Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 002-04885

Spec Title: AN204885 - F2MC-FM3 FAMILY WITH USB HOST MASS STORAGE BOOTLOADER

Replaced by: None

AN204885

F²MC-FM3 Family with USB Host Mass Storage Bootloader

This application note describes the implementation of a bootloader which uses USB mass storage support driver to read a Motorola S-Record file named *app.mhx* in root directory of the storage device and flash it as main application. Supported file systems are FAT, FAT16 and FAT32. An optional debug information is displayed via UART 0 (115200,N,1).

Contents

1 Introduction

1.1 Overview

This application note describes the implementation of a bootloader which uses USB mass storage support driver to read a Motorola S-Record file named *app.mhx* in root directory of the storage device and flash it as main application. Supported file systems are FAT, FAT16 and FAT32. An optional debug information is displayed via UART 0 (115200,N,1).

The software is developed for the Cypress FM3 series USB microcontroller. This example is designed primarily for Cypress microcontroller with existing evaluation boards.

Also in this application note a FAT file system is used which can be downloaded from the developer's website: http://elm-chan.org/fsw/ff/00index_e.html

1.2 Features

- Supports USB stick / USB card reader (tested with max. 320GB USB hard drive)
- Supports FAT, FAT16 and FAT32
- Supports reset vector check (If not valid, load bootloader)
- Starts bootloader only at power on (except no valid Reset Vector)
- Needs max. 16KByte Flash
- Debug Information via UART 0

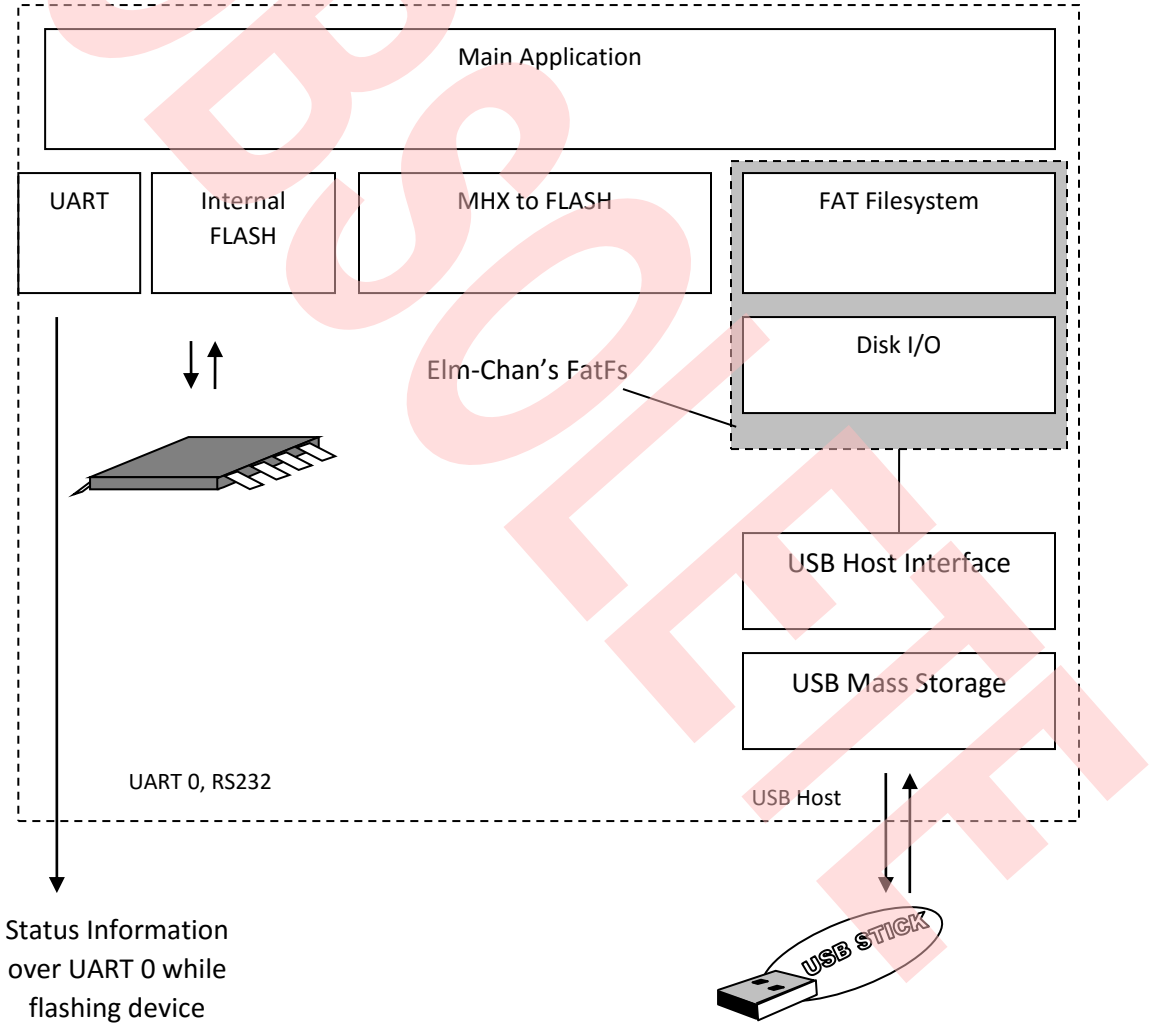
2 Overview

Overview of the USB/SD Card Bootloader

2.1 Project Parts

- USB Host & USB Mass Storage – Disk I/O Driver
- FatFs Module (FAT, FAT16, FAT32 driver) from Elm-Chan
- Internal Flash Routines
- Main Application (bootloader) and MHX/S-Record File Converter
- UART for Debug Information

