Debugging TMS470R1x Applications Using Internal RAM

Rainer Troppmann *Advanced Embedded Control*

ABSTRACT

This document describes how to debug TMS470R1x application software using the internal RAM. Examples are shown using the SE470R1VB8AD system emulation device (F05 technology). The second section gives some details on the memory setup and the memory allocation, and methods for debugging using internal RAM are described in the third section.

This document does not describe the bootloading of application software from the internal flash memory into the internal RAM.

Contents

1 Introduction

To allow a flexible design process and a more powerful debugging flow, you should run the application software out of the volatile RAM instead of the nonvolatile flash memory.

Doing so avoids the burden caused by multiple flash programming cycles and gives a better debugging environment. For devices having no access to external replacement RAM, you can use the internal RAM.

2 Memory Setup

In general, two Code Composer Studio™ (CCStudio) project files are involved in the memory setup used during a debugging session: a startup file and a GEL file.

To configure a TMS470 device, you must set up the memory map. This is normally done in the startup file (typically named startup.c). Figure 1 shows an example of a startup file (excerpt only) used for the SE470R1VB8AD system emulation device. This example is based on the structure `e_SARDEC_ST`, which overlays the register set of the system module. The names of the structure elements, such as `mfbahR0_UW`, are derived from the register names of the system module. The suffix refers to the data type of the structure element, e.g., UW means *unsigned word*.

2.1 Example Startup File

The following code snippet shows a startup file.

```
/* SE470R1VB8AD                  */

                                      /* up to 2MB internal Flash          */
e_SARDEC_ST.mfbahR0_UW = 0x0000;      /* address: 0x00000000               */
e_SARDEC_ST.mfbalR0_UW = 0x00B0;      /* size:     1 MB                    */
e_SARDEC_ST.mfbahR1_UW = 0x0010;      /* address: 0x00100000               */
e_SARDEC_ST.mfbalR1_UW = 0x00B0;      /* size:     1 MB                    */
                                      /* up to 128kB internal RAM          */
e_SARDEC_ST.mfbahR2_UW = 0x0020;      /* address: 0x00200000               */
e_SARDEC_ST.mfbalR2_UW = 0x0060;      /* size:    32 kB                    */
e_SARDEC_ST.mfbahR3_UW = 0x0020;      /* address: 0x00208000               */
e_SARDEC_ST.mfbalR3_UW = 0x8050;      /* size:    16 kB                    */

                                      /* up to 2kB internal HET RAM        */
e_SARDEC_ST.mfbahR4_UW = 0x0080;      /* address: 0x00800000               */
e_SARDEC_ST.mfbalR4_UW = 0x0010;      /* size:     1 kB                    */
e_SARMMC_ST.smcR1_UW   = 0x0072;      /* 7 wait states, 32 bit             */
                                      /* protected/user mode access        */
e_SARMMC_ST.pprot_UW   = 0x0000;      /* user access on all                */
                                      /* write control                     */
e_SARMMC_ST.wcr0_UN.wcr0_UW |= 0x0003; /* no implicit TWS, write buffer on */
                                      /* finally enable the CHIP SELECTS   */
e_SARDEC_ST.mfbalR0_UW |= 0x0100;
```

