Actel SmartFusion® MSS PDMA Driver User's Guide

Version 2.0

Actel
POWER MATTERS

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200189-0

Release: February 2010

Trademarks

Table of Contents

Introduction

The SmartFusion® Microcontroller Subsystem (MSS) includes an 8 channel Peripheral DMA (PDMA) controller. This software driver provides a set of functions for controlling the MSS PDMA controller as part of a bare metal system where no operating system is available. This driver can be adapted for use as part of an operating system but the implementation of the adaptation layer between this driver and the operating system's driver model is outside the scope of this driver.

Features

The MSS PDMA driver provides the following features:

- Support for configuring the operating modes of each individual PDMA channel
- Support for initiating DMA transfers
- Support for generating continuous DMA transfers (ping pong operations)
- Support for detecting completion of DMA transfers
- Support for enabling, disabling and clearing PDMA interrupts

The MSS PDMA driver is provided as C source code.

Supported Hardware IP

The MSS PDMA bare metal driver can be used with Actel's MSS_PDMA IP version 0.1 or higher included in the SmartFusion MSS.

Files Provided

The files provided as part of the MSS PDMA driver fall into three main categories: documentation, driver source code, and example projects. The driver is distributed via the Actel Firmware Catalog, which provides access to the documentation for the driver, generates the driver's source files into an application project, and generates example projects that illustrate how to use the driver. The Actel Firmware Catalog is available from the Actel web site: www.actel.com/products/software/firmwarecat/default.aspx.

Documentation

The Actel Firmware Catalog provides access to these documents for the driver:

- User's guide (this document)
- A copy of the license agreement for the driver source code
- Release notes

Driver Source Code

The Actel Firmware Catalog generates the driver's source code into the *drivers\mss_pdma* subdirectory of the selected software project directory. The files making up the driver are detailed below.

mss_pdma.h

This header file contains the public application programming interface (API) of the MSS PDMA software driver. This file should be included in any C source file that uses the MSS PDMA software driver.

mss_pdma.c

This C source file contains the implementation of the MSS PDMA software driver.

Example Code

The Actel Firmware Catalog provides access to example projects illustrating the use of the driver. Each example project is self contained and is targeted at a specific processor and software toolchain combination. The example projects are targeted at the FPGA designs in the hardware development tutorials supplied with Actel's development boards. The tutorial designs may be found on the Actel Development Kit web page (www.actel.com/products/hardware).

Driver Deployment

This driver is intended to be deployed from the Actel Firmware Catalog into a software project by generating the driver's source files into the project directory. The driver uses the SmartFusion Cortex Microcontroller Software Interface Standard – Peripheral Access Layer (CMSIS-PAL) to access MSS hardware registers. You must ensure that the SmartFusion CMSIS-PAL is either included in the software tool chain used to build your project or is included in your project. The most up-to-date SmartFusion CMSIS-PAL files can be obtained using the Actel Firmware Catalog.

The following example shows the intended directory structure for a SoftConsole ARM® Cortex™-M3 project targeted at the SmartFusion MSS. This project uses the MSS PDMA and MSS Watchdog drivers. Both of these drivers rely on SmartFusion CMSIS-PAL for accessing the hardware. The contents of the *drivers* directory result from generating the source files for each driver into the project. The contents of the *CMSIS* directory result from generating the source files for the SmartFusion CMSIS-PAL into the project.

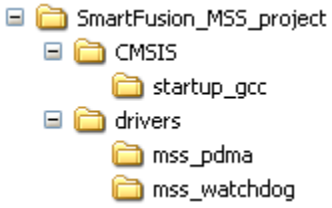

Figure 1 · SmartFusion MSS Project Example

Driver Configuration

The configuration of all features of the MSS PDMA is covered by this driver. There are no dependencies on the hardware flow when configuring the SmartFusion MSS PDMA.

The source and destination of the DMA transfer must be enabled, either in the hardware design or by your application program. The source and destination addresses must be located at valid addresses in the SmartFusion MSS memory map, and must be appropriate to the configuration that you specify for the DMA channel.

The base address and register addresses and interrupt number assignment for the MSS PDMA block are defined as constants in the SmartFusion CMSIS-PAL. You must ensure that the SmartFusion CMSIS-PAL is either included in the software tool chain used to build your project or is included in your project.

Application Programming Interface

This section describes the driver's API. The functions and related data structures described in this section are used by the application programmer to control the MSS PDMA controller from the user's application.

