µTasker – Using the Bare Minimum Boot-loader for the M5223X

Introduction

The µTasker "Bare-Minimum" Boot-loader allows software uploading via Ethernet or the Internet to be performed with only a small boot loader, occupying one FLASH sector (2k) on the M5223X.

The Bare-Minimum Boot-loader is detailed in the document "uTaskerBoot.doc".

This document explains the method for using and programming the project for the M5223X and the Code Warrior V6.3 project. It includes full details of operating with encrypted upload files and also external SPI FLASH. The SPI FLASH supported are the following types from ATMEL (128k to 2Meg byte):

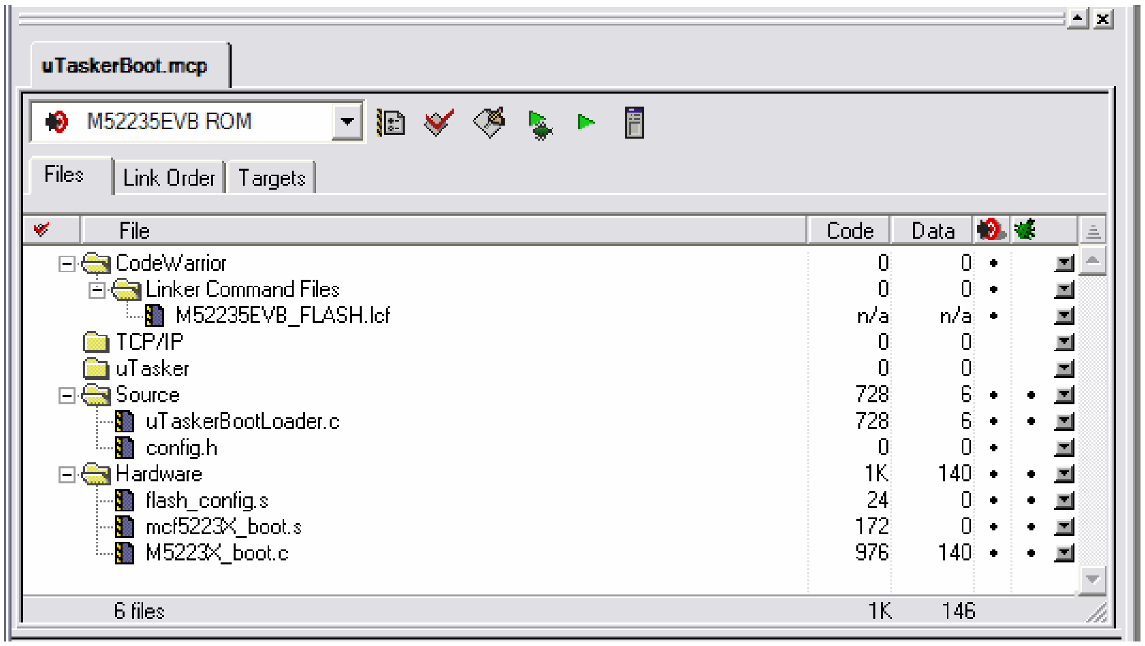
`AT45DB011B, AT45DB021B, AT45DB041B, AT45DB041D, AT45DB081B, AT45DB081D, AT45DB161D`

Projects and Targets

There are two CodeWarrior projects to be used to achieve the goals of programming the project with "Bare-Minimum" loading support. These projects are delivered in the µTasker demo project and so can be used immediately for first tests and also as a basis for further projects.

Both projects can be opened in the CodeWarrior IDE at the same time, making their use also very comfortable.