2.2 Example of Flash Bank Configuration Startup File

In the startup file, there is typically an additional configuration for the flash banks of the flash module. The following code snippet shows an example of the flash bank configuration using a structure where the pointer `FWPROGRAM` refers to the beginning of the register set. The names used for the structure elements are derived from the register names of the flash module. To save power, it is possible to power down unused flash banks. The flash performance can be influenced by the number of wait states `FLASHWS` programmed for each active flash bank.

```
/* --- ENABLE FLASH WRAPPER ACCESS --- */
e_SARSYS_ST.GlbCtrl_UW |= 0x0010;
/* --- setup Flash Banks --- */
FWPROGRAM->FMMAC2 = 0xFFF8;               /* bank0         */
FWPROGRAM->FMBAC1 = 0x00FF;               /* active        */
FWPROGRAM->FMBAC2 = 0x7F00 | FLASHWS;     /* wait states   */
FWPROGRAM->FMMAC2 = 0xFFF9;               /* bank1         */
FWPROGRAM->FMBAC1 = 0x00FC;               /* power down    */
FWPROGRAM->FMMAC2 = 0xFFFA;               /* bank2         */
FWPROGRAM->FMBAC1 = 0x00FC;               /* power down    */
FWPROGRAM->FMMAC2 = 0xFFFB;               /* bank3         */
FWPROGRAM->FMBAC1 = 0x00FC;               /* power down    */
FWPROGRAM->FMMAC2 = 0xFFFC;               /* bank4         */
FWPROGRAM->FMBAC1 = 0x00FF;               /* active        */
FWPROGRAM->FMBAC2 = 0x7F00 | FLASHWS;     /* wait states   */
FWPROGRAM->FMMAC2 = 0xFFFD;               /* bank5         */
FWPROGRAM->FMBAC1 = 0x00FF;               /* active        */
FWPROGRAM->FMBAC2 = 0x7F00 | FLASHWS;     /* wait states   */
FWPROGRAM->FMMAC2 = 0xFFFE;               /* bank6         */
FWPROGRAM->FMBAC1 = 0x00FF;               /* active        */
FWPROGRAM->FMBAC2 = 0x7F00 | FLASHWS;     /* wait states   */
FWPROGRAM->FMMAC2 = 0xFFFF;               /* bank7         */
FWPROGRAM->FMBAC1 = 0x00FF;               /* active        */
FWPROGRAM->FMBAC2 = 0x7F00 | FLASHWS;     /* wait states   */

/* --- ENABLE PIPELINE MODE --- */
FWPROGRAM->FMREGOPT = FLASHPM & 1;
/* --- DISABLE FLASH WRAPPER ACCESS --- */
e_SARSYS_ST.GlbCtrl_UW &= ~0x0010;
```

2.3 Additional Memory Map

To control the memory access of the CCStudio debugger tool, you will need to set up an additional memory map. This CCStudio tool−related memory setup can be done using a CCStudio menu (Option→Memory Map) or by providing the memory map information in a GEL file. GEL commands allow direct access to the memory and register content.

NOTE: Keep in mind that the access to some control registers, such as the registers for the memory configuration, can only be done in privilege mode. Normally, the mode is set to user mode at the end of the startup file.

A GEL file example (excerpt only), establishing the CCStudio tool−related memory mapping for a SE470R1VB8AD system emulation device, is shown in the following code snippet.

```
hotmenu Memory_Map()
{
```

```
GEL_MapOn();
GEL_MapReset();
//
// Memory Map setup
//
// SE3 blocks (address, page, length, readable, writeable)
// Memory
//----------------------------
/* Internal Flash (2MB): MFABxR0 and MFABxR1 register) */
GEL_MapAdd(0x00000000, 0, 0x00100000, 1, 0);      /* 1st Flash (1MB)            */
GEL_MapAdd(0x00100000, 0, 0x00100000, 1, 0);      /* 2nd Flash (1MB)            */
/* External RAM replacing internal Flash (2MB): MFABxR0 and MFABxR1 register)   */
//GEL_MapAdd(0x00000000, 0, 0x00100000, 1, 1);    /* 1st Flash (1MB)            */
//GEL_MapAdd(0x00100000, 0, 0x00100000, 1, 1);    /* 2nd Flash (1MB)            */
//----------------------------
                                                  /* Internal RAM (128kB): MFABxR2
                                                     and MFABxR3 register)      */
GEL_MapAdd(0x00200000, 0, 0x00010000, 1, 1);      /* 1st RAM (64kB, variable)   */
GEL_MapAdd(0x00210000, 0, 0x00010000, 1, 1);      /* 2nd RAM (64kB, variable)   */
                                                  /* Internal HET RAM (2kB):
                                                     MFABxR4 register           */
GEL_MapAdd(0x00400000, 0, 0x00000800, 1, 1);      /* HET RAM 2kbyte             */
//----------------------------
// Register
GEL_MapAdd(0xFFE84000, 0, 0x00000024, 1, 1);      /* MPU control registers      */
GEL_MapAdd(0xFFE88000, 0, 0x00008000, 1, 1);      /* Flash control registers    */
GEL_MapAdd(0xFFF00000, 0, 0x00080000, 1, 1);      /* Peripherals 512 kbyte      */
GEL_MapAdd(0xFFF80000, 0, 0x00080000, 1, 1);      /* SAR 512 kbyte              */
}
```

NOTE: Setting up overlapping memory will cause a reset of the device.