Theory of Operation

The MSS PDMA driver functions are grouped into the following categories:

- Initialization
- Configuration
- DMA transfer control
- Interrupt control

Initialization

The MSS PDMA driver is initialized through a call to the *PDMA_init()* function. The *PDMA_init()* function must be called before any other PDMA driver functions can be called.

Configuration

Each PDMA channel is individually configured through a call to the *PDMA_configure()* function. Configuration includes:

- channel priority
- transfer size
- source and/or destination address increment
- source or destination of the DMA transfer

PDMA channels can be divided into high and low priority channels. High priority channels are given more opportunities to perform transfers than low priority channels when there are continuous high priority channels requests. The ratio of high priority to low priority DMA transfers is configurable through the *PDMA_set_priority_ratio()* function.

PDMA channels can be configured to perform byte (8 bits), half-word (16 bits) or word (32 bits) transfers.

The source and destination address of a PDMA channel's transfers can be independently configured to increment by 0, 1, 2 or 4 bytes. For example, the content of a byte buffer located in RAM can be transferred into a peripheral's transmit register by configuring the source address increment to one byte and no increment of the destination address.

The source or destination of a PDMA channel's transfers can be configured to be one of the MSS peripherals. This allows the PDMA controller to use some hardware flow control signaling with the peripheral to avoid overrunning the peripheral's data buffer when the peripheral is the destination of the DMA transfer, or attempting to read data from the peripheral while it is not ready when the peripheral is the source of the transfer.

A PDMA channel can also be configured to transfer data between two memory mapped locations (memory to memory). No hardware flow control is used by the PDMA controller for data transfer in this configuration.

DMA transfer control

A DMA transfer can be initiated by a call to the *PDMA_start()* function after a PDMA channel has been configured. Once started, further data can be pushed through the PDMA channel by calling the *PDMA_load_next_buffer()* function. The *PDMA_load_next_buffer()* function can be called every time a call to the *PDMA_status()* function indicates that the PDMA channel used for the transfer has a free buffer or it can be called as a result of a PDMA interrupt.

A DMA transfer can be paused and resumed through calls to functions *PDMA_pause()* and *PDMA_resume()*.

Interrupt control

Your application can manage DMA transfers using interrupts through the use of the following functions:

- *PDMA_set_irq_handler()*
- *PDMA_enable_irq()*
- *PDMA_clear_irq()*
- *PDMA_disable_irq()*

The *PDMA_set_irq_handler()* function is used to register PDMA channel interrupt handler functions with the driver. You must create and register an interrupt handler function for each interrupt driven PDMA channel used by the application. Use the *PDMA_enable_irq()* function to enable interrupts for the PDMA channels.

Every time a PDMA channel completes the transfer of a buffer it causes a PDMA interrupt to occur and the PDMA driver will call the interrupt handler registered by the application for that PDMA channel.

Types

pdma_channel_id_t

Prototype

```
typedef enum __pdma_channel_id
{
    PDMA_CHANNEL_0 = 0,
    PDMA_CHANNEL_1,
    PDMA_CHANNEL_2,
    PDMA_CHANNEL_3,
    PDMA_CHANNEL_4,
    PDMA_CHANNEL_5,
    PDMA_CHANNEL_6,
    PDMA_CHANNEL_7
} pdma_channel_id_t;
```

Description

The *pdma_channel_id_t* enumeration is used to identify peripheral DMA channels. It is used as function parameter to specify the PDMA channel used.

pdma_priority_ratio_t

Prototype

```
typedef enum __pdma_priority_ratio_t
{
    PDMA_ROUND_ROBIN = 0,
    PDMA_RATIO_HIGH_LOW_1_TO_1 = 1,
    PDMA_RATIO_HIGH_LOW_3_TO_1 = 3,
    PDMA_RATIO_HIGH_LOW_7_TO_1 = 7,
    PDMA_RATIO_HIGH_LOW_15_TO_1 = 15,
    PDMA_RATIO_HIGH_LOW_31_TO_1 = 31,
    PDMA_RATIO_HIGH_LOW_63_TO_1 = 63,
    PDMA_RATIO_HIGH_LOW_127_TO_1 = 127,
    PDMA_RATIO_HIGH_LOW_255_TO_1 = 255
} pdma_priority_ratio_t;
```

