input/output signal.

These test features are controlled by test registers. This allows you to test the PrimeCell MCI in isolation from the rest of the system using only transfers from the AMBA APB.

Off-chip test vectors are supplied using a 32-bit parallel *External Bus Interface* (EBI) and converted to internal AMBA bus transfers. The application of test vectors is controlled through the *Test Interface Controller* (TIC) AMBA bus master module.

## 4.2 Scan testing

The PrimeCell MCI has been designed to simplify:

- insertion of scan test cells
- use of *Automatic Test Pattern Generation* (ATPG).

This provides an alternative method of manufacturing test.

## 4.3 Test registers

The PrimeCell MCI test registers are memory-mapped as shown in Table 4-1.

**Table 4-1 Test registers memory map**

| Address | Type | Width | Reset value | Name | Description |
|---|---|---|---|---|---|
| MCIBase + 0x100 | Read/write | 4 | 0000 | MCITCR | Test control register |
| MCIBase + 0x104 | Read/write | 6/1 | 000000 | MCIITIP | Integration test input register |
| MCIBase + 0x108 | Read/write | 12 | 0x000 | MCIITOP | Integration test output register |

Each register shown in Table 4-1 is described below.

### 4.3.1 Test control register, MCITCR

MCITCR is a four-bit test control register. The ITEN bit in this register has two functions:

- forces the multiplexers on the inputs and outputs to use the test values
- enables the multimedia card bus lines (**nMCICMDEN**, **nMCIDAT0EN**, and **nMCIDATEN**), which allows the test data to be available on the pads.

Table 4-2 shows the bit assignment of the MCITCR register.

**Table 4-2 MCITCR register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 31:4 | Reserved | - | - |
| 3 | REGTEST | Read/write | Register Test Bit:<br>0 = default, normal mode.<br>Accesses to the registers are controlled by the hardware protection circuitry.<br>1 = test mode.<br>The hardware protection circuitry is bypassed. Normal Write/Read/Write/ Read tests can be performed independently of the link side. |
| 2:1 | FIFOTEST | Read/write | FIFO Test:<br>00 = default, normal mode.<br>Reads to the DataRegister return data from the read port of the FIFO if the direction control bit is configured for receive. If the direction control bit is configured for transmit, this read has no effect.<br>Writes to the DataRegister write data into the write port of the FIFO if the direction control bit is configured for transmit. If the direction control bit is configured for receive, this write has no effect.<br>01 = test mode.<br>Reads to the DataRegister return data from the read port of the FIFO irrespective of the setting of the direction bit, or whether the PrimeCell MCI is data enabled.<br>Writes to the DataRegister write data into the write Port of the FIFO irrespective of the setting of the direction bit, or whether the PrimeCell MCI is data enabled.<br>10 = reserved.<br>11 = test mode.<br>Reads to the DataRegister return data from the read port of the FIFO irrespective of the setting of the direction bit, or whether the PrimeCell MCI is data enabled.<br>Additionally, this read access automatically generates a Write Access to the write port of the FIFO (the write data pattern is pre-defined in this case).<br>Writes to the DataRegister write data into the write port of the FIFO irrespective of the setting of the direction bit, or whether the PrimeCell MCI is data enabled (the test code will not perform writes when FIFOTEST = 11). |
| 0 | ITEN | Read/write | Integration Test Enable:<br>0 = PrimeCell MCI placed in normal mode.<br>1 = PrimeCell MCI placed in Integration test mode. |

### 4.3.2 Integration test input read/set register, MCIITIP

MCIITIP is six-bit register. Out of these six bits, only the MCIDMACLR bit is writable. All other bits MCIITIP[5:1] are read-only.

The ITEN bit is the control bit for the multiplexor. This is used in the read path of the **MCIDMACLR** intra-chip input. If the ITEN control bit is de-asserted, the **MCIDMACLR** intra-chip input is routed as the internal **MCIDMACLR** input. If not, the stored register value is driven on the internal line. All other read-only bits in the MCIITIP register are directly connected to the primary input pins.

Table 4-3 shows the bit assignment of the MCIITIP register.

