Flash Bootloader Hardware

Technical Reference

TMS470 Family

Version 1.06

Authors	Florian Hees, Marco Wierer, Andreas Wenckebach, Christian Bäuerle, Alexander Starke
Status	Released

Document Information

History

Author	Date	Version	Remarks
Florian Hees	2005-08-31	1.00	Creation, based on Technical Reference TMS470-Titan
Marco Wierer	2006-05-02	1.01	Revision and extension
Marco Wierer	2006-05-22	1.02	Updated document index for P-family
Andreas Wenckebach	2007-11-09	1.03	Added information for GHS compiler
Christian Bäuerle	2008-07-15	1.04	Added information for ARM- and Thumb state
Alexander Starke	2014-04-30	1.05	Adapted document to be more generic for different TMS470 derivatives

Reference Documents

No.	Source	Title
	TI	Texas Instruments “Device Reference Guide TMS470R1VF3xx”, “Device Reference Guide TMS470PVFxx”
	TI	Texas Instruments "TMS470 Family Platform F05 Flash Module Software Peripheral Driver User's Specification"

Contents

Illustrations

1 Introduction

This document covers the hardware-related particularities of the Flash Bootloader. It complements the explanations started in the user manual with hardware-specific details. All references there are resumed here in this document again and explained in detail.

The connection between a reference in the user manual and its specific description in this document is the headline. Both the reference and its explanation can be found below the same headline.

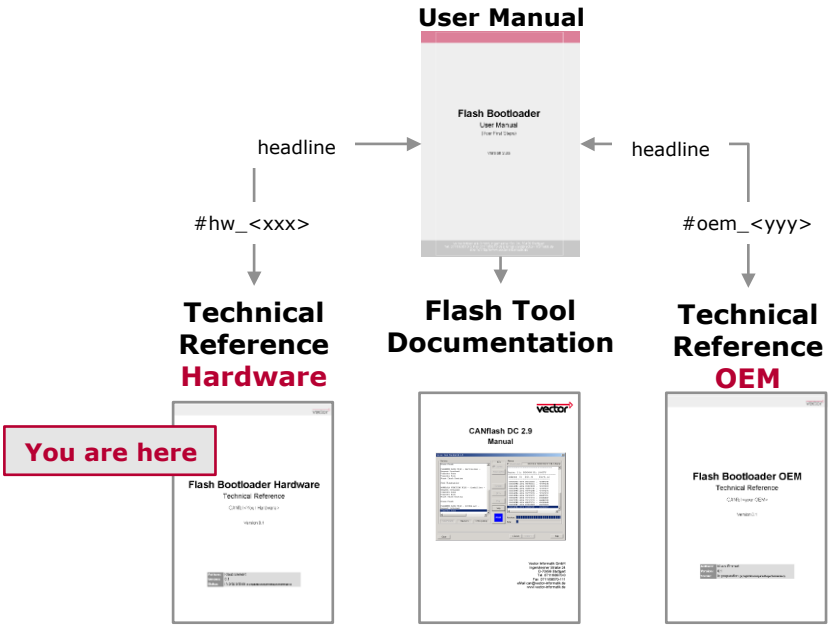

Figure 1-1 Manuals and References for the Flash Bootloader

Additionally this headline is marked with the ID of the reference from the User Manual. This ID looks like: **[#hw_<xxx>]**.

2 Memory Model

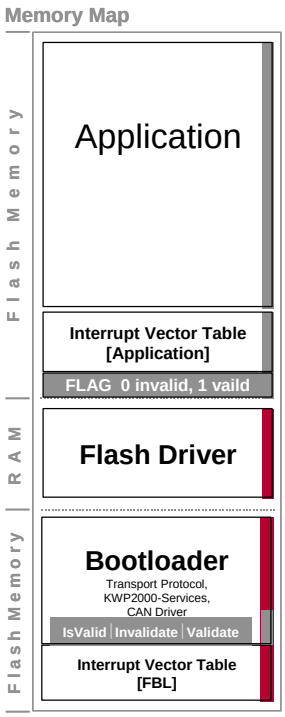

Figure 2-1 Memory Map

2.1 [#hw_mem] - Design the Memory Layout

For more general information about this see the UserManual_FlashBootloader in the chapter **Design Memory Layout**.