Description

The *pdma_priority_ratio_t* enumeration is used to configure the ratio of high priority to low priority PDMA channels. This ratio specifies how many DMA transfer opportunities will be given to high priority channels before a DMA transfer opportunity is given to a low priority channel when there are continuous requests from high priority channels. This enumeration is used as parameter to function *PDMA_set_priority_ratio()*.

pdma_src_dest_t

Prototype

```
typedef enum __pdma_src_dest
{
    PDMA_FROM_UART_0 = 0,
    PDMA_TO_UART_0,
    PDMA_FROM_UART_1,
    PDMA_TO_UART_1,
    PDMA_FROM_SPI_0,
    PDMA_TO_SPI_0,
    PDMA_FROM_SPI_1,
    PDMA_TO_SPI_1,
    PDMA_FROM_FPGA_1,
    PDMA_TO_FPGA_1,
    PDMA_FROM_FPGA_0,
    PDMA_TO_FPGA_0,
    PDMA_TO_ACE,
    PDMA_FROM_ACE,
    PDMA_MEM_TO_MEM
} pdma_src_dest_t;
```

Description

The *pdma_src_dest_*t enumeration is used to specify the source or destination of transfers on a PDMA channel. It specifies which hardware peripheral will be the source or destination of DMA transfers. This allows the PDMA controller to use hardware flow control signals to avoid overrunning a destination peripheral with data it is not ready to receive, or attempting to transfer data from a peripheral while it has no data ready to transfer.

The *pdma_src_dest_*t enumeration can also be used to specify that a PDMA channel is configured to transfer data between two memory mapped locations (memory to memory). No hardware data flow control is used by the PDMA controller in this configuration.

This enumeration is used as parameter to function *PDMA_configure()*.

pdma_channel_isr_t

Prototype

```
typedef void(* pdma_channel_isr_t)(void);
```

Description

The *pdma_channel_isr_t* type is a pointer to a PDMA channel interrupt handler function. It specifies the function prototype of functions that can be registered as PDMA channel interrupt handlers. It is used as parameter to function *PDMA_set_irq_handler()*.

Constant Values

DMA channel transfer size

These constants are used to build the *channel_cfg* parameter of the *PDMA_configure()* function. They specify the data width of the transfers performed by a PDMA channel.

Constant	Description
PDMA_BYTE_TRANSFER	Byte transfers (8 bits)
PDMA_HALFWORD_TRANSFER	Half-word transfers (16 bits)
PDMA_WORD_TRANSFER	Word transfers (32 bits)

Table 1 • DMA transfer sizes

DMA channel priority

These constants are used to build the *channel_cfg* parameter of the *PDMA_configure()* function. They specify whether a channel is a high or low priority channel.

Constant	Description
PDMA_LOW_PRIORITY	Low priority channel
PDMA_HIGH_PRIORITY	High priority channel

Table 2 • DMA channel priority groups

DMA channel source and/or destination address increment

These constants are used to build the *channel_cfg* parameter of the *PDMA_configure()* function. They specify the PDMA channel's source and destination address increment.

Constant	Description
PDMA_NO_INC	No address increment between transfers
PDMA_INC_DEST_ONE_BYTE	Increment destination address by one byte after each transfer
PDMA_INC_DEST_TWO_BYTES	Increment destination address by two bytes after each transfer
PDMA_INC_DEST_FOUR_BYTES	Increment destination address by four bytes after each transfer
PDMA_INC_SRC_ONE_BYTE	Increment source address by one byte after each transfer
PDMA_INC_SRC_TWO_BYTES	Increment source address by two bytes after each transfer
PDMA_INC_SRC_FOUR_BYTES	Increment source address by four bytes after each transfer

Table 3 • Source and destination address increment

Source address for DMA transfer

These constants are used to specify the *src_addr* parameter to the *PDMA_start()* and *PDMA_load_next_buffer()* functions. They specify the receive register address of peripherals that can be the source of a DMA transfer.

When a PDMA channel is configured for DMA transfers from a peripheral to memory, the constant specifying that peripheral's receive register address must be used as the *src_addr* parameter.

Constant	Description
PDMA_SPI0_RX_REGISTER	SPI0 receive data register
PDMA_SPI1_RX_REGISTER	SPI1 receive data register
PDMA_UART0_RX_REGISTER	UART0 receive buffer register
PDMA_UART1_RX_REGISTER	UART1 receive buffer register
PDMA_ACE_PPE_DATAOUT	ACE post processing engine data output register

Table 4 • Transfer source registers

Destination address for DMA transfer

These constants are used to specify the *dest_addr* parameter to the *PDMA_start()* and *PDMA_load_next_buffer()* functions. They specify the transmit register address of peripherals that can be the destination of a DMA transfer.