2.4 Memory Allocation: Linker Command File for Internal Flash Usage

A third CCStudio project file controls the memory allocation of the various sections of the program: the linker command file. The following code snippet shows an excerpt of a typical linker command file, where the program (program code, initialization tables, and constants) is allocated to the internal flash memory.

```
/*****************************************************************************/
/* SPECIFY THE SYSTEM MEMORY MAP                                             */
/*****************************************************************************/
MEMORY
{
   VECTORS   (X) : origin=0x00000000 length=0x24         /* vector table  */
```

```
   FLASH    (RX) : origin=0x00002000 length=0x001FE000   /* application   */
   STACKS   (RW) : origin=0x00200000 length=0x00001000   /* stacks        */
   RAM      (RW) : origin=0x00201000 length=0x0001F000   /* internal SRAM */
...
}
/*****************************************************************************/
/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY                               */
/*****************************************************************************/
SECTIONS
{
     .intvecs : {} > VECTORS      /* INTERRUPT VECTORS          */
     .stack   : {
                     _StackUSER_   = .+ (0x1000 - (4+128+4+4+128));
                     _StackFIQ_    = _StackUSER_   +   4;
                     _StackIRQ_    = _StackFIQ_ + 128;
                     _StackABORT_  = _StackIRQ_    +   4;
                     _StackUND_    = _StackABORT_  +   4;
                     _StackSUPER_  = _StackUND_    + 128;
         } > STACKS     /* SOFTWARE SYSTEM STACK      */
     .text    : {} > FLASH        /* CODE                              */
     .cinit   : {} > FLASH        /* INITIALIZATION TABLES             */
     .const   : {} > FLASH        /* CONSTANT DATA                     */
     .bss     : {} > RAM          /* GLOBAL & STATIC VARS              */
     .sysmem  : {} > RAM          /* DYNAMIC MEMORY ALLOCATION AREA    */
...
}
```

3 Debugging Methods

Before sections of the program (like the sections of the application software) can be loaded to the memory blocks specified in the linker command file, you must establish the memory setup for the CCStudio tool and the TMS470 device first.

NOTE: Resetting the CPU will make the memory setup of the TMS470 device invalid.

You can use any one of three debugging methods using internal RAM.