**Table 4-3 MCIITIP register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 31:6 | Reserved | - | - |
| 5 | MCICMDIN | Read-only | Reads return the value on the MCICMDIN primary input. |
| 4:1 | MCIDATIN[3:0] | Read-only | Reads return the value on the MCIDATIN primary inputs. |
| 0 | MCIDMACLR | Read/write | Writes specify the value to be driven on the intra-chip MCIDMACLR input in the Integration Test Mode.<br>Reads return the value on MCIDMACLR at the output of the test multiplexor. |

### 4.3.3 Integration test output read/set register, MCIITOP

MCIITOP is a 12-bit register. MCIITOP[5:0] are connected to intra-chip outputs and MCIITOP[11:6] are connected to the primary outputs. The read path is different for the intra-chip and the primary output pins. If the ITEN bit is set for intra-chip output, MCIITOP[5:0] is connected with the actual outputs. If not, these pins are connected with the stored register values.

Table 4-4 on page 4-7 shows the bit assignment for the MCIITOP register

**Table 4-4 IMCIITOP register**

| Bit | Name | Type | Function |
|---|---|---|---|
| 31:12 | Reserved | - | - |
| 11 | MCIPOWER | Read/write | Primary output.<br>Writes specify the value to be driven on the MCIPOWER primary output in the integration test mode.<br>Reads return the value written into this field. |
| 10 | MCICMDOUT | Read/write | Primary output.<br>Writes specify the value to be driven on the MCICMDOUT primary output in the integration test mode.<br>Reads return the value written into this field. |
| 9:6 | MCIDATOUT[3:0] | Read/write | Primary output.<br>Writes specify the value to be driven on the MCIDATOUT[3:0] primary output in the integration test mode.<br>Reads return the value written into this field. |
| 5 | MCIDMALBREQ | Read/write | Intra-chip output.<br>Writes specify the value to be driven on the intra-chip MCIDMALBREQ output in the integration test mode.<br>Reads return the value on MCIDMALBREQ at the output of the test multiplexor. |
| 4 | MCIDMALSREQ | Read/write | Intra-chip output.<br>Writes specify the value to be driven on the intra-chip MCIDMALSREQ output in the integration test mode.<br>Reads return the value on MCIDMALSREQ at the output of the test multiplexor. |
| 3 | MCIDMABREQ | Read/write | Intra-chip output.<br>Writes specify the value to be driven on the intra-chip MCIDMABREQ output in the integration test mode.<br>Reads return the value on MCIDMABREQ at the output of the test multiplexor. |

**Table 4-4 IMCIITOP register (continued)**

| Bit | Name | Type | Function |
|---|---|---|---|
| 2 | MCIDMASREQ | Read/write | Intra-chip output.<br>Writes specify the value to be driven on the intra-chip MCIDMASREQ output in the integration test mode.<br>Reads return the value on MCIDMASREQ at the output of the test multiplexor. |
| 1 | MCIINTR1 | Read/write | Intra-chip output.<br>Writes specify the value to be driven on the intra-chip MCIINTR1 output in the integration test mode.<br>Reads return the value on MCIINTR1 at the output of the test multiplexor. |
| 0 | MCIINTR0 | Read/write | Intra-chip output.<br>Writes specify the value to be driven on the intra-chip MCIINTR0 output in the integration test mode.<br>Reads return the value on MCIINTR0 at the output of the test multiplexor. |

## 4.4 Integration testing of block inputs

The following sections describe the integration testing for the block inputs:

- *Intra-chip inputs*
- *Primary inputs* on page 4-10.

### 4.4.1 Intra-chip inputs

When you run integration tests with the PrimeCell MCI in a standalone test setup:

- Write a 1 to the ITEN bit in the control register. This selects the test path from the MCIITIP[0] register bit to the internal **MCIDMACLR** signal.
- Write a 1 and then a 0 to the MCIITIP[0] register bit and read the same register bit to ensure that the value written is read out.

When you run integration tests with the PrimeCell MCI as part of an integrated system:

- Write a 0 to the ITEN bit in the control register. This selects the normal path from the external MCIDMACLR pin to the internal **MCIDMACLR** signal.
- Write a 1 and then a 0 to the internal test registers of the DMA controller to toggle the MCIDMACLR signal connection between the DMA controller and the PrimeCell MCI. Read from the MCIITIP[0] register bit to verify that the value written into the DMA controller is read out through the PrimeCell MCI.

Figure 4-1 on page 4-10 shows details of the implementation of the input integration test harness.