When a PDMA channel is configured for DMA transfers from memory to a peripheral, the constant specifying that peripheral's transmit register address must be used as the *dest_addr* parameter.

Constant	Description
PDMA_SPI0_TX_REGISTER	SPI0 transmit data register
PDMA_SPI1_TX_REGISTER	SPI1 transmit data register
PDMA_UART0_TX_REGISTER	UART0 transmit holding register
PDMA_UART1_TX_REGISTER	UART1 transmit holding register
PDMA_ACE_SSE_DATAIN	ACE sample sequencing engine data input register

Table 5 • Transfer destination registers

Write adjust default value

The PDMA_DEFAULT_WRITE_ADJ constant provides a suitable default value for the *PDMA_configure()* function *write_adjust* parameter.

Data Structures

There are no MSS PDMA driver specific data structures.

Functions

PDMA_init

prototype

```
void PDMA_init (void);
```

Description

The *PDMA_init()* function initializes the peripheral DMA hardware and driver internal data. It resets the PDMA and it also clears any pending PDMA interrupts in the Cortex-M3 interrupt controller. When the function exits, it takes the PDMA block out of reset.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

PDMA_configure

prototype

```
void PDMA_configure
(
        pdma_channel_id_t channel_id,
        pdma_src_dest_t src_dest,
        uint32_t channel_cfg,
        uint8_t write_adjust
);
```

Description

The *PDMA_configure()* function configures a PDMA channel. It specifies:

- The peripheral which will be the source or destination of the DMA transfer
- Whether the DMA channel will be a high or low priority channel
- The source and destination address increment that will take place after each transfer

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

src_dest

The *src_dest* parameter specifies the source or destination of the DMA transfers that will be performed. It can be one of the following:

- PDMA_FROM_UART_0
- PDMA_TO_UART_0
- PDMA_FROM_UART_1
- PDMA_TO_UART_1

- PDMA_FROM_SPI_0
- PDMA_TO_SPI_0
- PDMA_FROM_SPI_1
- PDMA_TO_SPI_1
- PDMA_FROM_FPGA_1
- PDMA_TO_FPGA_1
- PDMA_FROM_FPGA_0
- PDMA_TO_FPGA_0
- PDMA_TO_ACE
- PDMA_FROM_ACE
- PDMA_MEM_TO_MEM

channel_cfg

The *channel_cfg* parameter specifies the configuration of the PDMA channel. The configuration includes:

- channel priority
- transfer size
- source and/or destination address increment

The *channel_cfg* parameter value is a logical OR of these components:

One of the following to specify the channel priority:

- PDMA_LOW_PRIORITY
- PDMA_HIGH_PRIORITY

One of the following to specify the transfer size:

- PDMA_BYTE_TRANSFER
- PDMA_HALFWORD_TRANSFER
- PDMA_WORD_TRANSFER

One or two of the following to specify the source and/or destination address increment:

- PDMA_NO_INC
- PDMA_INC_SRC_ONE_BYTE
- PDMA_INC_SRC_TWO_BYTES
- PDMA_INC_SRC_FOUR_BYTES
- PDMA_INC_DEST_ONE_BYTE
- PDMA_INC_DEST_TWO_BYTES
- PDMA_INC_DEST_FOUR_BYTES

write_adjust

The *write_adjust* parameter specifies the number of Cortex-M3 clock cycles the PDMA controller will wait before attempting another transfer cycle. This delay is necessary when peripherals are used as destination of a DMA transfer to ensure the DMA controller interprets the state of the peripheral's ready signal only after data has actually been written to the peripheral. This delay accounts for posted writes (dump and run) for write accesses to peripherals. The effect of posted writes is that if the PDMA performs a write operation to a peripheral, the data is not actually written into the peripheral until sometime after the PDMA controller thinks it is written.

A suitable value for *write_adjust* depends on the target of the DMA transfer. Guidelines for choosing this value are as follows:

- The PDMA_DEFAULT_WRITE_ADJ constant provides a suitable default value for the *write_adjust* parameter when the PDMA channel is configured for transfers with MSS peripherals.

