
Accessing Serial Flash Memory Using SPI Interface

Table of Contents

Introduction	1
Design Example Overview	1
Design Description	2
Interface Description	3
Software Implementation	3
Running the Design	4
Conclusion	5
Appendix A – Design Files	5
Appendix B – Driver Application Programming Interfaces (APIs)	6
List of Changes	9

Introduction

The SmartFusion[®] customizable system-on-chip (cSoC) device contains a hard embedded microcontroller subsystem (MSS), programmable analog circuitry, and FPGA fabric consisting of logic tiles, static random access memory (SRAM), and phase-locked loops (PLLs). The MSS consists of a 100 MHz ARM[®] Cortex™-M3 processor, advanced high-performance bus (AHB) matrix, system registers, Ethernet MAC, DMA engine, real-time counter (RTC), embedded nonvolatile memory (eNVM), embedded SRAM (eSRAM), fabric interface controller (FIC), the Philips Inter-Integrated Circuit (I²C), serial peripheral interface (SPI), and external memory controller (EMC).

The MSS has two identical SPI peripherals. These peripherals provide serial interface compliance with the Motorola SPI, Texas Instruments synchronous serial, and National Semiconductor MICROWIRE™ formats. The SPI peripherals in SmartFusion cSoC devices can operate as either a Master or a Slave. When operating in Master mode, the SPI peripherals generate a serial clock and data to the slave device that needs to be accessed. The SPI peripherals can generate the serial clock from 390 KHz to 50 MHz by dividing the MSS clock, which can be controlled by software. The peripheral DMA (PDMA) in the MSS can be used for continuous DMA streaming for large SPI transfers, and thus helps to free up the ARM Cortex-M3 processor. Refer to the *SmartFusion Microcontroller Subsystem User's Guide* for more details on SPI peripherals.

This application note explains how to use the SmartFusion SPI interface to access serial flash memory. A basic understanding of the SmartFusion design flow is assumed. Refer to the *Using UART with a SmartFusion cSoC - Libero SoC and SoftConsole Flow Tutorial* to understand the SmartFusion design flow.

Design Example Overview

This design example demonstrates using SPI flash on the SmartFusion Development Kit Board and the SmartFusion Evaluation Kit Board. These kits have an Atmel SPI flash memory, AT25DF641-MWH-T, which is connected to SPI_0 on the SmartFusion Evaluation Kit Board, and to SPI_1 on the SmartFusion Development Kit Board. For this reason, this design example is created using two SPI interfaces; so that the same design can be used with both the kits by changing the macro definition in the SPI flash header file (spi_flash.h). Atmel SPI flash memory provides flexible, optimized erase architecture, and supports uniform block erase (4 Kbyte, 32 Kbyte, and 64 Kbyte), and full chip erase. Refer to the Atmel AT25DF641-MWH-T datasheet, 64-Megabit 2.7-volt Minimum SPI Serial Flash Memory, available at www.atmel.com.

Figure 1 shows the top-level interface signals used in this design example. The UART in the MSS acts as a user interface for writing/reading string data into SPI flash through HyperTerminal.