2.2 Memory Mapping

2.2.1 Memory Mapping of Application and FBL

The Bootloader is located in the first Flash sectors together with the interrupt vector table. These sectors are completely occupied by the Bootloader because they can only be erased entirely. The number of Flash sectors occupied by the Bootloader is OEM specific – check the map file for exact values.

2.2.2 Memory range of the FBL

The size of the FBL depends on the used car manufacturer specification, the configuration of the FBL as well as the implementation of the user call-back functions.

2.2.3 Memory Range of the Application

The application can start directly after the Flash Bootloader. The application areas must not overlap the Flash sectors which are used by the Flash Bootloader.

2.2.4 Compiler-/Linker specific memory mapping issues

2.2.4.1 Linker Command File

The following code shows a sample TI linker command file. Please note that depending on the used car manufacturer, some additional sections might be necessary. The linker command-file differs regarding the syntax, if an ARM or GHS compiler is used. Refer to the delivered DemoFbl-project to see a compiler and OEM specific example.

```
1.  /*****************************************************************************************/
2.  /* Options                                                                               */
3.  /*****************************************************************************************/
4.  -l rts32.lib
5.  /*****************************************************************************************/
6.  /* SPECIFY THE SYSTEM MEMORY MAP for VF348 */
7.  /*****************************************************************************************/
8.  MEMORY
9.  {
10.    VECTORS (X)     :  origin=0x00000000   length=0x00000020
11.    /* 256k - length - 0x10 because of protection keys */
12.    ROM LOW (RX)    :  origin=0x00000020   length=0x00001fd0
13.    PROTECT (RX)    :  origin=0x00001ff0   length=0x00000010  fill=0xffffffff
14.    ROM     (RX)    :  origin=0x00002000   length=0x0003E000
15.    /* 16k (block 0) + 16k (block 1) - Stack 2k */
16.    RAM     (RW)    :  origin=0x00401000   length=0x00001700
17.    STACKS  (RW)    :  origin=0x00400000   length=0x00001000
18.
19.    /* Peripherals */
20.    HETPROG (RW)    :  origin=0x00800000   length=0x00001000
21.
22.    /* SAR module addresses */
23.    MMC     (RW)    :  origin=0xFFFFFD00   length=0x40
24.    DEC     (RW)    :  origin=0xFFFFFE00   length=0x60
25.    SYS     (RW)    :  origin=0xFFFFFFD0   length=0x30
26.    FLASHCTL0 (RW) :  origin=0xFFE88000   length=0x00004000
27. }
28.
29. /*****************************************************************************************/
30. /* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY                                           */
31. /*****************************************************************************************/
32. SECTIONS
33. {
34.     .stack   : {
35.                       StackUSER   = .+ (0x1000 - (128+128+128+128+128));
36.                       StackFIQ    =  StackUSER   + 128;
37.                      _StackIRQ_   = _StackFIQ_   + 128;
38.                      _StackABORT_ = _StackIRQ_   + 128;
39.                       StackUND    =  StackABORT  + 128;
40.                       StackSUPER  =  StackUND    + 128;
41.                } > STACKS       /* SOFTWARE SYSTEM STACK             */
42.
43.     .bss     : {} > RAM         /* GLOBAL & STATIC VARS              */
44.     .sysmem  : {} > RAM         /* DYNAMIC MEMORY ALLOCATION AREA    */
45.     .data    : {} > RAM         /* INITIALIZED DATA                  */
46.
47.     .intvecs : {} > VECTORS     /* INTERRUPT VECTORS                 */
48.     .startup : {} > ROM         /* START ADDRESS                     */
49.     .text    : {} > ROM         /* CODE                              */
50.     .const   : {} > ROM         /* CONSTANT DATA                     */
51.     .cinit   : {} > ROM         /* INITIALIZATION TABLES             */
52.
53.     .MMC     : {_e_SARMMC_ST = .;}  > MMC
54.     .DEC     : {_e_SARDEC_ST = .;}  > DEC
55.     .SYS     : {_e_SARSYS_ST = .;}  > SYS
56.     .FLASHCTL0: {_e_FLASHCTL0_ST = .;} > FLASHCTL0
57.
58.     /* FBL header address */
59.     .FBLSTART        : {} > 0x2000
60.
61.     /* Flash driver buffer in RAM */
62.     .FLASHDRV        : {} > 0x00401200  /* locate FlashCode into RAM */
63.
```

```
64.     /* Application vector table */
65.     .APPLVECT        : {} > 0x001FFE0
66. }
```