2.1.1 Project Block Diagram



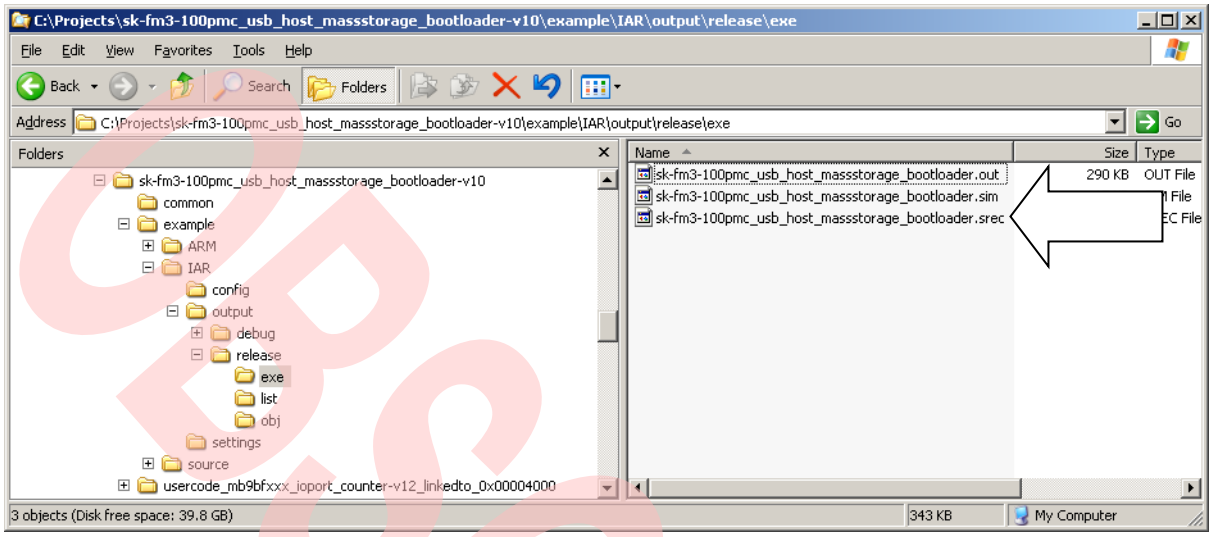
2.2 Quick Start using SK-FM3-100PMC

Choose *sk-fm3-100pmc_usb_host_massstorage_bootloader-v10.zip* located at website:

http://www.cypress.com/SK-FM3-100PMC

or located in all software examples:http://www.cypress.com/cypress-mcu-product-softwareexamples

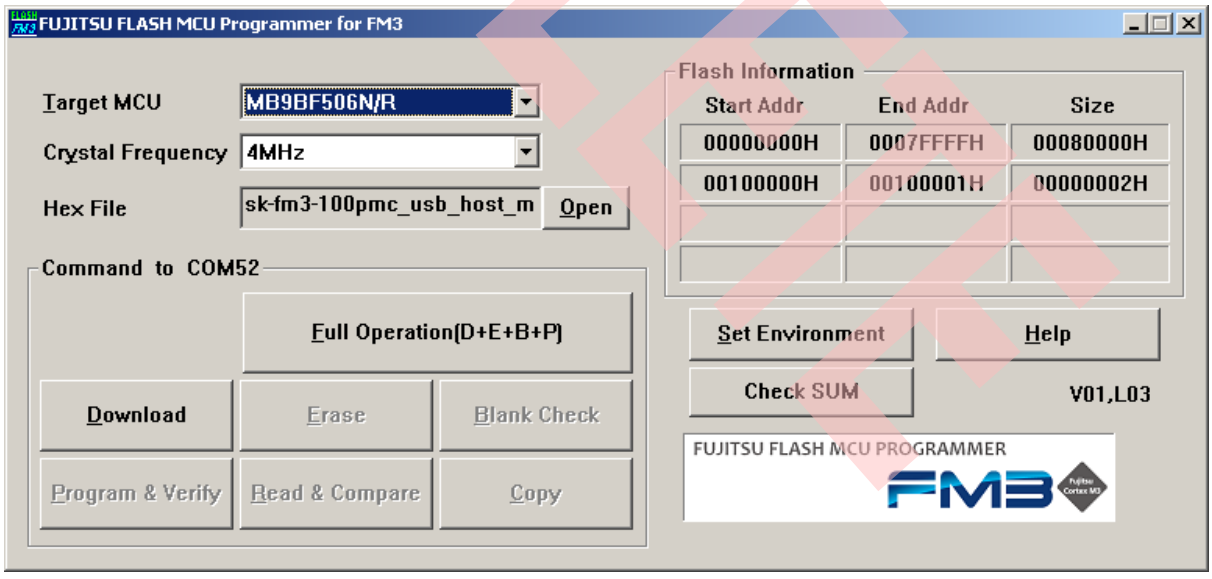
Extract the zip file and navigate to *example\IAR\output\release\exe*:



Here a *.srec file should be located:

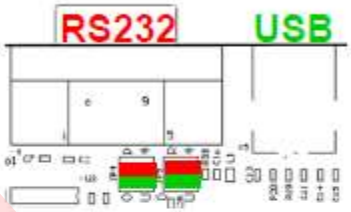
sk-fm3-100pmc_usb_host_massstorage_bootloader.srec

This file must be flashed using the Cypress USB Direct Programmer (via USB) or the Cypress FLASH MCU Programmer for FM3 (via UART). In this case the Cypress FLASH MCU Programmer for FM3 will be used.



Via “Set Environment” the COM port for programming the evaluation board can be chosen and details of evaluation board can be found on http://www.cypress.com/sk-fm3-100pmc. Programming via UART only works with UART 0. There are two different options: Using RS232 or USB to UART converter for use with UART 0 / 4.

- UART0 = USB-connector (X5), UART4 = Sub-D9 (X4) (default)
  - Setting of Jumper JP4 and JP5: U-0 / R-4



(default)

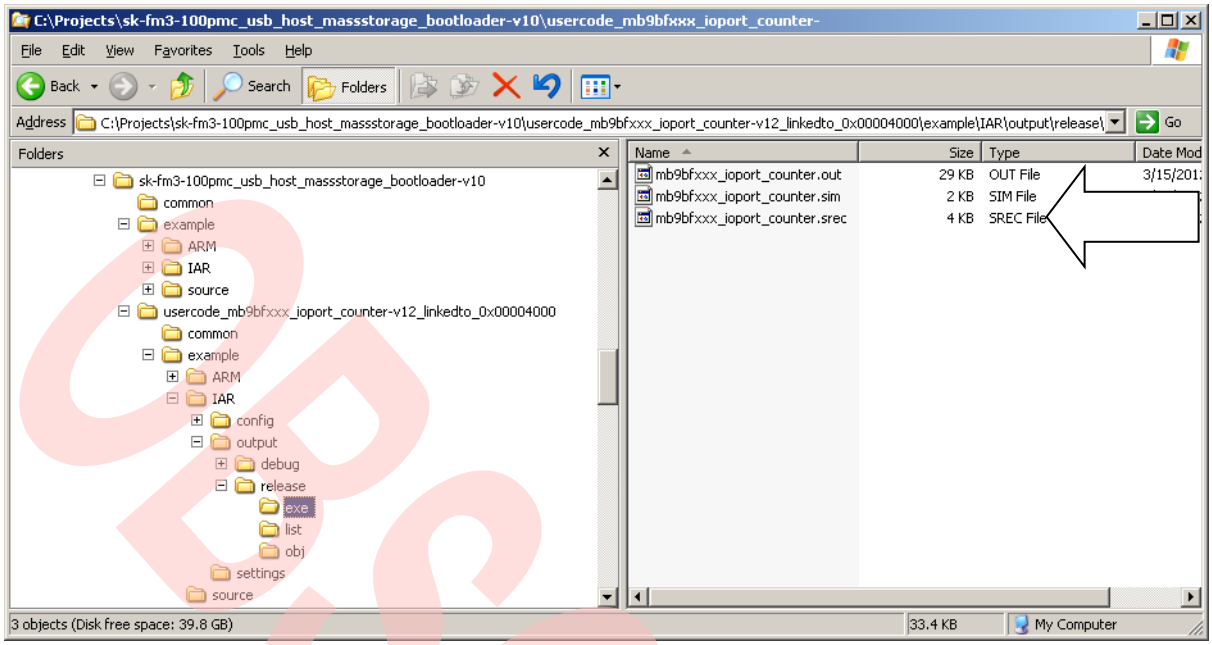
- UART0 = Sub-D9 (X4), UART4 = USB-connector (X5)
  - Setting of Jumper JP4 and JP5: U-4 / R-0

Switch the board to programming mode and press “Full Operation” and following the instructions.

After programming is completed, switch back to run mode and press reset on the evaluation board. The bootloader starts only at power on or if no user code application exists. Because no user application code exists, the bootloader will start. Plug in an USB stick into the PC and copy an application code named to *app.mhx* to root directory. Now this USB stick can be used with the evaluation board. To re-flash later an application, the USB stick must be plugged in before powering the evaluation board.