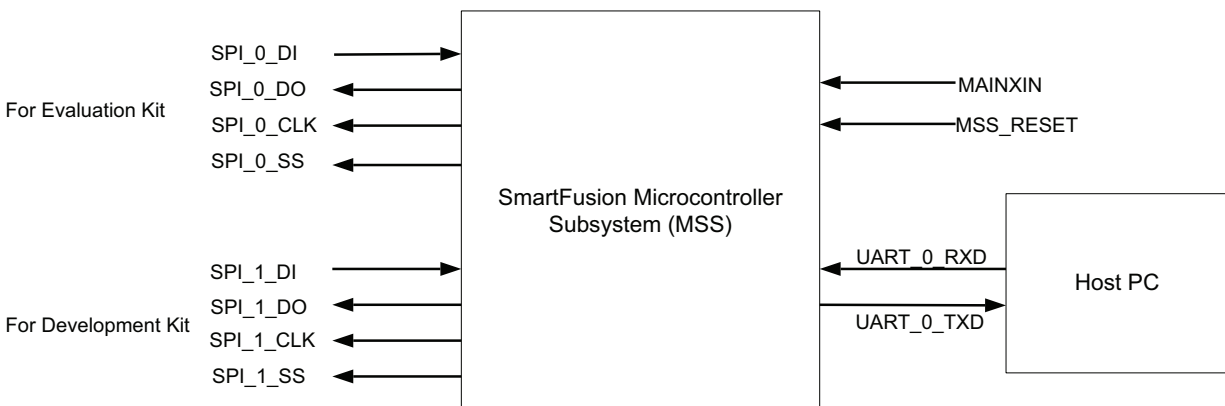


Figure 1 • Top-Level Interface Signals

Design Description

The MSS is configured to use both SPI interfaces (SPI_0 and SPI_1) and one UART interface (UART_0). SPI_0 is clocked by PCLK0 on APB bus 0 and SPI_1 is clocked by PCLK1 on APB bus 1. PCLK0 and PCLK1 are derived from clock conditioning circuitry (CCC) in the MSS that generates an 80 MHz clock.

The SPI peripherals in the MSS are configured with SPI protocol mode 3 and APB bus clock (PCLK) divider as 128 to generate 625 KHz serial clocks (SPI_0_CLK and SPI_1_CLK). Refer to the [SmartFusion MSS SPI Drivers v2.0 User's Guide](#) for more details on the SPI driver's API.

This design example uses UART as a user interface for writing string data into SPI flash and reading string data from SPI flash on HyperTerminal. You can enter a maximum of 4 Kbytes of string data through HyperTerminal. The application first erases the first 4 Kbyte memory location starting from address 0x0000000, using the block erase command, and then writes the string data into the SPI flash. It then reads string data from memory after the write operation that prints on HyperTerminal.

Figure 2 shows the string data flow from HyperTerminal to SPI flash memory and vice versa.

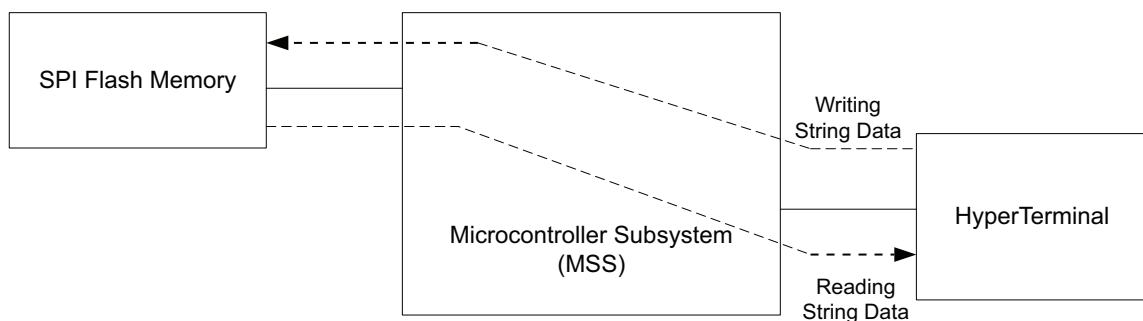


Figure 2 • SPI Flash Memory Read/Write Flow

The Verilog Libero[®] System-on-Chip (SoC) projects are provided in the design files associated with this design example.

Interface Description

Table 1 shows the top-level interface signal descriptions.