- The PDMA_DEFAULT_WRITE_ADJ constant can also be used for DMA transfers with FPGA fabric implemented peripherals making use of the DMAREADY0 or DMAREADY1fabric interface signal to indicate that the peripheral is ready for another DMA transfer.
- The *write_adjust* parameter can be set to zero to achieve maximum transfer speed for genuine memory to memory transfers.
- The internal latency of FPGA implemented peripherals will decide the *write_adjust* value for fabric peripherals that do not use the DMAREADY0 or DMAREADY1 fabric interface signals. You need to check the fabric peripheral documentation for the value to use.

Return Value

This function does not return a value.

Example

```
PDMA_configure
    (
        PDMA_CHANNEL_0,
        PDMA_TO_SPI_1,
        PDMA_LOW_PRIORITY | PDMA_BYTE_TRANSFER | PDMA_INC_SRC_ONE_BYTE,
        PDMA_DEFAULT_WRITE_ADJ
    );
```

PDMA_set_priority_ratio

prototype

```
void PDMA_set_priority_ratio (pdma_priority_ratio_t priority_ratio);
```

Description

The *PDMA_set_priority_ratio()* function sets the ratio of high priority to low priority DMA access opportunities. This ratio is used by the PDMA controller arbiter to decide which PDMA channel will be given the opportunity to perform a transfer when multiple PDMA channels are requesting to transfer data at the same time. The priority ratio specifies how many DMA transfer opportunities will be given to high priority channels before a DMA transfer opportunity is given to a low priority channel when there are continuous requests from high priority channels.

Parameters

priority_ratio

The *priority_ratio* parameter specifies the ratio of DMA transfer opportunities given to high priority channels versus low priority channels. Allowed values for this parameter are as follows:

- PDMA_ROUND_ROBIN
- PDMA_RATIO_HIGH_LOW_1_TO_1
- PDMA_RATIO_HIGH_LOW_3_TO_1
- PDMA_RATIO_HIGH_LOW_7_TO_1
- PDMA_RATIO_HIGH_LOW_15_TO_1
- PDMA_RATIO_HIGH_LOW_31_TO_1
- PDMA_RATIO_HIGH_LOW_63_TO_1
- PDMA_RATIO_HIGH_LOW_127_TO_1
- PDMA_RATIO_HIGH_LOW_255_TO_1

Return Value

This function does not return a value.

Example

```
PDMA_set_priority_ratio( PDMA_ROUND_ROBIN );
```

PDMA_start

prototype

```
void PDMA_start
(
        pdma_channel_id_t channel_id,
        uint32_t src_addr,
        uint32_t dest_addr,
        uint16_t transfer_count
);
```

Description

The *PDMA_start()* function initiates a DMA transfer. It specifies the source and destination address of the transfer as well as the number of transfers that must take place. The source and destination addresses can be the address of peripheral registers.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

src_addr

The *src_addr* parameter specifies the address location of the data to be transferred. You must ensure that this source address is consistent with the DMA source configured for the selected channel using the *PDMA_configure()* function.

For DMA transfers from MSS peripheral to memory, the following *src_addr* parameter values are allowed:

- PDMA_SPI0_RX_REGISTER
- PDMA_SPI1_RX_REGISTER
- PDMA_UART0_RX_REGISTER
- PDMA_UART1_RX_REGISTER
- PDMA_ACE_PPE_DATAOUT

For DMA transfers from FPGA fabric peripheral to memory, the following *src_addr* parameter values are allowed:

- An address in the FPGA fabric address space (0x40050000-0x400FFFFF)

For DMA transfers from memory to MSS peripheral, or from memory to FPGA fabric peripheral, or from memory to memory, the following *src_addr* parameter values are allowed:

- Any memory mapped address.

dest_addr

The *dest_addr* parameter specifies the destination address of the PDMA transfer. You must ensure that this matches with the DMA destination configured for the selected channel.

For DMA transfers from memory to MSS peripheral, the following *dest_addr* parameter values are allowed:

- PDMA_SPI0_TX_REGISTER
- PDMA_SPI1_TX_REGISTER
- PDMA_UART0_TX_REGISTER
- PDMA_UART1_TX_REGISTER
- PDMA_ACE_SSE_DATAIN

For DMA transfers from memory to FPGA fabric peripheral, the following *dest_addr* parameter values are allowed:

- An address in the FPGA fabric address space (0x40050000-0x400FFFFF)

For DMA transfers from MSS peripheral to memory, or from FPGA fabric peripheral to memory, or from memory to memory, the following *dest_addr* parameter values are allowed:

- Any memory mapped address.

transfer_count

The *transfer_count* parameter specifies the number of transfers to be performed. It is the number of bytes to transfer if the PDMA channel is configured for byte transfer, the number of half-words to transfer if the PDMA channel is configured for half-word transfer, or the number of words to transfer if the PDMA channel is configured for word transfer.