An example linked to `0x0000_4000` can be found in the example folder:

usercode_mb9bfxxx_ioport_counter-v12_linkedto_0x00004000\example\IAR\output\release\exe



mb9bfxxx_ioport_counter.srec must be renamed to *app.mhx* and must be placed at the usb stick. After plugging in the USB stick into the evaluation board, the bootloader can start to flash the MCU.

For change linker settings in own applications, refer chapter 3.6.1.

The seven segment display shows following status:

01: Bootoader started

02: USB Stick detected

03: User Code found

04: Start erasing flash

05: Start flashing

00: Operation completed successfully, remove USB Stick

11: Error while erasing, System halt

12: Error CRC, System halt

If the seven segment reaches “00” the program was flashed and the USB stick can be removed to restart in user application code.

3 Bootloader Operation

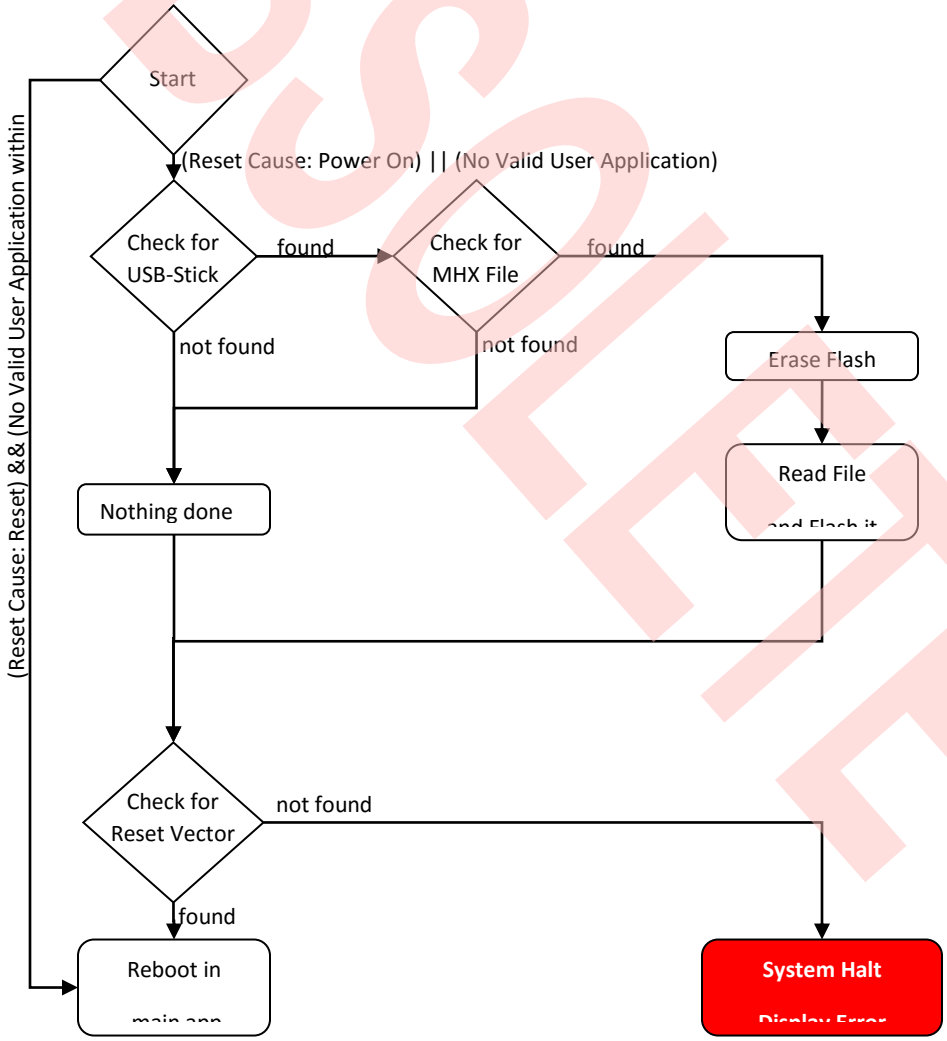
Bootloader Process

3.1 Bootloader operation flow-chart

The bootloader starts with some initialisations: Before it handles its own start code, it checks for a valid user application code and checks the power on state. If no valid user application code exists within the microcontrollers flash memory, the microcontroller would execute not wanted undefined code. So if it does not exist, the bootloader will be started instead of the main application. If the “Power On” state was a reset, the main application will be started instead no valid boot vector exists. If the reset cause was power on, the bootloader starts.

After USB-initialisation the bootloader enumerates the USB stick - if it is available - and tries to mount it. After a timeout of about 500ms the function gives up to find a valid USB mass storage. If no USB stick was found, the bootloader restarts in user application code. If a USB stick was found, it will be checked whether a file *app.mhx* exists in root directory. If the file was found, the bootloader begins to erase all Flash sections except the bootloader Flash sections and flashes the main application from *app.mhx*.