Table 1 • Interface Description

Signal	Direction	Description
MSS_RESET_N	Input	Active low reset signal for the MSS
MAINXIN	Input	Main crystal oscillator circuit Input to the crystal oscillator circuit Pin for connecting an external crystal, ceramic resonator, or RC network
UART_0_TXD	Output	UART transmit data
UART_0_RXD	Input	UART receive data
SPI_0_DI	Input	Serial data input
SPI_0_DO	Output	Serial data output
SPI_0_CLK	Output	Serial clock
SPI_0_SS	Output	Serial select
SPI_1_DI	Input	Serial data input
SPI_1_DO	Output	Serial data output
SPI_1_CLK	Output	Serial clock
SPI_1_SS	Output	Serial select

Software Implementation

The following SPI flash driver APIs are used in the example design to access Atmel SPI flash memory, AT25DF641-MWH-T.

spi_flash_int ()

This function initializes and configures the SPI peripheral and PDMA for data transfer. It configures the SPI controller with Protocol mode, serial clock speed, and frame size for SPI flash memory. The design example is configured with SPI Protocol mode as SPI mode 3, APB bus clock (PCLK) divider as 128, and frame size as 8 bit.

spi_flash_control_hw ()

This function performs various operations on the serial flash based on the command passed as first parameter. The operation of the each command is explained in "[Appendix B – Driver Application Programming Interfaces \(APIs\)](#)" on page 6.

The design example uses the following API call to unprotect and erase flash sectors:

```
spi_flash_control_hw (SPI_FLASH_SECTOR_UNPROTECT, 0, NULL);
spi_flash_control_hw (SPI_FLASH_4KBLOCK_ERASE, 0, NULL);
```

spi_flash_write ()

This function writes the content of the buffer passed as a parameter to serial flash. The data is written from the memory location specified by the first parameter. This address ranges from 0 to SPI flash size and is not the processor's absolute range.

spi_flash_read ()

This function reads the content from the serial flash. The data is read from the memory location specified by the first parameter. This address ranges from 0 to SPI flash size and is not the processor's absolute range.

Refer to "[Appendix B – Driver Application Programming Interfaces \(APIs\)](#)" on page 6 for more details on SPI initialization and configuration, and SPI flash read and write operations.

Macro Settings

This design example works with the SmartFusion Evaluation Kit Board and the SmartFusion Development Kit Board. The following macros are to be used in the SPI flash API file (spi_flash.h) to enable the appropriate board:

- `#define SPI_FLASH_ON_SF_DEV_KIT 1`: This macro enables the SPI flash driver software for the SmartFusion Development Kit Board.
- `#define SPI_FLASH_ON_SF_EVAL_KIT 1`: This macro enables the SPI flash driver software for the SmartFusion Evaluation Kit Board.

Comment out the macros based on the board used to run this example.

This design example supports the DMA transfer for SPI flash. The following macros are to be used to enable/disable the DMA:

- `#define USE_DMA_FOR_SPI_FLASH 1`: This macro enables the PDMA transfer for the SPI driver.
- `#define SPI_FLASH_DMA_CHANNEL 0`: This macro represents the DMA channel number.

To disable PDMA transfer, make sure the above macros are commented out; rebuild the project and launch the debugger.

Running the Design

Board Settings

The design example works on the SmartFusion Development Kit Board and the SmartFusion Evaluation Kit Board with default board settings. Refer to the following user's guides for default board settings:

- [SmartFusion Development Kit User's Guide](#)
- [SmartFusion Evaluation Kit User's Guide](#)

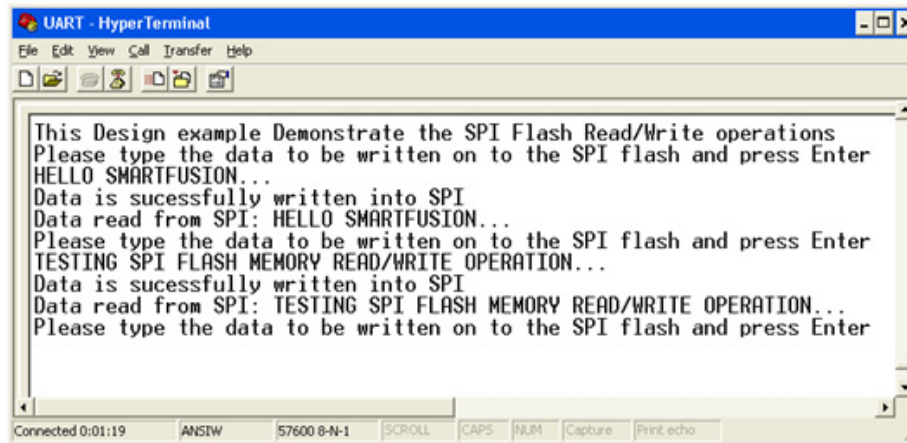
Program the Design and Running the Application

