



DataFlash® Flash File System Hardware Porting Reference

Version 1.0



Copyright © 1999 – 2002 Atmel Corporation All Rights Reserved
2125 Orchard Parkway
San Jose, CA 95131
Phone (408) 441-0311

Table of Contents

- CHAPTER 1: INTRODUCTION..... 1
- CHAPTER 2: DRIVER CONFIGURATION..... 2
 - Starting..... 2
 - I/O..... 2
 - Spi Config..... 3
- CHAPTER 3: FLASH DRIVER FUNCTION CALLS..... 4
 - ERASE_PAGE 5
 - FLASH_REFRESH 6
 - MOVE_FLASH_To_BUFFER 7
 - PROGRAM_BUFFER..... 8
 - READ_BUFFER..... 9
 - READ_FLASH..... 10
 - WRITE_BUFFER 11
 - WRITE_FLASH_DATA..... 12
- APPENDIX A: MEMORY STRUCTURES AND ENUMERATES 13
 - FLASH SIZE ENUMERATE 13
 - FLASH BUFFER SIZE 13
 - FLASH STRUCTURE INFO 14
 - FLASH COMMANDS..... 14

Chapter 1: Introduction

The Flash Driver has been designed for ease of porting among various hardware platforms. There are three functions that must be created/modified for the target platform. The functions are `Fs_Inw`, `Fs_Outw` and `Config_Spi` they are used by the flash driver. There are also `#defines` for mapping the SPI hardware as well as `#defines` for address mapping for the SPI hardware, the portline I/O mapping and the actual I/O port lines within the port for the Slave Select (SS) and SPI clock. This document describes the porting effort.

Chapter 2: Driver Configuration

Starting

To start the porting effort, please become familiar with the files `fs_hw.c` and `fs_hw.h`. These files have been provided as an example and template for conversions to other hardware platforms.

I/O

The flash driver assumes there are two functions/macros to perform single word reads and writes within the system. Their prototypes are

```
uint16 file_inw(uint32 address)
void file_outw(uint32 address, uint16 data)
```

These functions are used to isolate the driver from the various platform architectures. The example below is for the Motorola 56800E family of DSP's. This example may be found at the bottom of `fs_hw.c`.

```
void file_outw( uint32 address, uint16 data )
{
    asm
    {
        move.l    a10,r3
        nop
        nop
        move.w    y0,x:(r3)
    }
}
```

For platforms that can directly perform I/O using 32 bit pointers a simple macro could be created. Instead of a function call. This would improve performance throughput.

```
#define file_outw(a,b) *((WORD_PTR)(a)) = b
```

Spi Config

The SPI configuration information probably will not correctly align with every processor. This information is provided only as a reference and an aid to quickly port the flash driver to a new processor. The driver expects to call Config_Spi and all of the port configuration and SPI configuration will be performed correctly.

With the creation of file_inw and file_outw the function Config_Spi can be completed. To configure and perform the I/O to the SPI hardware three sets of #defines need to be configured for the address mapping for the I/O and SPI. On many processors the SPI bus is multiplexed with port line I/O. The SS and CLK defines are used to correctly setup and use the port lines for the slave select and SPI clock. The final set of #defines are the actual addresses for the SPI status, control, receive and transmit registers.

```
#define      SS_DATA_REG          //SLAVE SELECT DATA REGISTER
#define      SS_DIR_REG          //SLAVE SELECT DATA DIRECTION REGISTER
#define      SS_PER_REG          //SLAVE SELECT PERIPHERAL CONTROL REGISTER
#define      CLK_DATA_REG        //SPI CLOCK DATA REGISER
#define      CLK_DIR_REG        //SPI CLOCK DATA DIRECTION REGISTER
#define      CLK_PER_REG        //SPI CLOCK PERIPHERAL CONTROL REGISTER
#define      SPI_STATUS_REG      //SPI STATUS REGISTER
#define      SPI_CONTROL_REG     //SPI CONTROL REGISTER
#define      SPI_RX_REG          //SPI RECEIVE REGISTER
#define      SPI_TX_REG          //SPI TRANSMIT REGISTER
```

The SPI configuration defines are used to configure the actual SPI hardware. This example assumes there are configuration parameters in both the spi control register and the spi status register. SPI_CONFIG1 is used for the status and SPI_CONFIG2 is used to configure the control register. The define SPI_ENABLE is assumed to be part of the status register. At the bottom of Config_Spi must be the function Query_Flash_Status. This function will read the Flash part and determine the type of flash connected.

The next two defines are SPI_SLAVE_SELECT and SPI_CLOCK. These two defines specify which port is being used in for the address in SS_DATA_REG and CLK_DATA_REG.

```
#define      SPI_SLAVE_SELECT
#define      SPI_CLOCK
```

The last set of defines are the SPI control flags. The driver expects the defines for enabling the SPI, checking if the SPI receive full flag and checking the SPI transmit empty flag in the SPI hardware

```
#define      SPI_ENABLE
#define      SPI_RECEIVER_FULL
#define      SPI_TRANSMIT_EMPTY
```

Chapter 3: Flash Driver Function Calls

This chapter describes all of the API function calls for the flash driver.

Erase_Page

Description: This function will erase the page in the flash device

Parameters:

uint16 page_number	This is the page number to be erased.
--------------------	---------------------------------------

Prototype: `void Erase_Page(uint16 page_number)`

Example:

```
Erase_Page (page_number);
```


Flash_Refresh

Description: This function will refresh all pages from the passed in start page location to the final passed in page location.

Parameters:

uint16 start_page	This is the first page number to refresh.
uint16 finish_page	This is the last page number to refresh

Prototype: void Flash_Refresh(uint16 start_page, uint16 finish_page)

Example:

```
Flash_Refresh(0, 100);
```

Move_Flash_To_Buffer

Description: This function will move the specified flash block into the internal ram buffer(buffer1).