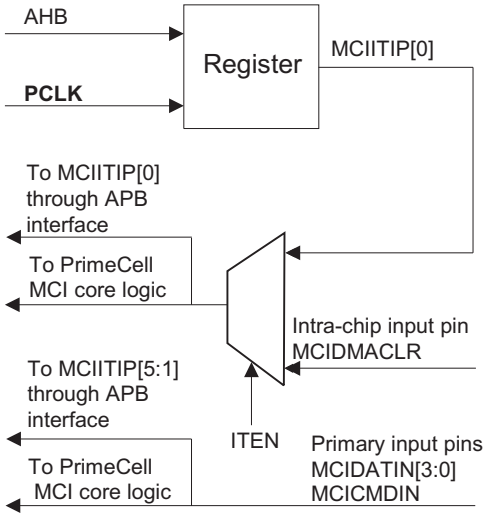



**Figure 4-1 Input integration test harness**

### 4.4.2 Primary inputs

Use this test for the following inputs:

- **MCIDATIN[3:0]**
- **MCICMDIN**
- **MCIFBCLK**.

Test **MCIDATIN[3:1]** using the integration vector trickbox by looping back the **MCIDATOUT[3:1]**. 1s and 0s are driven on to the **MCIDATOUT[3:1]** lines through the MCIITOP register bits [9:7] and read back through the MCIITIP register bits [4:2].

The sequence for testing **MCICMDIN** is detailed in *Primary outputs* on page 4-12.

Test **MCICDATIN[0]** using the integration vector trickbox by looping back the **MCICMDOUT** line. The exact test sequence to verify connectivity of **MCIDATIN[0]** is detailed in *Primary outputs* on page 4-12.

Test **MCIFBCLK** using the integration vector trickbox by looping back the **MCICLKOUT** line.

## 4.5 Integration testing of block outputs

The following sections describe the integration testing for the block outputs:

- *Intra-chip outputs*
- *Primary outputs* on page 4-12.

### 4.5.1 Intra-chip outputs

Use this test for the following outputs:

- **MCIINTR0**
- **MCIINTR1**
- **MCIDMASREQ**
- **MCIDMALREQ**
- **MCIDMALSREQ**
- **MCIDMALBREQ**.

When you run integration tests with the PrimeCell MCI in a standalone test setup:

- Write a 1 to the ITEN bit in the control register. This selects the test path from the MCIITOP[5:0] register bits to the intra-chip output signals.
- Write a 1 and then a 0 to the MCIITOP[5:0] register bits and read the same register bits to ensure that the value written is read out.

When you run integration tests with the PrimeCell MCI as part of an integrated system:

- Write a 1 to the ITEN bit in the control register. This selects the test path from the MCIITOP[5:0] register bits to the intra-chip output signals.
- Write a 1 and then a 0 to the MCIITOP[5:0] register bits to toggle the signal connections between the DMA controller/interrupt controller and the PrimeCell MCI. Read from the internal test registers of the DMA controller/interrupt controller to ensure that the value written into the MCIITOP[5:0] register bits is read out through the PrimeCell MCI.

Figure 4-2 on page 4-12 shows details of the implementation of the output integration test harness in the case of intra-chip outputs.

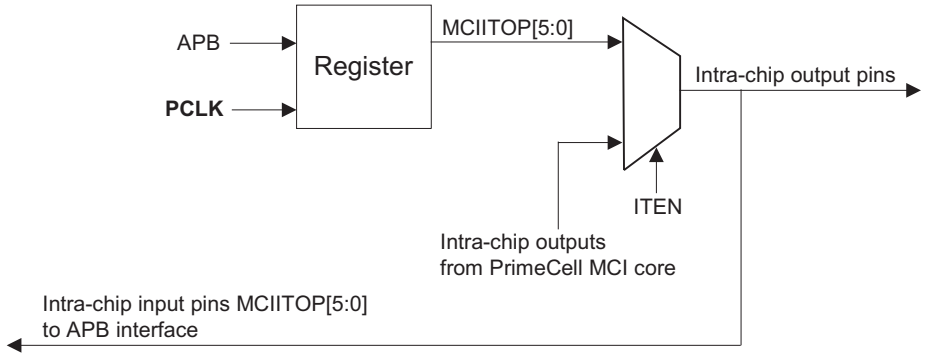



**Figure 4-2 Output integration test harness, intra-chip outputs**

### 4.5.2 Primary outputs

Integration testing of primary outputs and primary inputs is carried out using the integration vector trickbox. Use this test for the following outputs:

- **MCIDATOUT[3:0]**
- **MCICMDOUT**
- **MCICLKOUT**.