Return Value

This function does not return a value.

Example

```
PDMA_start
(
    PDMA_CHANNEL_3,
    SPI1_RX_REGISTER,
    (uint32_t)slave_rx_buffer,
    sizeof(slave_rx_buffer)
);
```

PDMA_load_next_buffer

prototype

```
void PDMA_load_next_buffer
(
        pdma_channel_id_t channel_id,
        uint32_t src_addr,
        uint32_t dest_addr,
        uint16_t transfer_count
);
```

Description

The *PDMA_load_next_buffer()* function sets up the next buffer to be transferred. This function is called after a transfer has been initiated using the *PDMA_start()* function. Its purpose is to keep feeding a PDMA channel with data buffers.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

src_addr

The *src_addr* parameter specifies the address location of the data to be transferred. You must ensure that this source address is consistent with the DMA source configured for the selected channel using the *PDMA_configure()* function.

For DMA transfers from MSS peripheral to memory, the following *src_addr* parameter values are allowed:

- PDMA_SPI0_RX_REGISTER
- PDMA_SPI1_RX_REGISTER
- PDMA_UART0_RX_REGISTER
- PDMA_UART1_RX_REGISTER
- PDMA_ACE_PPE_DATAOUT

For DMA transfers from FPGA fabric peripheral to memory, the following *src_addr* parameter values are allowed:

- An address in the FPGA fabric address space (0x40050000-0x400FFFFF)

For DMA transfers from memory to MSS peripheral, or from memory to FPGA fabric peripheral, or from memory to memory, the following *src_addr* parameter values are allowed:

- Any memory mapped address.

dest_addr

The *dest_addr* parameter specifies the destination address of the PDMA transfer. You must ensure that this matches with the DMA destination configured for the selected channel.

For DMA transfers from memory to MSS peripheral, the following *dest_addr* parameter values are allowed:

- PDMA_SPI0_TX_REGISTER
- PDMA_SPI1_TX_REGISTER
- PDMA_UART0_TX_REGISTER
- PDMA_UART1_TX_REGISTER
- PDMA_ACE_SSE_DATAIN

For DMA transfers from memory to FPGA fabric peripheral, the following *dest_addr* parameter values are allowed:

- An address in the FPGA fabric address space (0x40050000-0x400FFFFF)

For DMA transfers from MSS peripheral to memory, or from FPGA fabric peripheral to memory, or from memory to memory, the following *dest_addr* parameter values are allowed:

- Any memory mapped address.

transfer_count

The *transfer_count* parameter specifies the number of transfers to be performed. It is the number of bytes to transfer if the PDMA channel is configured for byte transfer, the number of half-words to transfer if the PDMA channel is configured for half-word transfer or the number of words to transfer if the PDMA channel is configured for word transfer.

Return Value

This function does not return a value.

Example

```
void write_cmd_data
(
    mss_spi_instance_t * this_spi,
    const uint8_t * cmd_buffer,
    uint16_t cmd_byte_size,
    uint8_t * data_buffer,
    uint16_t data_byte_size
)
{
    uint32_t transfer_size;
    transfer_size = cmd_byte_size + data_byte_size;

    MSS_SPI_disable( this_spi );
    MSS_SPI_set_transfer_byte_count( this_spi, transfer_size );

    PDMA_start
        (
            PDMA_CHANNEL_0,
            (uint32_t)cmd_buffer,
            PDMA_SPI1_TX_REGISTER,
            cmd_byte_size
        );

    PDMA_load_next_buffer
        (
            PDMA_CHANNEL_0,
            (uint32_t)data_buffer,
            PDMA_SPI1_TX_REGISTER,
            data_byte_size
        );

    MSS_SPI_enable( this_spi );
        while ( !MSS_SPI_tx_done(this_spi) ) { ; }
}
```

PDMA_pause

prototype

```
void PDMA_pause
(
        pdma_channel_id_t channel_id
);
```

Description

The *PDMA_pause()* function temporarily pauses a DMA transfer taking place on the specified PDMA channel. The transfer can later be resumed by using the *PDMA_resume()* function.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

Return Value

This function does not return a value.

