Application Report
SPNA093A – February 2006 – Revised December 2007

In-System Programming With Catalog TMS470 Devices

John Mangino
TMS470 Applications

ABSTRACT

This document gives two examples of reprogramming the flash memory in-system on the TMS470 devices. These examples are intended as a reference to enable users to create their own in-system programming (ISP) methods. The reference design includes TMS470 software for use with the IAR Embedded Workbench™ tool.

Note: The examples in this application report require the Flash API Modules (SPRC236) within the "Tools & Software" folder.

Contents

List of Figures

1 Introduction

This application report introduces an understanding of in-system programming (ISP) of the TMS470 flash-memory devices. These examples use the IAR Embedded Workbench tool and have information for code generation specific to the IAR tools. The examples use compiled TI flash-memory API modules that are tested for flash-memory manipulation. It is recommended to use the IAR-compiled versions with the IAR Embedded Workbench tool. When using a different compiler, the sequence of the API modules must be verified. If a compiler optimizes an operation, it might not correctly perform the flash-memory operation (compact, erase, or program).

Note: Do not overwrite the flash-memory protection keys and the memory security module (MSM) keys. If the keys are rewritten and the data is not known, the part cannot be reprogrammed or accessed in the case of the MSM.

1.1 Flash-Memory Programming Overview

The TMS470 devices require a specific sequence to correctly erase and program the flash memory. Incorrectly programming the flash memory may result in unreliable operation or, in the worst case, cause flash-memory cell depletion. Texas Instruments provides the F05 TI flash-memory API modules that follow the correct sequence and are tested and proven to be the correct method. The flash-memory API modules simplify flash-memory operations and ensure correct operation.

The TI F05 flash-memory API routines are a library of routines that, when called with the proper parameters in the proper sequence, erase, program, or verify flash memory on the TMS470 family of Texas Instruments microcontrollers. These routines must be run in a privileged mode (mode other than user) to allow access to the flash-memory control registers and to the interrupt disable bits. Most of the routines enter flash-memory configuration mode and, therefore, the system clock should not exceed 24 MHz. The flash-memory API routines are described in the *TMS470 Family – F05 Flash Module Software Peripheral Driver User's Specification* (SPNU257).

The compiled routines were verified on the IAR 4.30a ARM C compiler.

1.1.1 Flash-Memory Erase Flow

Figure 1 describes the flow for erasing an arbitrary number of sectors on a device using the Flash_Erase_B function. This flow is desirable from a throughput standpoint, because sectors that already read as blank are processed much faster. The hallmark of the flow is the ability to disable preconditioning in Flash_Erase_B using the first 32-bit value in the **FLASH_STATUS_ST status** structure as a *key* when erasing sectors read as blank by Flash_Blank_B. Disabling preconditioning significantly speeds up erase of blank sectors.

It is not advisable to skip erase altogether on sectors that read as blank, because these sectors may require repair to marginally erased bits or depleted columns, and the repair is performed during execution of Flash_Erase_B. Also note that Flash_Erase_B is the only function that enables erase that allows for the collection of erase and compaction pulse counts.