Verify the **MCICLKOUT**, **MCIFBCLK**, **MCICMDOUT**, and **MCIDATIN[0]** primary input/output pin connections as follows:

- Primary output **MCICLKOUT** and primary input **MCIFBCLK** are connected inside the integration vector trickbox with a delay element between the lines.
- Primary output **MCICMDOUT** is directly connected to the primary input pin **MCIDATIN[0]**. Enable the **MCICLK** through the MCIPower control register and the MCIClock register. Enable the CPSM in transmit mode and the DPSM in receive mode. The command transmitted by the CPSM must be looped back from **MCICMDOUT** pin to **MCIDATIN[0]** and stored in the FIFO. If the data read from the FIFO matches the transmitted command, the connectivity of **MCICLKOUT**, **MCIFBLCK**, **MCICMDOUT**, and **MCIDATIN[0]** is proven.
- Primary output pins **MCIDATOUT[3:1]** are directly connected to the primary input pins **MCIDATIN[3:1]** using the integration vector trickbox. These pin connections are verified by writing a 1 and a 0 to each of these outputs through the MCIITOP register and reading back the input pins through the MCIITIP register.

Verify the **MCIVDD, MCICARDSEL, MCIROD, MCIPOWER, MCIDATOUT[0]** and **MCICMDIN** pin connections as follows:

- Primary output pins, **MCIDATOUT[0]**, **MCIPWR**, **MCIROD**, **MCICARDSEL[3:0]**, and **MCIVDD[3:0]** are XORed together and routed back to the primary input pin **MCICMDIN** through the integration vector trickbox.
- You can strobe primary outputs **MCIVDD[3:0]** and **MCIROD** through the MCIPower register. You can control **MCICARDSEL[3:0]** through the MCISDSelect register, and you can access all other primary outputs through the MCIITOP register. Different data patterns are written to the output pins using the MCIPower and MCIITOP registers. Read back the XORed data from the **MCICMDIN** pin through the MCIITIP register.

Figure 4-3 shows details of the implementation of the output integration test harness in the case of primary outputs.

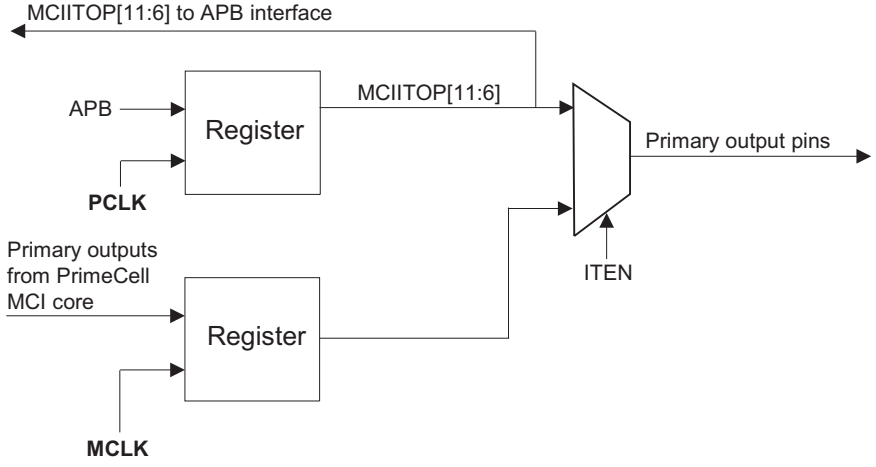



**Figure 4-3 Output integration test harness, primary outputs**

## 4.6 Integration test summary

summarizes the integration test strategy for all PrimeCell MCI pins.