Program the SmartFusion Evaluation Kit Board or the SmartFusion Development Kit Board with the generated/provided *.stp file (refer to "[Appendix A – Design Files](#)" on page 5) using FlashPro, and then power cycle the board. Invoke the SoftConsole IDE by clicking on the **Write Application code** under **Develop Firmware** in Libero SoC tool (refer to "[Appendix A – Design Files](#)" on page 5) and launch the debugger. Start a HyperTerminal with the baud rate set to 57600, 8 data bits, 1 stop bit, no parity, and no flow control.

If your PC does not have HyperTerminal program, use any free serial terminal emulation program like PuTTY or Tera Term. Refer to the [Configuring Serial Terminal Emulation Programs](#) tutorial for configuring the HyperTerminal, Tera Term, and PuTTY.

When you run the debugger in SoftConsole, the HyperTerminal window prompts you to enter the string data to be written into the SPI flash memory. Once you enter the string data, the application reads data from memory and prints on the HyperTerminal display.

Figure 3 shows a screen shot of HyperTerminal displaying SP flash memory read/write operation.



```
UART - HyperTerminal
File Edit View Call Transfer Help
This Design example Demonstrate the SPI Flash Read/Write operations
Please type the data to be written on to the SPI flash and press Enter
HELLO SMARTFUSION...
Data is successfully written into SPI
Data read from SPI: HELLO SMARTFUSION...
Please type the data to be written on to the SPI flash and press Enter
TESTING SPI FLASH MEMORY READ/WRITE OPERATION...
Data is successfully written into SPI
Data read from SPI: TESTING SPI FLASH MEMORY READ/WRITE OPERATION...
Please type the data to be written on to the SPI flash and press Enter
Connected 0:01:19 ANSIW 57600 8-N-1 SCROLL CAPS NUM Capture Print echo
```

Figure 3 • The HyperTerminal Display

Release mode

The release mode programming file (STAPL) is also provided. Refer to the Readme.txt file included in the programming zip file for more information.

Refer to the [Building Executable Image in Release Mode and Loading into eNVM tutorial](#) for more information on building an application in release mode.

Conclusion

This design example demonstrated the usage of SPI peripherals on the SmartFusion cSoC FPGAs to access serial flash memory with sample software. It also explained the usage of SPI flash driver APIs to implement memory write/read operations.

Appendix A – Design Files

You can download the design files from the Microsemi SoC Products Group website:

www.microsemi.com/soc/download/rsc/?f=A2F_AC343_DF.

The design file consists of Libero Verilog, SoftConsole software project, programming files (*.stp) for A2F500-DEV-KIT, and A2F-EVAL-KIT. Refer to the Readme.txt file included in the design file for the directory structure and description.

You can download the programming files (*.stp) in release mode from the Microsemi SoC Products Group website: www.microsemi.com/soc/download/rsc/?f=A2F_AC343_PF.

The programming zip file consists of STAPL programming file (*.stp) for A2F500-DEV-KIT and A2F-EVAL-KIT, and a Readme.txt file.

Appendix B – Driver Application Programming Interfaces (APIs)

This section describes the software driver APIs used in this design to carry out transactions with SPI flash. These drivers are included in the design files with this design example.