PDMA_resume

prototype

```
void PDMA_resume
(
        pdma_channel_id_t channel_id
);
```

Description

The *PDMA_resume()* function resumes a transfer previously paused using the *PDMA_pause()* function.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

Return Value

This function does not return a value.

PDMA_set_irq_handler

prototype

```
void PDMA_set_irq_handler
(
       pdma_channel_id_t channel_id,
       pdma_channel_isr_t handler
);
```

Description

The *PDMA_set_irq_handler()* function registers a handler function for interrupts generated on the completion of a transfer on a specific PDMA channel. This function also enables the PDMA interrupt both in the PDMA controller and in the Cortex-M3 interrupt controller.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

handler

The *handler* parameter is a pointer to the function that will be called when a transfer completes on the PDMA channel identified by *channel_id* and the interrupt is enabled for that channel.

Return Value

This function does not return a value.

Example

```
void slave_dma_irq_handler( void )
{
    if ( g_spi1_rx_buffer[2] == 0x99 )
    {
        PDMA_load_next_buffer
        (
            PDMA_CHANNEL_0,
            (uint32_t)g_spi1_tx_buffer_b,
            PDMA_SPI1_TX_REGISTER,
            sizeof(g_spi1_tx_buffer_b)
        );
    }
    PDMA_disable_irq( PDMA_CHANNEL_3 );
}

void setup_dma( void )
{
    PDMA_init();
    PDMA_configure
    (
         PDMA_CHANNEL_0,
         PDMA_TO_SPI_1,
```

```
        PDMA_LOW_PRIORITY | PDMA_BYTE_TRANSFER | PDMA_INC_SRC_ONE_BYTE
    );
    PDMA_configure
    (
        PDMA_CHANNEL_3,
        PDMA_FROM_SPI_1,
        PDMA_HIGH_PRIORITY | PDMA_BYTE_TRANSFER | PDMA_INC_DEST_ONE_BYTE
    );
    PDMA_set_irq_handler( PDMA_CHANNEL_3, slave_dma_irq_handler );
    PDMA_start( PDMA_CHANNEL_3, PDMA_SPI1_RX_REGISTER, (uint32_t)g_spi1_rx_buffer, 3 );
}
```

PDMA_enable_irq

prototype

```
void PDMA_enable_irq
(
        pdma_channel_id_t channel_id
);
```

Description

The *PDMA_enable_irq()* enables the PDMA hardware to generate an interrupt when a DMA transfer completes on the specified PDMA channel. This function also enables the PDMA interrupt in the Cortex-M3 interrupt controller.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

Return Value

This function does not return a value.

PDMA_disable_irq

prototype

```
void PDMA_disable_irq
(
        pdma_channel_id_t channel_id
);
```

Description

The *PDMA_disable_irq()* function disables interrupts for a specific PDMA channel.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

Return Value

This function does not return a value.

PDMA_clear_irq

prototype

```
void PDMA_clear_irq
(
        pdma_channel_id_t channel_id
);
```

Description

The *PDMA_clear_irq()* function clears interrupts for a specific PDMA channel. This function also clears the PDMA interrupt in the Cortex-M3 NVIC.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

Return Value

This function does not return a value.

PDMA_status

prototype

```
uint32_t PDMA_status
(
        pdma_channel_id_t channel_id
);
```

Description

The *PDMA_status()* function returns the status of a PDMA channel. The returned value indicates whether transfers have been completed using buffer A or buffer B of the PDMA hardware block.

Parameters

channel_id

The *channel_id* parameter identifies the PDMA channel used by the function.

Return Value

Bit 0 of the return value indicates whether buffer A has been transferred. It is set to 1 if the transfer has completed.

Bit 1 of the return value indicates whether buffer B has been transferred. It is set to 1 if the transfer has completed.

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650. 318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (http://www.actel.com/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at http://www.actel.com/.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/company/contact/default.aspx.

Actel is the leader in low-power FPGAs and mixed-signal FPGAs and offers the most comprehensive portfolio of system and power management solutions. Power Matters. Learn more at http://www.actel.com .

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 • USA*
Phone 650.318.4200 • Fax 650.318.4600 • Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd*. • River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom*
Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

Actel Japan *• EXOS Ebisu Building 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • http://jp.actel.com

Actel Hong Kong *• Room 2107, China Resources Building • 26 Harbour Road • Wanchai • Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200189-0/2.10