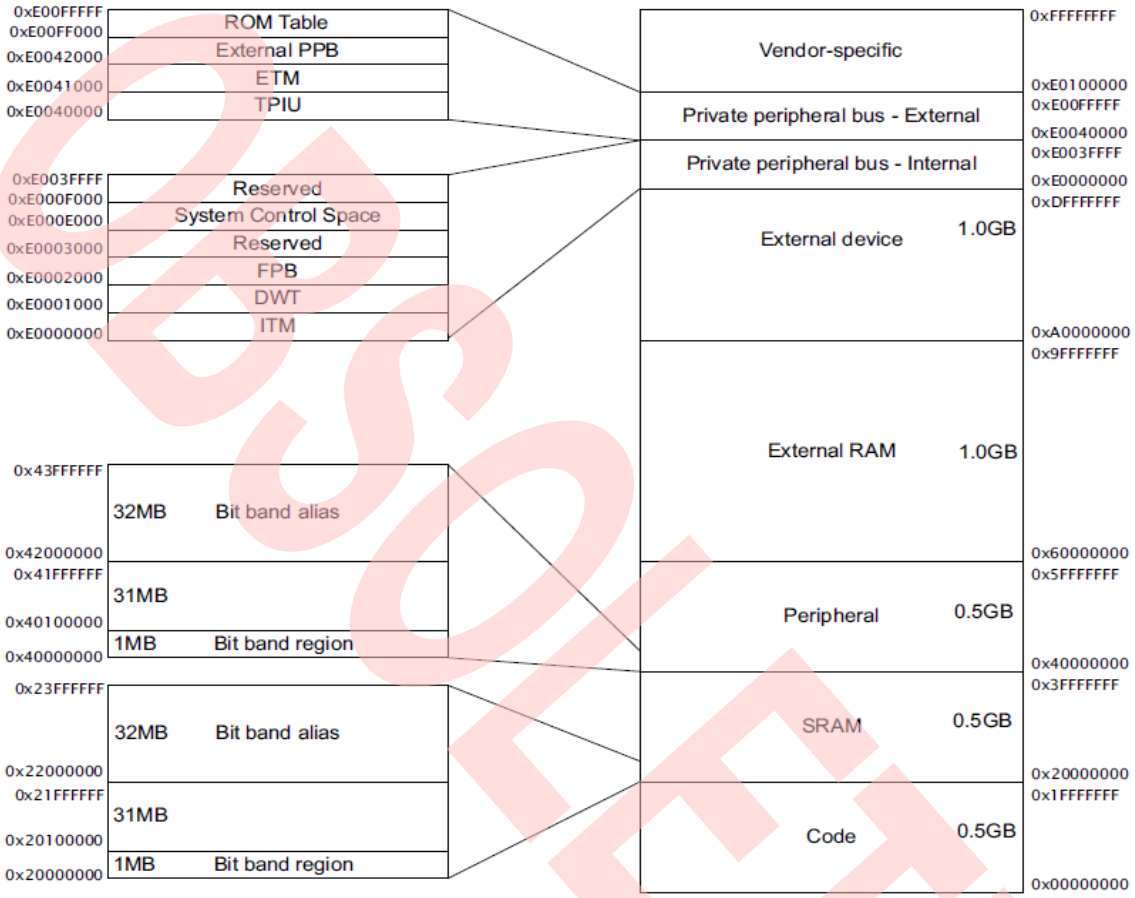
Note: The Power-On Flag is reset by reading it by the bootloader. It is not available for the main application anymore.



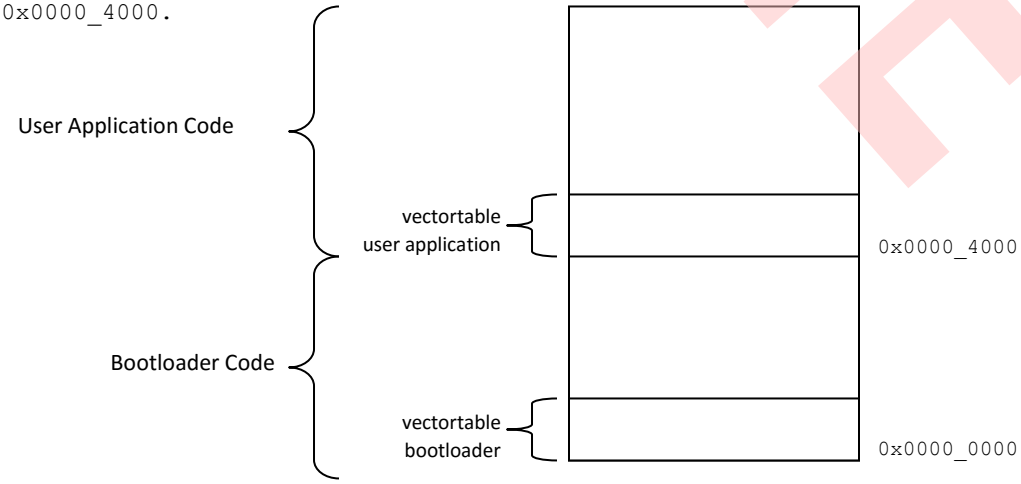
3.2 Memory Map

As described in the *Technical Reference Manual* of ARM for Cortex-M3 core, ARM Cortex M3 microcontrollers have its flash memory starting at `0x0000_0000`, RAM memory starting at `0x2000_0000` and so on.