Function Description of Data Structures

Enum: *spi_flash_status_t*

This enum represents the status of different APIs such as `spi_flash_status_t`, `spi_flash_read`, and `spi_flash_write`.

```
typedef enum {
    SPI_FLASH_SUCCESS = 0,
    SPI_FLASH_PROTECTION_ERROR,
    SPI_FLASH_WRITE_ERROR,
    SPI_FLASH_INVALID_ARGUMENTS,
    SPI_FLASH_INVALID_ADDRESS,
    SPI_FLASH_UNSUCCESS
};
```

Enum: *spi_flash_control_hw_t*

This enum represents the status of the API `spi_flash_control_hw`.

```
typedef enum {
    SPI_FLASH_SECTOR_UNPROTECT = 0,
    SPI_FLASH_SECTOR_PROTECT,
    SPI_FLASH_GLOBAL_UNPROTECT,
    SPI_FLASH_GLOBAL_PROTECT,
    SPI_FLASH_GET_STATUS,
    SPI_FLASH_4KBLOCK_ERASE,
    SPI_FLASH_32KBLOCK_ERASE,
    SPI_FLASH_64KBLOCK_ERASE,
    SPI_FLASH_CHIP_ERASE,
    SPI_FLASH_READ_DEVICE_ID,
    SPI_FLASH_RESET
};
```

Structure: *device_Info*

This structure represents the device ID and manufacturing ID of the serial flash device.

```
device_Info{
    uint8_t manufacturer_id;
    uint8_t device_id;
};
```

Function Description of the API

spi_flash_status_t spi_flash_init(void);

This function initializes the SPI peripheral and PDMA for data transfer, SPI 0 for the SmartFusion Evaluation Kit Board, and SPI 1 for the SmartFusion Development Kit Board.

spi_flash_status_t spi_flash_control_hw(spi_flash_control_hw_t operation, uint32_t peram1, void * ptrPeram);

This function performs various operations on the serial flash based on the command passed as the first parameter. The operations supported are as per the enum `spi_flash_control_hw_t` defined. The functionality is as follows:

- `SPI_FLASH_SECTOR_UNPROTECT`: Atmel SPI flash memory, AT25DF641-MWH-T, supports 128 sectors of 64 Kbytes each and there is a corresponding bit set for unprotect of each sector. You must call this operation to unprotect the block and perform a write or erase operation. The second parameter for this function, 'peram1', is the block address to unprotect.
- `SPI_FLASH_SECTOR_PROTECT`: Atmel SPI flash memory, AT25DF641-MWH-T, supports 128 sectors of 64 Kbytes each and there is a corresponding bits set for protection of each sector. You must call this operation to protect the data from being modified (write/erase). The second parameter for this function, 'peram1', is the block address to protect.
- `SPI_FLASH_GLOBAL_UNPROTECT`: This command is used to switch off protect mode for the entire flash for modify operations.
- `SPI_FLASH_GLOBAL_PROTECT`: This command is used to protect/lock the entire flash from modify operations.
- `SPI_FLASH_GET_STATUS`: This function is used to get the SPI flash status register content for more details of the status bits. Refer to page 36 of the Atmel AT25DF641-MWH-T datasheet, 64-Megabit 2.7-volt Minimum SPI Serial Flash Memory, available at www.atmel.com.
- `SPI_FLASH_4KBLOCK_ERASE`: This command is used to erase the block starting at the 4KB boundary. The starting address of the 4K block is passed in the second parameter of this API, peram1.
- `SPI_FLASH_32KBLOCK_ERASE`: This command is used to erase the block starting at the 32KB boundary. The starting address of the 32K block is passed in the second parameter of this API, peram1.
- `SPI_FLASH_64KBLOCK_ERASE`: This command is used to erase the block starting at the 64KB boundary. The starting address of the 64K block is passed in the second parameter of this API, peram1.
- `SPI_FLASH_CHIP_ERASE`: This command is used to erase the entire flash chip.
- `SPI_FLASH_READ_DEVICE_ID`: This command is used to read the device properties. The values are filled in the third parameter of this API, ptrPeram.
- `SPI_FLASH_RESET`: The Reset command allows program or erase operations in progress to be ended abruptly and returns the device to an idle state. The return value indicates whether or not the write was successful. The possible values are as follows:
 - `SPI_FLASH_SUCCESS`: Indicates that the SPI flash operation is correct and complete.
 - `SPI_FLASH_PROTECTION_ERROR`: Indicates that the sector is protected and is not allowing the operation. Unprotect the sector and perform the operation.
 - `SPI_FLASH_INVALID_ARGUMENTS`: Indicates that invalid arguments have been passed to the function.
 - `SPI_FLASH_INVALID_ADDRESS`: Indicates that the function has received an invalid address.
 - `SPI_FLASH_UNSUCCESS`: Indicates that the SPI flash operation is incomplete.