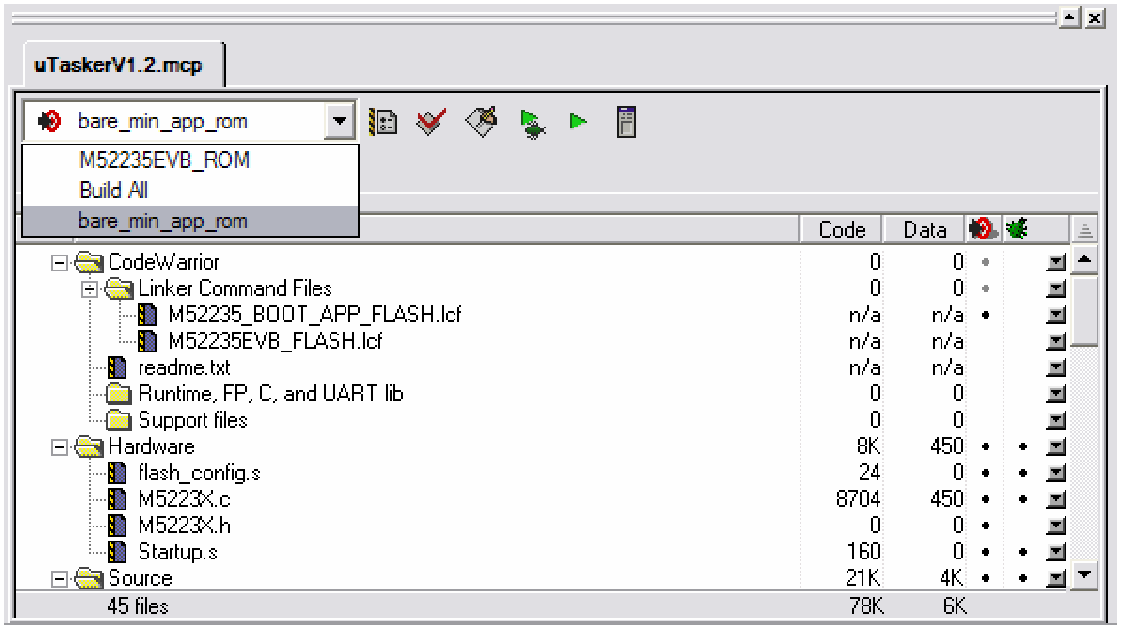
The boot project is shown in the project window below. It has only one target (M52235EVB ROM) and compiles and links to take over the start-up sequence of the target.



File	Code	Data		
CodeWarrior	0	0	●	
Linker Command Files	0	0	●	
M52235EVB_FLASH.lcf	n/a	n/a	●	
TCP/IP	0	0		
uTasker	0	0		
Source	728	6	●	●
uTaskerBootLoader.c	728	6	●	●
config.h	0	0	●	
Hardware	1K	140	●	●
flash_config.s	24	0	●	●
mcf5223X_boot.s	172	0	●	●
M5223X_boot.c	976	140	●	●
6 files	1K	146		

Once this project has been compiled, it can be loaded to the target using the normal loading sequence (see the µTasker tutorial for the M5223X) and then occupies the first 2k of FLASH (4k with external SPI FLASH support).

When loading the boot loader for the first time it is advisable to delete its sector and any application code. If a file system is to be left intact just delete up to its start location (eg. 0…0x17fff in the demo project). This is also discussed in the µTasker tutorial.



	Code	Data		
CodeWarrior	0	0	•	
Linker Command Files	0	0	•	
M52235_BOOT_APP_FLASH.lcf	n/a	n/a	•	
M52235EVB_FLASH.lcf	n/a	n/a		
readme.txt	n/a	n/a		
Runtime, FP, C, and UART lib	0	0		
Support files	0	0		
Hardware	8K	450	•	•
flash_config.s	24	0	•	•
M5223X.c	8704	450	•	•
M5223X.h	0	0	•	
Startup.s	160	0	•	•
Source	21K	4K	•	•
45 files	78K	6K		

This project window shows the µTasker demo project which has two target configurations: M52235EVB and bare_min_app_rom.

M52235EVB is the target used when no boot loader is to be used. It is linked to the standard reset vector location (0x00000000) and can be loaded alone to the target, meaning that it neither needs, or works together with, the boot loader. Of course, this target configuration will not support software uploads based on the bare-minimum boot loader principle.

The target generates the code for “uTasker_full.elf”

bare_min_app_rom is used when the project is to support the bare minimum boot loader or if the code is to be uploaded to a board supporting the boot loader principle.

The code is linked to start at either the address 0x00000800 (the second FLASH sector in the M5223X) or at the address 0x00001000 (the third FLASH sector).

The target generates the code for “uTasker_BM.elf”

There are various options supported by the boot loader. The options must be correctly set in the boot loader project and also in the application project which is later to be uploaded to the card.

The following settings need to be made.

Boot loader project

	Config.h	uTaskerBootLoader.c	Start address of application code
Internal FLASH	No defines	_ENCRYPTED disabled	0x800 (2k bootloader)
Internal FLASH with encryption	No defines	_ENCRYPTED enabled	0x800 (2k bootloader)
SPI FLASH	SPI_SW_UPLOAD	_ENCRYPTED disabled	0x1000 (4k bootloader)
SPI FLASH with encryption	SPI_SW_UPLOAD	_ENCRYPTED enabled	0x1000 (4k bootloader)

Other settings depend on the exact location of code and its size, plus conversion program settings as described in the sections below.