Typically the following sections are used by the FBL:

Section Name	Address
.intvecs (or .intvect)	This is the location of the primary interrupt vector table (interrupt vector table of the Bootloader).
.FBLSTART (or .FBLHEADER)	Flash Bootloader header table start address. In this section the “FblHeader” structure is linked. This structure includes several information like the Bootloader version, the start address of the Bootloader, etc.[1] This address must be compliant to the address configured in the generation tool in the field “Header Address” in the FBL-options.
.APPLVECT	Section for the interrupt redirections to the interrupt service functions of the application. This section has to be linked to the same memory location in both application and bootloader
.FLASHDRV	Allocates the RAM buffer for the Flash driver. The size of the buffer is defined in the generation tool as the “Flash code buffer size (bytes)” parameter. This parameter must be large enough to hold the complete flashdriver. Link .FLASHDRV to the same address as used in your flashdrivers project (flashdriver is not relocatable!).

[1] Please refer to the manufacturer specific documentation for details

2.3 Example Memory Mapping (TMS470VF346)[2]

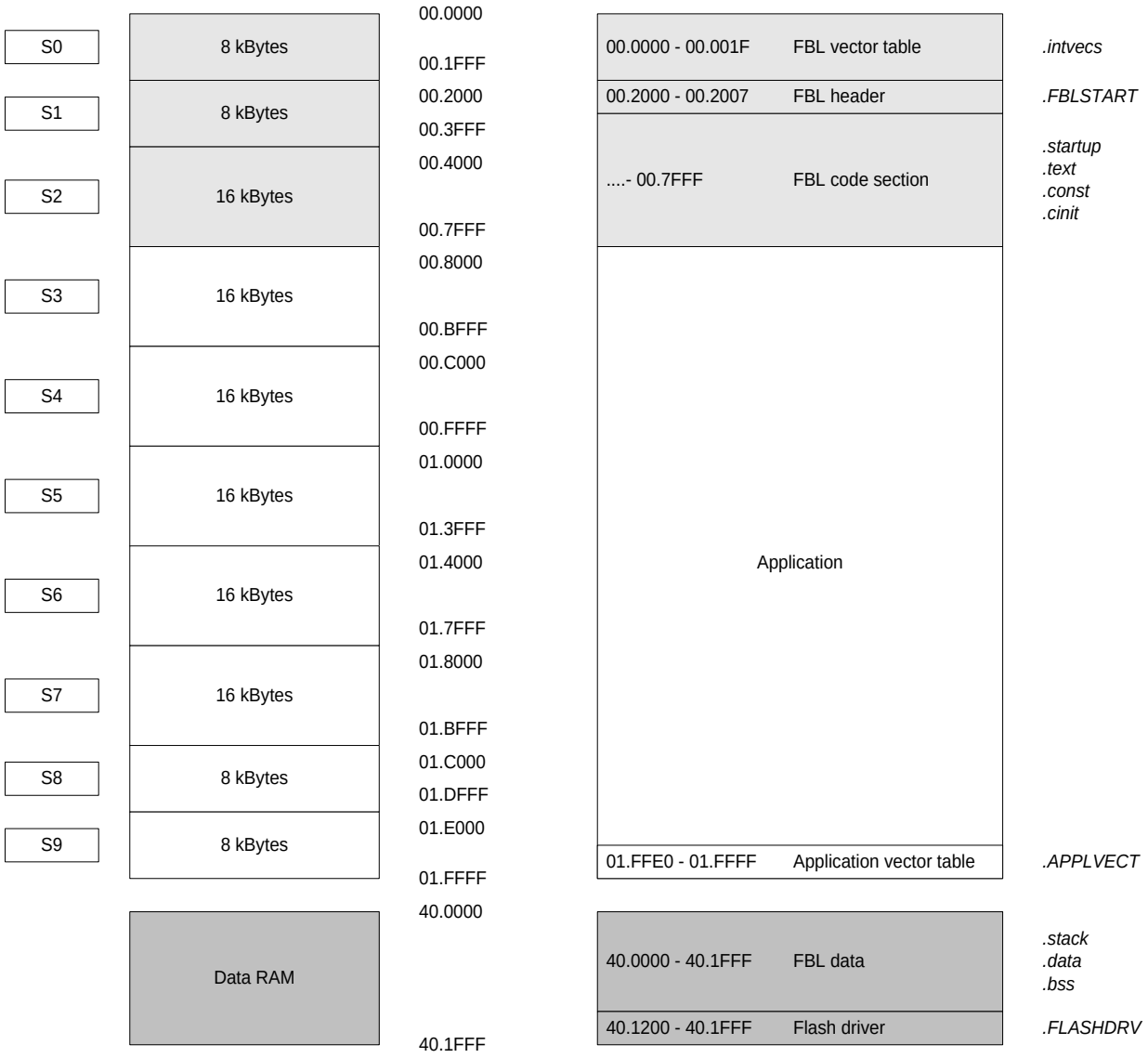

Figure 2-2 Example TMS470VF346 Memory Map

2.4 [#hw_intvect] - The Interrupt Vector Tables

For more general information about this see the UserManual_FlashBootloader in the chapter The Interrupt Vector Tables and Fbl_vect.c / Applvect.c(.h) - The Interrupt Vector Tables.