Parameters:

uint16 page_number	This is the page number to be copied into the buffer.
--------------------	---

Prototype: void Move_Flash_To_Buffer(uint16 page_number)

Example:

```
Move_Flash_To_Buffer(uint16 page_number);
```

Program_Buffer

Description: This function will take the contents stored in the internal buffer and program them to the specified page number.

.

Parameters:

uint16 page_number	This is the page number where the buffer contents will be programmed to.
--------------------	--

Prototype: `void Program_Buffer(uint16 page_number)`

Example:

```
Program_Buffer (page_number);
```

Read_Buffer

Description: This function will read the contents from the internal flash buffer.

.

Parameters:

uint16 byte_location	This is where to start reading inside the buffer
uint8 * data_pointer	This is the pointer used to place the data
uint16 length	This is the amount of data to be read from the buffer

Prototype: `void Read_Buffer(uint16 byte_location, uint8 * data_pointer, uint16 length)`

Example:

```
uint8 Data[16];  
  
Read_Buffer(0, &Data[0], 16);
```

Read_Flash

Description: This function will read directly from the flash part and store the data into the specified data space.

Parameters:

uint16 page_number	This is the page number to read from
uint16 byte_location	This is where to start reading inside the page
uint8 * data_pointer	This is the pointer used to place the data
uint16 length	This is the amount of data to be read from the flash

Prototype: `void Read_Flash(uint16 page_number, uint16 byte_location, uint8 * data_pointer, uint16 length)`

Example:

```
uint8 Data[16];  
  
Read_Flash(0,0, &Data[0],16);
```

Write_Buffer

Description: This function will take the data_pointer and put the data into the internal flash buffer starting at the specified byte location.

.

Parameters:

uint16 byte_location	This is where to start reading inside the buffer
uint8 * data_pointer	This is the pointer used to place the data
uint16 length	This is the amount of data to be read from the buffer

Prototype: `void Write_Buffer(uint16 byte_location, uint8 * data_pointer, uint16 data_size)`

Example:

```
uint8 Data[16];  
  
Write_Buffer (0, &Data[0],16);
```

Write_Flash_Data

Description: This function will take the pointer of the user supplied data and write it into the flash at the specified page and byte offset.

Parameters:

uint16 page_number	This is the page number to write to.
uint16 byte_location	This is where to start writing inside the page
uint8 * data_pointer	This is the pointer used to get the data.
uint16 length	This is the amount of data to be write to the flash

Prototype: void Write_Flash_Data(uint16 page_number, uint16 byte_location, uint8 * data_pointer, uint16 data_size)

Example:

```
uint8 Data[16];  
  
Read_Flash(0,0, &Data[0],16);
```

Appendix A: Memory Structures and Enumerates

Flash Size enumerate

```
enum FLASH_SIZE
{
    SIZE_1_MEG = 3,
    SIZE_2_MEG = 5,
    SIZE_4_MEG = 7,
    SIZE_8_MEG = 9,
    SIZE_16_MEG = 11,
    SIZE_32_MEG = 13,
    SIZE_64_MEG = 15,
    SIZE_128_MEG = 4,
    SIZE_256_MEG = 6,
    SIZE_512_MEG = 8,
    SIZE_1_GIG = 10,
    SIZE_2_GIG = 12,
    SIZE_4_GIG = 14
};
```

Flash Buffer Size

```
enum FLASH_BUFFER_SIZE
{
    BUFFER_264 = 0,
    BUFFER_528,
    BUFFER_1056,
    BUFFER_2112
};
```


Flash Structure info

```
typedef struct FLASH_FILE_Info
{
    uint16 file_system_start;
    uint16 file_system_size;
    uint16 buffer;
    uint16 buffer_overhead;
    enum FLASH_BUFFER_SIZE buffer_size;
    enum FLASH_SIZE flash_size;
    uint16 percent_size;
}FLASH_FILE_Info;
```

Flash Commands

#define	STATUS_COMMAND	0xD7
#define	READ_CONTINUOUS_COMMAND	0xE8
#define	READ_MAIN_MEMORY_COMMAND	0xD2
#define	BUFFER1_READ_COMMAND	0xD4
#define	BUFFER2_READ_COMMAND	0xD6
#define	BUFFER1_WRITE	0x84
#define	BUFFER2_WRITE	0x87
#define	BUFFER1_PROGRAM_ERASE	0x83
#define	BUFFER2_PROGRAM_ERASE	0x86
#define	BUFFER1_PROGRAM	0x88
#define	BUFFER2_PROGRAM	0x89
#define	PAGE_ERASE	0x81
#define	BLOCK_ERASE	0x50
#define	BUFFER1_WRITE_AND_PROGRAM	0x82
#define	BUFFER2_WRITE_AND_PROGRAM	0x85
#define	MEMORY_TO_BUFFER1	0x53
#define	MEMORY_TO_BUFFER2	0x55
#define	COMPARE_MEMORY_TO_BUFFER1	0x60
#define	COMPARE_MEMORY_TO_BUFFER2	0x61
#define	BUFFER1_REWRITE	0x58
#define	BUFFER2_REWRITE	0x59

Index

D

Driver Configuration, 2

E

Erase_Page, 5

F

Flash Buffer Size, 13
Flash Commands, 14
Flash Size enumerate, 13
Flash Structure info, 14
Flash_Refresh, 6
Function Calls, 4

I

I/O, 2
Introdcution, 1

M

Move_Flash_To_Buffer, 7

P

Program_Buffer, 8

R

Read_Buffer, 9
Read_Flash, 10

S

Spi Config, 3
Starting, 2

W

Write_Buffer, 11
Write_Flash_Data, 12