Application project

	Config.h	uTaskerConvert.exe parameters as per example bat file	Start address of application code
Internal FLASH	No defines	BM-Convert.bat	0x800 (2k bootloader)
Internal FLASH with encryption	No defines	BM-Convert_Encrypt.bat	0x800 (2k bootloader)
SPI FLASH	SPI_SW_UPLOAD	BM-Convert.bat	0x1000 (4k bootloader)
SPI FLASH with encryption	SPI_SW_UPLOAD	BM-Convert_Encrypt.bat	0x1000 (4k bootloader)

For more details about the use of uTaskerConvert.exe see the sections below.

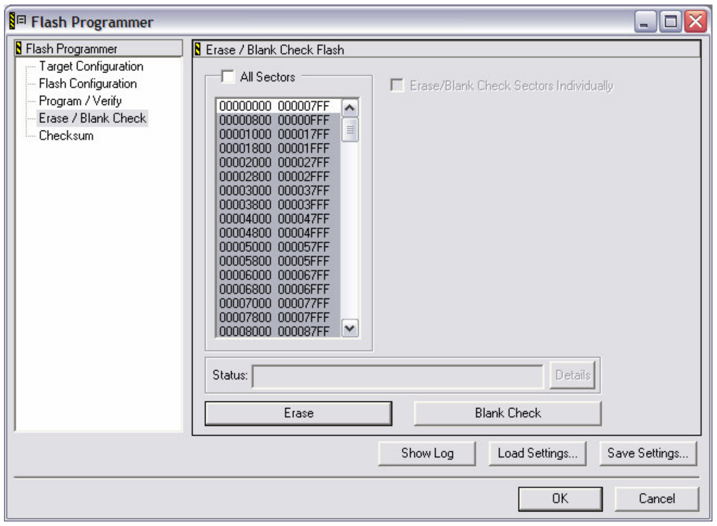
Since the application code needs to start at a slightly higher address when SPI FLASH is used (the boot loader code size increased due to the necessity of having both an internal FLASH driver and SPI FLASH at the same time), the application linker script file must be configured accordingly. To do this, open the file M52235EVB_BOOT_APP_FLASH.lcf and set the flash start location and size according by commenting the appropriate setting in/out.

```
MEMORY
{
    flash    (RX)    : ORIGIN = 0x0000800, LENGTH = 0x0003F800     /* Use this for
boot loader from internal FLASH */
 /* flash    (RX)    : ORIGIN = 0x0001000, LENGTH = 0x0003F000 */ /* Use this for
boot loader with SPI FLASH {2} */
    vectorram(RWX) : ORIGIN = 0x20000000, LENGTH = 0x00000400
    sram     (RWX)   : ORIGIN = 0x20000400, LENGTH = 0x00007C00
    ipsbar   (RWX)   : ORIGIN = 0x40000000, LENGTH = 0x0
}
```

Programming the First Application to the Target Hardware

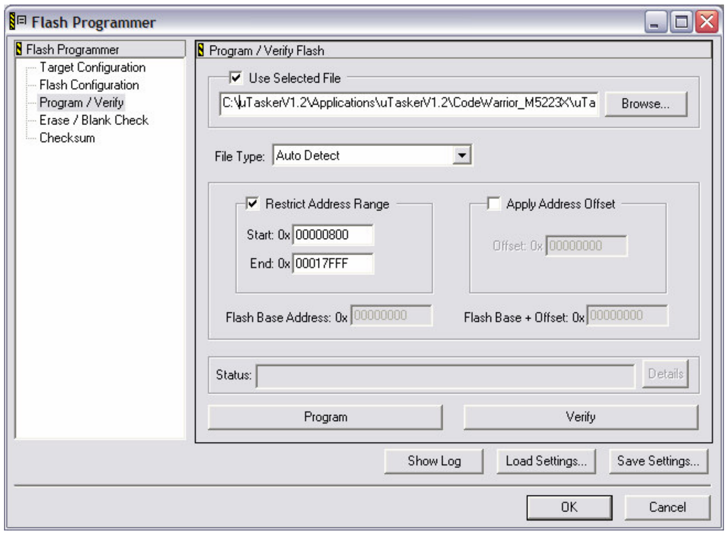
Assuming that the target has been loaded with the boot loader software and the *bare_min_app_rom* target is to be loaded to the target using the BDM, follow these programming instructions.