Each vector table entry of the TMS470 consists of four bytes which contain executable code and not only the address of the corresponding interrupt service routine(ISR). The interrupt vector table generated by the (Texas Instruments) compiler already consists of a jump-opcode and the ISR-address, so that this vector table can be used in connection with the Bootloader. The location of the application vector table is fixed and the same for the Bootloader as well as the application.
The application vector table linked to the Bootloader is used as a dummy table with all interrupts redirected to the startup code of the Bootloader (no interrupts allowed in the Bootloader).

[2] Please note that the actual values depend on the OEM and the configuration.

2.4.1 Flash Bootloader Interrupt Vector Table

The area from 0x00000000 to 0x0000001F is the primary interrupt vector table provided by the Bootloader.
Example Interrupt vector table (“fbl_vect.asm”):

```
1.     .state32
2.     .global  Start
3.     .global  isrirq
4.     .global  isrfiq
5.     .global  isruii
6.     .global _israpi
7.     .global _isradi
8.     .global _isrswi
9.
10.    .sect ".intvecs"
11.
12.   B  Start         ;   RESET INTERRUPT
13.
14.   B   isruii       ;   UNDEFINED INSTRUCTION INTERRUPT
15.   B   isrswi       ;   SOFTWARE INTERRUPT DISPATCHER
16.   B   israpi       ;   ABORT (PREFETCH) INTERRUPT
17.   B  _isradi       ;   ABORT (DATA) INTERRUPT
18.   B  Start         ;   RESERVED init with existing jump label
19.   B  _isrirq       ;   IRQ INTERRUPT
20.   B   isrfiq       ;   FIQ INTERRUPT
21.
22.    .end
```

All vectors within “fbl_vect.asm” except the reset vector are redirected to the application vector table.