Figure 1. Processor memory map (ARM Cortex-M3 Technical Reference Manual r2p0, 4-1)



As defined for ARM Cortex-M3 MCUs, the vector table starts with the stack pointer followed by the reset vector and so on. In case of using a bootloader, the vector table of the bootloader is fixed to `0x0000_0000` and the user code must be linked to upper areas of the flash memories. In case of Cypress bootloader, the user code starts `0x0000_4000`.



3.3 Jump into user application

To start the user application code, the stack pointer and vector table must be updated.
Following procedure is called, to start the user application code, linked to `0x0000_4000`:

```
#define USER_FLASH_START 0x00004000


int32_t main(void)

{

   BootloaderAPI_JumpBoot(USER_FLASH_START);

}


/**

 ************************************************************************

 ** \brief  Jump to user code application

 **

 ** \param  u32Address Address of user code

 ** \return none

 ************************************************************************/

#ifdef __ICCARM__

 void BootloaderAPI_JumpBoot(uint32_t u32Address)

 {

   __asm("LDR SP, [R0]");   //Load new stack pointer address

   __asm("LDR PC, [R0, #4]");          //Load new program counter address

 }

#elif  __CC_ARM

 __asm void BootloaderAPI_JumpBoot(uint32_t u32Address)

 {

   LDR SP, [R0]          ;Load new stack pointer address

   LDR PC, [R0, #4]                  ;Load new program counter address

 }
```

```
#else
 #error Please check compiler and linker settings
#endif


/**
 ******************************************************************************
 ** \brief  Execute main application
 **
 ** \param  none
 ** \return none
 ******************************************************************************/
void BootloaderAPI_ExecuteUserApplication(void)
{
   //Change the Vector Table to the USER_FLASH_START
   SCB->VTOR = USER_FLASH_START & 0x1FFFFF80;
   BootloaderAPI_JumpBoot(USER_FLASH_START);
}
```

The code relocates the vector table, loads the new stack pointer (starting at the user code vector table) and sets the program counter to the user code application start.

3.4 Bootloader start condition

Cypress FM3 series MCUs have a reset cause register. The reset cause register recognizes which kind of reset was executed. The bootloader starts only at power on. All other resets (software reset, hardware reset, watchdog reset, etc.) will execute the user code application.

```
/* If reset cause was not power on, start user application, if it is valid */

if ((bFM3_CRG_RST_STR_PONR == 0) && (BootloaderAPI_UserCodeValid() == TRUE))

{

   BootloaderAPI_ExecuteUserApplication();

}
```

3.5 MHX Format Conversion

3.5.1 S-Record Structure

The MHX Format is a Motorola-S-Record file, which has the ability to flash only used sections and leave sections, which are not used untouched. Each line represents a record. There are 10 different types of records (S0 – S9); however the mhx converter in the bootloader handles only the S1..3 type. The line is structured as followed:

3.5.1.1 Example: MHX Record

Data in S-Record:
S209F8B0788C7625096640

Start Code	Record Type	Byte Count	Address	Data	CRC
S	2	09	F8B078	8C76250966	40

Start-Code: Is every record line the first character and contains every time an “S”

Record-Type: Record Type (0-9), only 1..3 is supported

Byte-Count: Number of Bytes of record (Address + Data + CRC)

Address: Flash-Address (3 Bytes for S2 record)