3.1 GEL File−Based Memory Setup

The GEL file−based debugging method uses a flow similar to the debugging flow for the SE470R1VB8AD system emulation device with external RAM. In this method, the total memory setup is done with a GEL file (see example of a flash configuration file in 2.2) instead of coding the memory setup in the startup file. To avoid memory overlapping, you must remove the memory setup from the startup file, e.g., by using a #if *notIntRAM* ... #endif around the setup sequence, where *notIntRAM* is not defined or equals zero. The following code snippet shows an example of how the memory setup for a TMS470 device is done in the GEL file.

```
hotmenu Address_Decoder()
{
//
// Address Decoder setup
//

    // internal Flash
    // block 1
    // MFABHR0,MFABLR0
    GEL_MemoryFill(0xfffffe00, 0, 0x1, 0x0000);
    GEL_MemoryFill(0xfffffe04, 0, 0x1, 0x00B0);
    // block 2
    // MFABHR1,MFABLR1
    GEL_MemoryFill(0xfffffe08, 0, 0x1, 0x0010);
    GEL_MemoryFill(0xfffffe0C, 0, 0x1, 0x00B0);
    // internal RAM
    // block 1
    // MFABHR2,MFABLR2
    //       base = 0x00200000
    //       size = 64kB
    GEL_MemoryFill(0xfffffe10, 0, 0x1, 0x0020);
    GEL_MemoryFill(0xfffffe14, 0, 0x1, 0x0070);
    // block 2
    // MFABHR3,MFABLR3
    //       base = 0x00210000
    //       size = 64kB
    GEL_MemoryFill(0xfffffe18, 0, 0x1, 0x0021);
    GEL_MemoryFill(0xfffffe1C, 0, 0x1, 0x0070);
    // internal HET RAM
    // MFABHR4,MFABLR4
    //       base = 0x00400000
    //       size = 2kB
    GEL_MemoryFill(0xfffffe20, 0, 0x1, 0x0040);
    GEL_MemoryFill(0xfffffe24, 0, 0x1, 0x0020);
    //       WS   = 7
    GEL_MemoryFill(0xfffffd04, 0, 0x1, 0x0072);
    //
    // Memory setup Enable
    //

    // read-modify-write MFBALR0.8 (MS bit)
    { int mfbalr0;
      mfbalr0 = *(int*)0xfffffe04;
```

```
      mfbalr0 |= 0x00000100;            /* enable Memory setup */
      *(int*)0xfffffe04 = mfbalr0;
   }
}
...
hotmenu Setup_Flash()
{
   SetupFlash();
}
   SetupFlash()
   {
      EnableFlashConfig();          /* enable Flash wrapper access   */
      {
      ...
      // set Flash WaitStates
      int fl_ws;
      int fmmac2;
      int fmbac1;
      int fmbac2;
      fl_ws  = 0x11;
      fmmac2 = 0xFFF8;              /* bank0                         */
      *(int*)0xFFE8BC04 = fmmac2;
      fmbac1 = 0x00FF;              /* active                        */
      *(int*)0xFFE88000 = fmbac1;
      fmbac2 = 0x7F00 | fl_ws;
      *(int*)0xFFE88004 = fmbac2;   /* wait states                   */
      fmmac2 = 0xFFF9;              /* bank1                         */
      *(int*)0xFFE8BC04 = fmmac2;
      fmbac1 = 0x00FC;              /* powerdown                     */
      *(int*)0xFFE88000 = fmbac1;
      fmmac2 = 0xFFFA;              /* bank2                         */
      *(int*)0xFFE8BC04 = fmmac2;
      fmbac1 = 0x00FC;              /* powerdown                     */
      *(int*)0xFFE88000 = fmbac1;
      fmmac2 = 0xFFFB;              /* bank3                         */
      *(int*)0xFFE8BC04 = fmmac2;
      fmbac1 = 0x00FC;              /* powerdown                     */
      *(int*)0xFFE88000 = fmbac1;
      fmmac2 = 0xFFFC;              /* bank4                         */
      *(int*)0xFFE8BC04 = fmmac2;
      fmbac1 = 0x00FF;              /* active                        */
      *(int*)0xFFE88000 = fmbac1;
      fmbac2 = 0x7F00 | fl_ws;
```

```
        *(int*)0xFFE88004 = fmbac2;      /* wait states                    */
        fmmac2 = 0xFFFD;                 /* bank5                          */
        *(int*)0xFFE8BC04 = fmmac2;
        fmbac1 = 0x00FF;                 /* active                         */
        *(int*)0xFFE88000 = fmbac1;
        fmbac2 = 0x7F00 | fl_ws;
        *(int*)0xFFE88004 = fmbac2;      /* wait states                    */
        fmmac2 = 0xFFFE;                 /* bank6                          */
        *(int*)0xFFE8BC04 = fmmac2;
        fmbac1 = 0x00FF;                 /* active                         */
        *(int*)0xFFE88000 = fmbac1;
        fmbac2 = 0x7F00 | fl_ws;
        *(int*)0xFFE88004 = fmbac2;      /* wait states                    */
        fmmac2 = 0xFFFF;                 /* bank7                          */
        *(int*)0xFFE8BC04 = fmmac2;
        fmbac1 = 0x00FF;                 /* active                         */
        *(int*)0xFFE88000 = fmbac1;
        fmbac2 = 0x7F00 | fl_ws;
        *(int*)0xFFE88004 = fmbac2;      /* wait states                    */
        }
        DisableFlashConfig();            /* disable Flash wrapper access   */
    }
```

3.1.1 Steps to Debugging Using GEL−File Based Setup

After opening CCStudio, the GEL file with the memory setup for the CCStudio tool and the TMS470 device must be loaded by selecting File −> Load GEL, if it is not already loaded by default as specified in the CCStudio setup tool before the application software is loaded for debugging.