Caution
Please note that the NMI’s are also redirected to the application vector table. Therefore, all NMIs have to be disabled in hardware while the Bootloader is active to prevent any accidental access of the application in the Bootloader.

Example Interrupt vector table (applvect.c) used in the Bootloader:

```
1.  #include "applvect.h"
2.
3.  asm("   .global  Start");
4.  asm("   .global _isrirq");
5.  asm("   .global _isrfiq");
6.  asm("   .global _isruii");
7.  asm("   .global  israpi");
8.  asm("   .global  isradi");
9.  asm("   .global _isrswi");
10.
11. /* This is the RESET INTERRUPT                                          */
12. #pragma DATA_SECTION (ApplIntJmpTable, ".APPLVECT")
13. const tIntJmpTable ApplIntJmpTable[1] = {APPLVECT FROM BOOT};
14.
15. /* This are the other Interruptvectors                                  */
16. asm("   .sect \".APPLVECT\"");
17. asm("_isruii");
18. asm("   B  #-8          ");   /* UNDEFINED INSTRUCTION INTERRUPT         */
19. asm(" isrswi");
20. asm("   B  #-8          ");   /* SOFTWARE INTERRUPT                      */
21. asm(" israpi");
22. asm("   B  #-8          ");   /* ABORT (PREFETCH) INTERRUPT              */
23. asm("_isradi");
24. asm("   B  #-8          ");   /* ABORT (DATA) INTERRUPT                  */
25. asm("   B  #-8          ");   /* RESERVED                                */
26. asm("_isrirq   ");
27. asm("   B  Start       ");    /* IRQ INTERRUPT                           */
28. asm("_isrfiq   ");
29. asm("   B  Start       ");    /* FIQ INTERRUPT                           */
```

2.4.2 Application Vector Table

The application interrupt vector table contains the redirections of the interrupt vector table to the interrupt service routine in the application.
If there is a valid application, the dummy application vector table defined in the Bootloader will be overwritten with the real application vector table.
The application interrupt vector table must be linked to the same memory location as in the Bootloader.
See example below for an implementation of the application vector table:

```
1.  #include "applvect.h"
2.
3.  asm("   .global  Start");
4.  asm("   .global   BrsHwUndefinedInstr");
5.  asm("   .global   BrsHwSwiDispatcher");
6.  asm("   .global  _BrsHwAbortPrefetchInt");
7.  asm("   .global  _BrsHwAbortDataInt");
8.  asm("   .global  _BrsHwIRQInterrupt");
9.  asm("   .global   BrsHwFIQInterrupt");
10.
11. asm("   .global  isrirq");
12. asm("   .global _isrfiq");
13. asm("   .global _isruii");
14. asm("   .global _israpi");
15. asm("   .global  isradi");
16. asm("   .global  isrswi");
17.
18. /* Reset vector */
19. asm("   .sect \".APPLVECT\"");
20. asm("   B  Start                    ");
21.
22. /* This are the other Interruptvectors */
23. asm(" isruii");
24. asm("   B  _BrsHwUndefinedInstr     ");   /* UNDEFINED INSTRUCTION INTERRUPT          */
25. asm("_isrswi");
26. asm("   B  _BrsHwSwiDispatcher      ");   /* SOFTWARE INTERRUPT                       */
27. asm(" israpi");
28. asm("   B   BrsHwAbortPrefetchInt   ");   /* ABORT (PREFETCH) INTERRUPT               */
29. asm("_isradi");
30. asm("   B  _BrsHwAbortDataInt       ");   /* ABORT (DATA) INTERRUPT                   */
31. asm("   B  #-8                      ");   /* RESERVED                                 */
32. asm(" isrirq   ");
33. asm("   B   BrsHwIRQInterrupt       ");   /* IRQ INTERRUPT                            */
34. asm("_isrfiq   ");
35. asm("   B  _BrsHwFIQInterrupt       ");   /* FIQ INTERRUPT                            */
```

3 Hardware Layer

This chapter describes CAN, timer and Flash driver configuration items

3.1 Flash Memory and Flash Driver

The Flash driver will be delivered as a Motorola S-Record or Intel Hex file. The user is usually not required to change this file.