Data: Data to write at the specified address

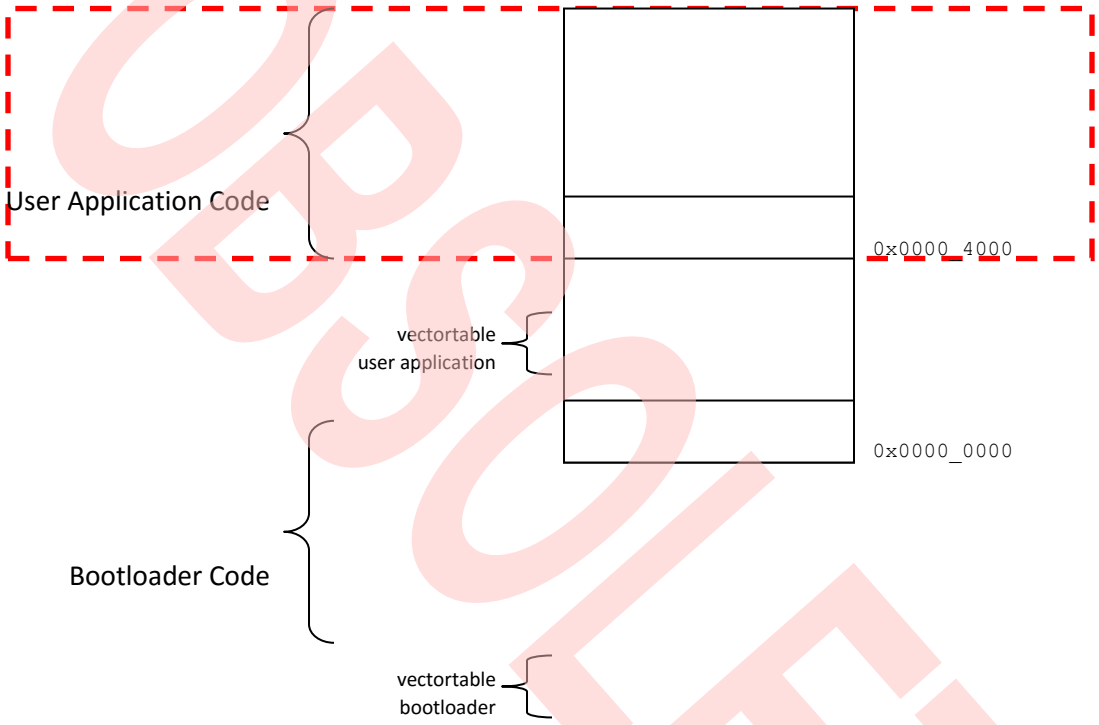
CRC: Checksum

3.6 Linker settings

For bootloader usage, linker settings for the bootloader and the user application must be changed.

3.6.1 Linker Settings User Code

For using the user application code with the bootloader in parallel, the user application must start from 0x0000_4000 instead of 0x0000_0000.



3.6.1.1 IAR user code linker settings

The following lines were changed in the standard FM3 template linker files (*.icf):

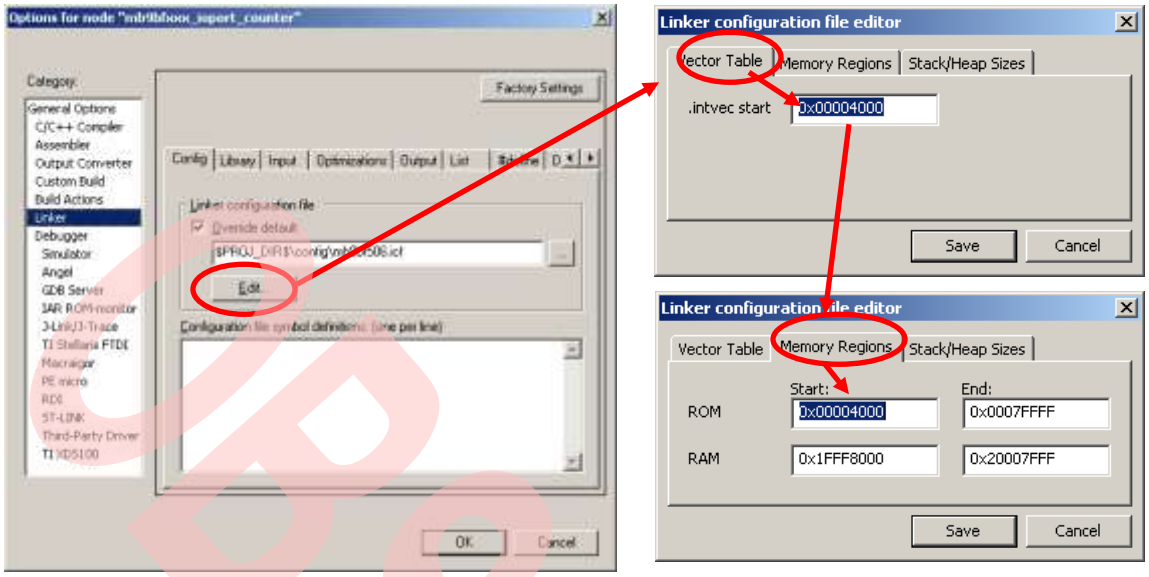
```
/*-Specials-*/

define symbol __ICFEDIT_intvec_start__ = 0x00004000;

/*-Memory Regions-*/

define symbol __ICFEDIT_region_ROM_start__ = 0x00004000;
```

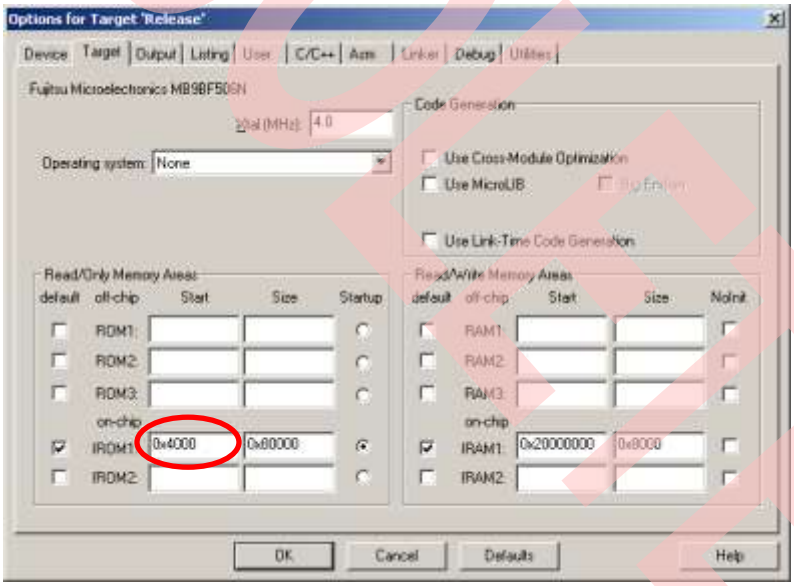
This can be done also via Project -> Options -> Linker. Here .intvec start and ROM start can be specified as well.