**Table 4-5 PrimeCell MCI integration test strategy**

| Name | Type | Source/ destination | Test strategy |
|---|---|---|---|
| **PCLK** | In | APB | Register read/write. |
| **PRESETn** | In | APB | Register read/write. |
| **PADDR[11:2]** | In | APB | Register read/write. |
| **PSEL** | In | APB | Register read/write. |
| **PENABLE** | In | APB | Register read/write. |
| **PWRITE** | In | APB | Register read/write. |
| **PWDATA[31:0]** | In | APB | Register read/write. |
| **PRDATA[31:0]** | Out | APB | Register read/write. |
| **MCIINTR0** | Out | Intra-chip | Use MCIITOP register. |
| **MCIINTR1** | Out | Intra-chip | Use MCIITOP register. |
| **MCIDMASREQ** | Out | Intra-chip | Use MCIITOP register. |
| **MCIDMABREQ** | Out | Intra-chip | Use MCIITOP register. |
| **MCIDMALSREQ** | Out | Intra-chip | Use MCIITOP register. |
| **MCIDMALBREQ** | Out | Intra-chip | Use MCIITOP register. |
| **MCIDMACLR** | In | Intra-chip | Use MCIITIP register. |
| **MCLK** | In | Primary | Indirectly tested by performing a command-to-data looped back data transfer. |
| **nMCLK** | In | Primary | Not tested using integration test vectors. |
| **nMCIRST** | In | Primary | Indirectly tested by performing a command-to-data looped back data transfer. |
| **MCICLKOUT** | Out | Primary | Use integration vector trickbox and perform a command-to-data looped back data transfer. |
| **MCIFBCLK** | In | Primary | Use integration vector trickbox and perform a command-to-data looped back data transfer. |
| **MCICMDIN** | In | Primary | Use integration vector trickbox, and MCIITIP, MCIITOP, MCISDSelect, and MCIPower registers. |

**Table 4-5 PrimeCell MCI integration test strategy (continued)**

| Name | Type | Source/ destination | Test strategy |
|---|---|---|---|
| **MCICMDOUT** | Out | Primary | Use integration vector trickbox and perform a command-to-data looped back data transfer. |
| **nMCICMDEN** | Out | Primary | Indirectly tested during a data transfer through MCICMD pad. |
| **MCIDATIN[3:1]** | In | Primary | Use integration vector trickbox, and MCIITIP and MCIITOP registers. |
| **MCIDATIN[0]** | In | Primary | Use integration vector trickbox and perform a command-to-data looped back data transfer. |
| **MCIDATOUT[3:1]** | Out | Primary | Use integration vector trickbox, and MCIITIP and MCIITOP registers. |
| **MCIDATOUT[0]** | Out | Primary | Use integration vector trickbox, and MCIITIP, MCIITOP, and MCIPower registers. |
| **nMCIDAT0EN** | Out | Primary | Indirectly tested during a data transfer through MCIDAT0 pad. |
| **nMCIDATEN** | Out | Primary | Indirectly tested during a data transfer through MCIDAT[3:1] pad. |
| **MCIPWR** | Out | Primary | Use integration vector trickbox, and MCIITIP, MCIITOP, MCISDSelect, and MCIPower registers. |
| **MCIVDD[3:0]** | Out | Primary | Use integration vector trickbox, and MCIITIP, MCIITOP, MCISDSelect, and MCIPower registers. |
| **MCIROD** | Out | Primary | Use integration vector trickbox, and MCIITIP, MCIITOP, MCISDSelect, and MCIPower registers. |
| **MCICARDSEL[3:0]** | Out | Primary | Use integration vector trickbox, and MCIITIP, MCIITOP, MCISDSelect, and MCIPower registers. |
| **SCANENABLE** | In | Test controller | Not tested using integration test vectors. |
| **SCANINPCLK** | In | Test controller | Not tested using integration test vectors. |
| **SCANINMCLK** | In | Test controller | Not tested using integration test vectors. |
| **SCANINMCIFBCLK** | In | Test controller | Not tested using integration test vectors. |

**Table 4-5 PrimeCell MCI integration test strategy (continued)**

| Name | Type | Source/ destination | Test strategy |
|---|---|---|---|
| **SCANOUTPCLK** | Out | Test controller | Not tested using integration test vectors. |
| **SCANOUTMCLK** | Out | Test controller | Not tested using integration test vectors. |
| **SCANOUTMCIFBCLK** | Out | Test controller | Not tested using integration test vectors. |

# Appendix A
# ARM PrimeCell MCI (PL180) Signal Descriptions

This appendix describes the signals that interface with the ARM PrimeCell *Multimedia Card Interface* (MCI). It contains the following sections:

## A.1 AMBA APB signals

The PrimeCell MCI module is connected to the AMBA APB as a bus slave. With the exception of the **BnRES** signal, the AMBA APB signals have a **P** prefix and are active HIGH. Active LOW signals contain a lower case **n**. The AMBA APB signals are described in Table A-1.