There are some preconditions for correct working of Flash driver:

- DELAY parameter in the “fbl_cfg.h” (GENy) file has to be set to a valid value. This is a value needed by the Texas Instruments Flash library – see [2] for reference on how to set this value.
- The maximum CPU clock frequency for Flash modification access might be limited. For details refer to the respective documentation of the TI Flash API.
- The Flash driver might not support pipeline mode. This is derivative-specific.

Caution
Setting these parameters to incorrect values, the Flash could be permanently damaged and the data retention time might be affected.

3.1.1 Rebuilding the Flash Driver Binary

The Flash driver for the TMS470 is not relocatable and must be linked to the same address as the Flashcode array in the Bootloader (“fbl_flio.c” file). After the Flash driver download, the Bootloader copies the received Flash driver data to this Flashcode array. The Flashcode array is linked to a special section named “.FLASHDRV”.[3]

Example
If you want to rebuild the Flash driver and link it to the address 0x401200, you will also have to link the “.FLASHDRV” section in the Bootloader to the address 0x401200.

If you have to compile or relink the Flash driver, the hex-code may contain gaps. These gaps must be filled. Otherwise you’ll get problems downloading the Flash driver.

Info
TI provides the programming library necessary to build the flashdriver for ARM and TI compilers only. It can be built with those compilers only. If you only have access to the GHS compiler, please contact us if you need to rebuild the driver.

[3] Some OEMs don’t download the Flash driver into RAM but store them into Flash memory

3.2 [#hw_size] - Flash Segment Size

For more general information about this see the UserManual_FlashBootloader in the chapter **Flash Segment Size**

The Flash segment size is 16. This value denotes the minimum number of bytes, which can be written at once. The erase sector size depends on the Flash sector to be re-written and is typically between 8 KBytes and 64 KBytes – see [1] for details.
Please note that the section start address of the application to be downloaded always has to be aligned to 16.
Also, please note that only complete sectors of the Flash can be erased.

3.3 Timer

The FBL uses the RTI for internal timings. The RTI is initialized for automatic reload.

The hardware independent modules of the Bootloader require a 1ms time base to derive the necessary call cycles of cyclic functions from that time base.
For a flexible timer configuration, the timer reload value and the clock source must be configured in the “fbl_cfg.h” file (GENy). These two items are the initialization values of the corresponding timer registers.

Symbolic Constant	Description
FBL_TIMER_COUNTER_VALUE	Timer counter initialization value
FBL_TIMER_PRESCALER_VALUE	Timer prescaler Initialization value.

Info
FBL_TIMER_PRESCALER_VALUE and FBL_TIMER_COUNTER_VALUE are calculated by one of Vector’s generation tools (CANGen or GENy).

3.4 Watchdog Support

To service the internal or an optional external watchdog circuit, the Bootloader cyclically calls the application specific function **ApplFblWDTrigger** to trigger the watchdog.

Caution
The function ApplFblWDTrigger is copied into RAM and therefore needs to be relocatable. Also make sure, that no code in ROM is referenced by this function.

3.5 Startup Code

3.5.1 Special Startup Code Requirements

There are no special requirements of the Bootloaders startup code.

Caution
The delivered startup code is just an example startup code, which was used for testing purposes. This code has to be replaced/reviewed by hardware responsible of this project.

3.6 ARM And Thumb State

The ARM platform provides the so called Thumb instruction set for higher performance and higher code density. The FBL and the flash driver can be either built using ARM instructions or Thumb instructions. It is also possible to choose a different instruction set for the flash driver and the FBL, e.g. the FBL may use Thumb instructions while the flash driver uses ARM instructions. All components of the FBL should be built for the desired state except the startup code. The startup code of the FBL delivery cannot use Thumb instructions and must use ARM instructions anyway.

The instruction set that is used by the application can be chosen independently from the instruction set of the FBL.

4 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector-informatik.com