3.6.1.2 Keil user code linker settings

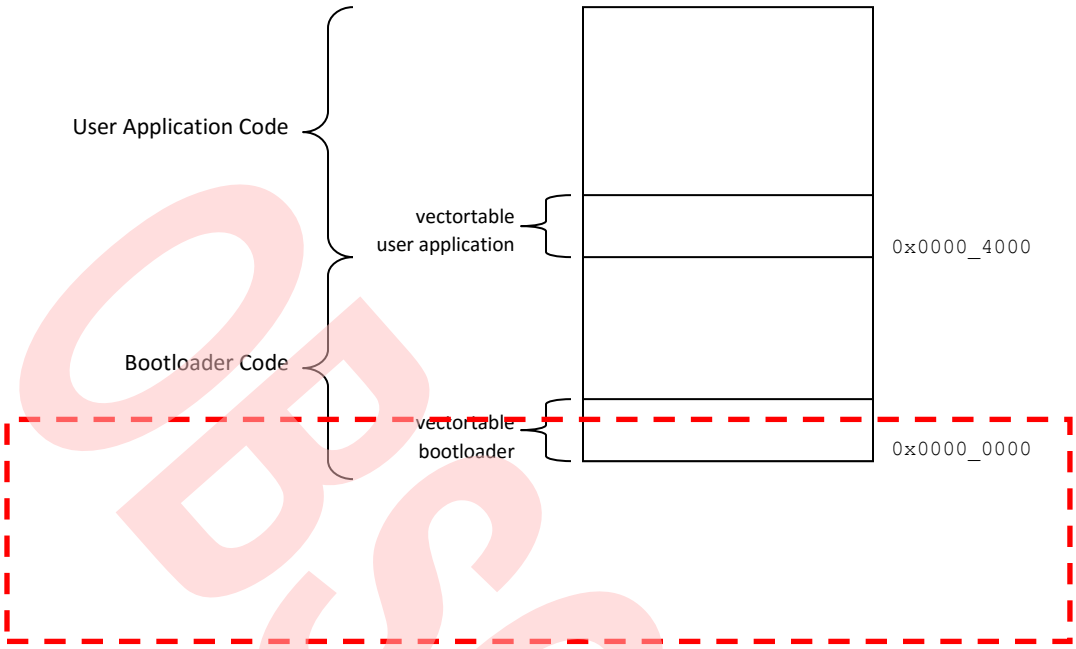
Project -> Options for Target 'Release' -> Target:

IROM1 - Start must be changed from 0x0 to 0x00004000



3.6.1.3 Linker Settings Bootloader

For writing flash, flash routines must be copied into RAM. This can be configured in the linker files and is normally done by the IDE startup code. To be on the safe side, the ROM (flash) area should be set to use only `0x0000_0000` to `0x0000_3FFF`.


User Application Code
Bootloader Code
vectortable
user application
0x0000_4000
vectortable
bootloader
0x0000_0000

3.6.1.4 IAR bootloader linker settings

For using RAM code for Flash erase/programming routines copied to RAM automatically by start-up and executed later in RAM, the following lines were added to the standard FM3 template linker files (*.icf):

```
define symbol __RAM_func_start__    = 0x20000000;

define symbol __RAM_func_end__      = 0x20007FFF;

define region RAM_func_region =   mem:[from __RAM_func_start__ to
__RAM_func_end__];


define block RamCode {section .flash_ram_code};

place in RAM_func_region  { block RamCode };
```

3.6.2 Keil bootloader linker settings

For using RAM code for Flash erase/programming routines copied to RAM automatically by start-up and executed later in RAM, the following adjustment was done:

Options for flash.c -> Properties -> Memory Assignment -> Code / Const:

IRAM (0x2000000-0x20007FFF)

Read application note mcu-an-300401-e-fm3_flash_programming for further details.

Appendix A Appendix

A.1 FatFs Module (FAT, FAT16, FAT32 driver)

For more information refer visit the developer's website:
http://elm-chan.org/fsw/ff/00index_e.html

Document History

Document Title: AN204885 - F²MC-FM3 Family with USB Host Mass Storage Bootloader

Document Number: 002-04885

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MSCH	03/20/2012	Initial Release
*A	5043548	MSCH	02/17/2016	Migrated Spansion Application Note from MCU-AN-300408-E-V11 to Cypress format
*B	5874581	AESATMP8	09/06/2017	Updated logo and Copyright.
*C	6021684	MBGR	01/09/2018	WOFR requested this document be made obsolete.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709