1. Build the project for the ***bare_min_app_rom*** target
2. Erase the target from 0x800 (0x1000 when SPI FLASH is used) to the start of the file system (for example 0x17fff when the file system start at 0x18000 – as does the demo project)



```
00000000  000007FF
00000800  00000FFF
00001000  000017FF
00001800  00001FFF
00002000  000027FF
00002800  00002FFF
00003000  000037FF
00003800  00003FFF
00004000  000047FF
00004800  00004FFF
00005000  000057FF
00005800  00005FFF
00006000  000067FF
00006800  00006FFF
00007000  000077FF
00007800  00007FFF
00008000  000087FF
```

Here the Flash Programmer has been set up to erase only the selected sectors. The first sector containing the boot loader is not selected and so will not be deleted (the first 2 sectors should not be selected when using SPI FLASH).



Use Selected File [✓]: C:\uTaskerV1.2\Applications\uTaskerV1.2\CodeWarrior_M5223X\uTa

File Type: Auto Detect

Restrict Address Range [✓]
Start: 0x 00000800
End: 0x 00017FFF

Apply Address Offset [ ]
Offset: 0x 00000000

Flash Base Address: 0x 00000000
Flash Base + Offset: 0x 00000000

When programming, the target "uTasker_BM.elf" should be programmed, rather than "uTasker_full.elf". To be sure that neither attempts are made to program to the boot loader sector or to the file system area, the address range has been restricted (example for internal FLASH – Start 0x1000 is correct for SPI FLASH), although not absolutely necessary if the correct file is selected.

Debugging with Code Warrior

Fortunately there is no difficulty involved when debugging the project with boot loader support. Simply start the debugging session as normal and the boot loader will start, see that there is no work to do and then jump to the start of the application code. This will be invisible to the user.

On the other hand, the project can also be started from the boot loader project, which will start and then also jump to the application code.

If the user has uploaded new software to the file system which should be copied, this process can also be stepped through in the boot loader project!!

So no worries, it is easy to use and works just great!

Uploading new software via Ethernet (internal FLASH with no Encryption)

In order to upload software, the application installed on the board must support the file transfer to the correct address. See "uTaskerBoot.doc" for more details. The µTasker demo project supports it via HTTP post and FTP.

The files generated by the ***bare_min_app_rom*** target include a binary file called *uTasker_BM.elf.bin*. This can not be loaded directly because it is missing an important header used for verification and authentication.

Unfortunately it doesn't seem possible to perform the conversion automatically together with the build procedure in CodeWarrior and so a bat file has to be executed to perform this conversion.

The file `BM-Convert.bat` in target directory …\CodeWarrior_M5223X\uTaskerV1.3\bin contains the following command:

```
uTaskerConvert.exe uTasker_BM.elf.bin H_Upload.bin -0x1234 -a748b6531124
```

This uses the conversion utility to take the generated file (`uTasker_BM.elf.bin`) and add the header (the code a748b6531124 corresponds to the boot loader setting so that only files generated with the correct key will be accepted – the key can of course be changed for your own project). The utility generates the output file H_Upload.bin. (Note that this file is automatically put to the correct location in the file system for an upload via FTP).

Since the demo project supports HTTP post, simply connect to the web server using a standard web browser and go to the page with the software upload support. Select the file to be uploaded and click on the "Upload new software" button. After about 10s the page will refresh and the new software will have been programmed in the meantime.

Even if the project doesn't have a web server supplying the upload support, the file can also be copied using FTP. Simply grab the file to be uploaded and drag it to a browser connected to the FTP serving running on the target. Reset the target and the boot loader will do the rest (if you are not next to the board this can also be commanded via the web browser on the administration page).

Both methods work well and even resets or power downs during the process will not cause catastrophic failure to occur. Again see the boot loader document for more information.

Uploading new software via Ethernet (internal FLASH with Encryption)

As in the case of the upload to internal FLASH without encryption, the binary file has to be first converted to the correct format. When in the correct format the file can be uploaded via HTTP post or FTP in the uTasker demo project.