spi_flash_status_t spi_flash_read (uint32_t address, uint8_t * rx_buffer, size_t size_in_bytes);

This function reads the content from the serial flash. The data is read from the memory location specified by the first parameter. This address ranges from 0 to SPI flash size and not the processor's absolute range.

- @param start_addr: This is the address at which data will be read. This address ranges from 0 to SPI flash size. This address range is not the processor's absolute range.
- @param p_data: This is a pointer to the buffer for holding the read data.
- @param nb_bytes: This is the number of bytes to be read from SPI flash.

- @return: The return value indicates whether or not the write was successful. The possible values are as follows:
 - SPI_FLASH_SUCCESS: Indicates the SPI flash operation is correct and complete.
 - SPI_FLASH_PROTECTION_ERROR: Indicates that the sector is protected and not allowing the operation. Unprotect the sector and do the operation.
 - SPI_FLASH_INVALID_ARGUMENTS: Indicates that invalid arguments have been passed to the function.
 - SPI_FLASH_INVALID_ADDRESS: Indicates that the function has received an invalid address.
 - SPI_FLASH_UNSUCCESS: Indicates that the SPI flash operation is incomplete.

spi_flash_status_t spi_flash_write(uint32_t address, uint8_t * write_buffer, size_t size_in_bytes);

This function writes the content of the buffer passed as a parameter to serial flash through SPI. The data is written from the memory location specified by the first parameter. This address ranges from 0 to SPI flash size and not the processor's absolute range.

- @param start_addr: This is the address at which data will be written. This address ranges from 0 to SPI flash size. This address range is not the processor's absolute range.
- @param p_data: This is a pointer to the buffer holding the data to be written into serial flash.
- @param nb_bytes: This is the number of bytes to be written into serial flash.
- @return: The return value indicates whether or not the write was successful. The possible values are as follows:
 - SPI_FLASH_SUCCESS: Indicates that the SPI flash operation is correct and complete.
 - SPI_FLASH_PROTECTION_ERROR: Indicates that the sector is protected and is not allowing the operation. Unprotect the sector and do the operation.
 - SPI_FLASH_WRITE_ERROR: Indicates the SPI flash write operation failed.
 - SPI_FLASH_INVALID_ARGUMENTS: Indicates that the invalid arguments have been passed to the function.
 - SPI_FLASH_INVALID_ADDRESS: Indicates that the function has received an invalid address. Address range should be between 0 to 8 MB.
 - SPI_FLASH_UNSUCCESS: Indicates that the SPI flash operation is incomplete.

List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision*	Changes	Page
Revision 4 (January 2013)	Added "Board Settings" section and modified "Running the Design" section (SAR 43469).	4
Revision 3 (February 2012)	Removed ".zip" extension in the links (SAR 36763).	5
Revision 2 (January 2012)	Modified the text below Figure 2 listed under "Design Description" section (SAR 35785).	2
	Modified information related to Libero SoC projects in the "Running the Design" section (SAR 35785).	4
	Added the "Release mode" section (SAR 35785).	5
	Modified the "Appendix A – Design Files" section (SAR 35785).	5
Revision 1 (August 2010)	Modified the section "Running the Design" (SAR 27470).	4
	Removed Figure 3 • HyperTerminal Settings on page 5 (SAR 27470).	
	Modified the section "Appendix A – Design Files" (SAR 27470).	5
	Removed Table 2 • Design Files Description (SAR 27470).	

*Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.*



Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at www.microsemi.com.

© 2013 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.