The only modification to be made in the linker command file, when switching from internal flash to internal RAM, is to redirect the .text, .cinit, and .const sections from internal flash memory to internal RAM memory. Assuming that the memory map stays the same as in the linker command file shown in section 2.4, the required modifications are shown in the following code snippet.

```
/*****************************************************************************/
/* SPECIFY THE SYSTEM MEMORY MAP                                             */
/*****************************************************************************/
MEMORY
{
   VECTORS   (X) : origin=0x00000000 length=0x24         /* vector table     */
   FLASH    (RX) : origin=0x00002000 length=0x001FE000   /* application      */
   STACKS   (RW) : origin=0x00200000 length=0x00001000   /* stacks           */
   RAM      (RW) : origin=0x00201000 length=0x0001F000   /* internal SRAM    */
...
}
```

```
/*****************************************************************************/
/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY                               */
/*****************************************************************************/
SECTIONS
{
     .intvecs : {} > VECTORS         /* INTERRUPT VECTORS                */
     .stack   : {
                    _StackUSER_   = .+ (0x1000 - (4+128+4+4+128));
                    _StackFIQ_    = _StackUSER_   +   4;
                    _StackIRQ_    = _StackFIQ_ + 128;
                    _StackABORT_  = _StackIRQ_    +   4;
                    _StackUND_    = _StackABORT_  +   4;
                    _StackSUPER_  = _StackUND_    + 128;
            } > STACKS               /* SOFTWARE SYSTEM STACK            */
     .text    : {} > RAM             /* CODE                             */
     .cinit   : {} > RAM             /* INITIALIZATION TABLES            */
     .const   : {} > RAM             /* CONSTANT DATA                    */
     .bss     : {} > RAM             /* GLOBAL & STATIC VARS             */
     .sysmem  : {} > RAM             /* DYNAMIC MEMORY ALLOCATION AREA   */
...
}
```

NOTE: When moving to internal RAM, the internal RAM is much smaller than the internal flash memory. Therefore, you should check if the application software fits into the RAM without being jeopardized by the stacks.

3.2 Flash−Based Memory Setup

This method requires a preparation step once before the beginning of the debugging session. Program a base device setup, using a startup file and a simple main routine that consists of a forever loop only into the flash memory. Allocate the appropriate memory to the flash memory in a linker command file like the one shown in the following code snippet.

A typical debugging session for this method runs as follows:

1. After opening CCStudio, the GEL file with the memory setup for the CCStudio tool and the TMS470 device must be loaded by selecting File→Load GEL, if it is not already loaded by default as specified in the CCStudio setup tool before the application software is loaded for debugging.
2. Reset the CPU by selecting the Debug→Reset CPU item.
3. Run the base setup implemented in the flash memory by selecting the Debug→Go Main item or the Debug→Run item followed by Debug→Halt.
4. Load the application project and the application software into the internal RAM. Here the linker command file must allocate the application software into the RAM as shown in the code snippet in section 3.1.
5. Run the debugging session for the application software, but keep in mind that after a CPU reset, the base setup implemented in the flash memory has to be rerun as shown in step 2, before the application software is reloaded.

3.3 Flash−Based Software

Another method to debug is as follows:

1. Implement the whole program in the internal flash memory.
2. Load the application software from the internal flash memory into the internal RAM by using a bootloader implemented in the main routine.

This option requires you to reprogram the flash every time the software is modified, but it also gives you a more powerful debugging tool, with use of software breakpoints, etc., and extra performance for the memory access when using the bootloading concept for the normal application.

4 Summary

Three different methods have been provided to support different debugging purposes. When moving the application software into the internal RAM, the only mandatory modification to be made is the adaptation of the target memory in the linker command. Compared to the few restrictions (like the smaller memory), there are a lot of debugging advantages (more useable breakpoints, etc.) when moving the application software into the internal RAM instead of the internal flash.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265