The file `BM-Convert_Encrypt.bat` in target directory …\CodeWarrior_M5223X\uTaskerV1.3\bin contains the following command:

```
uTaskerConvert.exe uTasker_BM.elf.bin H_Upload_Enc.bin -0x1235 -
a748b6531124 -ff25a788f2e681338777 -afe1 -226a
```

This uses the conversion utility to take the generated file (`uTasker_BM.elf.bin`) and add the header (the code a748b6531124 corresponds to the boot loader setting so that only files generated with the correct key will be accepted – the key can of course be changed for your own project). The utility generates the output file `H_Upload_Enc.bin`. (Note that this file is automatically put to the correct location in the file system for an upload via FTP).

The magic number 0x1235 is chosen to ensure different to the version without encryption and the further parameters define the encryption key, a prime value for a pseudo-random number generator plus a file offset to increase the level of decryption difficulty.

`ff25a788f2e681338777` should always have a length divisible by 2 bytes where ff25 represents the first two bytes. The longer the sequence, the better the encryption but this also takes up space in the boot code and will thus have a limit.

`afe1` is a priming number which should never have the value of 0!

The parameters used must correspond exactly to the values in the configuration of the boot loader project (`uTaskerBootLoader.c`).

The resulting file will contain seemingly random data which will then be extracted by the loot loader to result in the operating application code at the final FLASH location.

Uploading new software via Ethernet (external SPI FLASH)

When working with external SPI FLASH, both the application and the boot loader must be compiled with the define SPI_SW_UPLOAD (config.h in each project). This will cause the boot loader to occupy 4k of FLASH space (the first 2 sectors) and so the application must also be linked to start at 0x1000 rather than 0x800 as in the case of the internal FLASH configuration. *See M52235EVB_BOOT_APP_FLASH.lcf and the description of the project settings above.*

The project can be set up to use encryption if required, as described in the previous section.

Note that the project supports HTTP Post of the generated file H_Upload.bin <u>but not FTP</u>. The reason is that FTP will upload this file to the internal file system whereas the HTTP post method recognises the file as a software upload and locates it specifically to the SPI FLASH. It is in fact possible to load the file via FTP but it requires that the file first be renamed to "HS.bin". This file will then be specially handled and located to the external SPI FLASH rather than the internal file system. Since it is not in the internal file system it will not be visible when the file system contents are viewed via FTP. Should it be necessary to retrieve a software upload file from the SPI FLASH (assuming that it has not already been deleted during the normal SW upload sequence – eg. when it contains corrupted data) this can be achieved by establishing an FTP connection (using DOS FTP) and then performing a "GET HS.bin" command. This will be recognised as the specific software file which will then correctly be extracted from the external SPI FLASH. The actual name of the software file is specified in app_hw_m5223x.h.

The details about the operation of the HTTP post and FTP are also specific to the realisation in the µTasker demo project and so could differ in user's own projects if required.

Should the project be configured for SPI FLASH use but no SPI FLASH chip be detected by the boot loader, no software programming will be attempted.

If the µTasker demo project doesn't detect the SPI FLASH device it will copy all uploaded data to the internal file system. In this case the uploaded file H_Upload.bin will be visible when viewing with FTP and thus indicates the fact that no SPI FLASH is in use (either not attached or defect).

Summary

This document has detailed the use the µTasker “Bare-Minimum” Boot-loader for the M5223X with CodeWarrior. It has described how the boot loader and the application can be loaded to the FLASH memory to perform a platform for further software upgrades via Ethernet.

Each time a new application software is created it can be compiled for the Bare-Minimum target and converted to a binary file for simple transfer to the target via HTTP post or FTP.

The next time the board is reset, the new program is automatically updated in FLASH, allowing unlimited changes to the application code including operating system, driver, TCP/IP stack and application modifications.

The process is safe in the event of resets or power cycles during the process and is also secure against malicious software upgrade attacks due to the security key which must match between the key configured in the boot loader and the encoded key in the software header.

In addition to the standard upload to internal FLASH, the use of external SPI FLASH memory has been discussed. For applications where the distribution of code in a readable form is not desirable, the method of encrypted upload has also been detailed, where this can be used with both internal and external SPI FLASH memory.

For full technical details of the boot loader see the document “uTaskerBoot.doc”.

Modifications V0.02 – 7.12.2006	Text corrections and addition of summary.
Modifications V0.03 – 27.2.2007	Added HTTP post support in demo project.
Modifications V0.04 – 27.8.2007	Added encryption and SPI FLASH support.