**Table A-1 AMBA APB signals**

| Name | Type | Source/ destination | Description |
|---|---|---|---|
| **PCLK** | Input | Clock source | APB bus clock |
| **PRESETn** | Input | Reset controller | APB bus reset |
| **PADDR[11:2]** | Input | Master | APB bus address |
| **PSEL** | Input | Master | APB bus peripheral select |
| **PENABLE** | Input | Master | APB bus enable |
| **PWRITE** | Input | Master | APB bus write signal |
| **PWDATA[31:0]** | Input | Master | APB bus write data |
| **PRDATA[31:0]** | Output | Master | APB bus read data |

## A.2 Miscellaneous internal signals

Refer to Table A-2 for an overview of the miscellaneous internal signals.

**Table A-2 Miscellaneous internal signals**

| Name | Type | Source/ destination | Description |
|---|---|---|---|
| **MCIINTR0** | Output | Interrupt controller | Interrupt request 0 |
| **MCIINTR1** | Output | Interrupt controller | Interrupt request 1 |
| **MCIDMASREQ** | Output | DMA controller | DMA single word request |
| **MCIDMABREQ** | Output | DMA controller | DMA burst request |
| **MCIDMALSREQ** | Output | DMA controller | DMA last word request |
| **MCIDMALBREQ** | Output | DMA controller | DMA last burst request |
| **MCIDMACLR** | Input | DMA controller | DMA request clear |
| **MCLK** | Input | Clock source | Multimedia card adapter clock |
| **nMCLK** | Input | Clock source | Inverted MCLK |
| **nMCIRST** | Input | Reset controller | Reset signal for **MCLK** domain |

## A.3 Scan test control signals

The internal scan test control signals are shown in Table A-3.

**Table A-3 Scan test control signals**

| Name | Type | Source/ destination | Description |
|---|---|---|---|
| **SCANENABLE** | Input | Scan controller | Scan enable |
| **SCANINPCLK** | Input | Scan controller | Scan data input for **PCLK** domain |
| **SCANINMCLK** | Input | Scan controller | Scan data input for **MCLK** domain |
| **SCANINnMCLK** | Input | Scan controller | Scan data input for **nMCLK** domain |
| **SCANINMCIFBCLK** | Input | Scan controller | Scan data input for **FBCLK** domain |
| **SCANOUTPCLK** | Output | Scan controller | Scan data output for **PCLK** domain |
| **SCANOUTMCLK** | Output | Scan controller | Scan data output for **MCLK** domain |
| **SCANOUTnMCLK** | Output | Scan controller | Scan data output for **nMCLK** domain |
| **SCANOUTMCIFBCLK** | Output | Scan controller | Scan data output for **FBCLK** domain |

## A.4 Multimedia card interface signals

Refer to Table A-4 for an overview of the off-chip PrimeCell MCI signals.

**Table A-4 PrimeCell MCI signals**

| Name | Type | Source/ destination | Description |
|---|---|---|---|
| **MCICLKOUT** | Output | Pad | Multimedia card clock output |
| **MCICMDIN** | Input | Pad | Multimedia card command input |
| **MCICMDOUT** | Output | Pad | Multimedia card command output |
| **MCICCMDEN** | Output | Pad | Multimedia card command output enable |
| **MCIDATIN[3:0]** | Input | Pad | Multimedia card data input |
| **MCIDATOUT[3:0]** | Output | Pad | Multimedia card data output |
| **MCIDAT0EN** | Output | Pad | Multimedia card data enable (line 0) |
| **MCIDATEN** | Output | Pad | Multimedia card data enable (line [3:1]) |
| **MCIPWR** | Output | Pad | Power supply enable |
| **MCIVDD[3:0]** | Output | Pad | Power supply output voltage |
| **MCIROD** | Output | Pad | Open-drain resistor enable |
| **MCIFBCLK** | Input | Pad | Fed-back clock from multimedia card |
| **MCISDSELECT[3:0]** | Output | Pad | Multimedia card select |