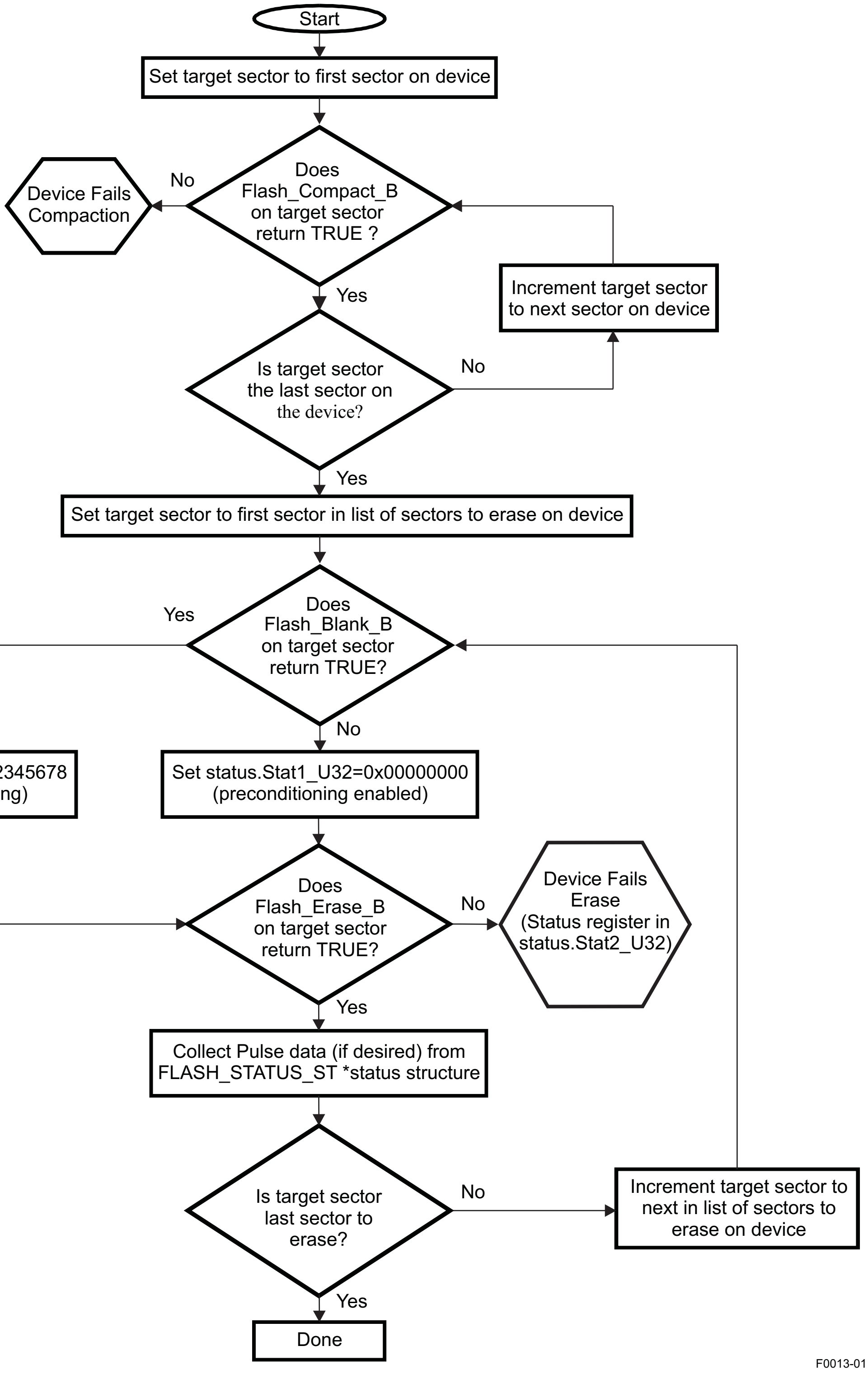
Start
Set target sector to first sector on device
Does Flash_Compact_B on target sector return TRUE ?
No
Device Fails Compaction
Yes
Is target sector the last sector on the device?
No
Increment target sector to next sector on device
Yes
Set target sector to first sector in list of sectors to erase on device
Does Flash_Blank_B on target sector return TRUE?
Yes
Set status.Stat1_U32=0x12345678 (disables preconditioning)
No
Set status.Stat1_U32=0x00000000 (preconditioning enabled)
Does Flash_Erase_B on target sector return TRUE?
No
Device Fails Erase (Status register in status.Stat2_U32)
Yes
Collect Pulse data (if desired) from FLASH_STATUS_ST *status structure
Is target sector last sector to erase?
No
Increment target sector to next in list of sectors to erase on device
Yes
Done
F0013-01

Figure 1. Flash-Memory Erase Flow Chart

1.1.2 Flash-Memory Programming Flow

Figure 2 describes the flow for programming the sectors on a device. When programming the flash memory, first erase all affected sectors using the previously described erase flow, managing the data buffers being programmed to flash memory such that they do not cross boundaries between flash-memory banks.

For example, for 1 KB of data to write, starting at the last 768 bytes of bank-0 on a device with more than one bank, the data must be divided into a 768-byte chunk to be written to bank-0 with one call to Flash_Prog_B, and the remaining 256 bytes are to be written to bank-1 with a second call to Flash_Prog_B. Within the same bank, any amount of data may be programmed within the limits of the available data buffer.

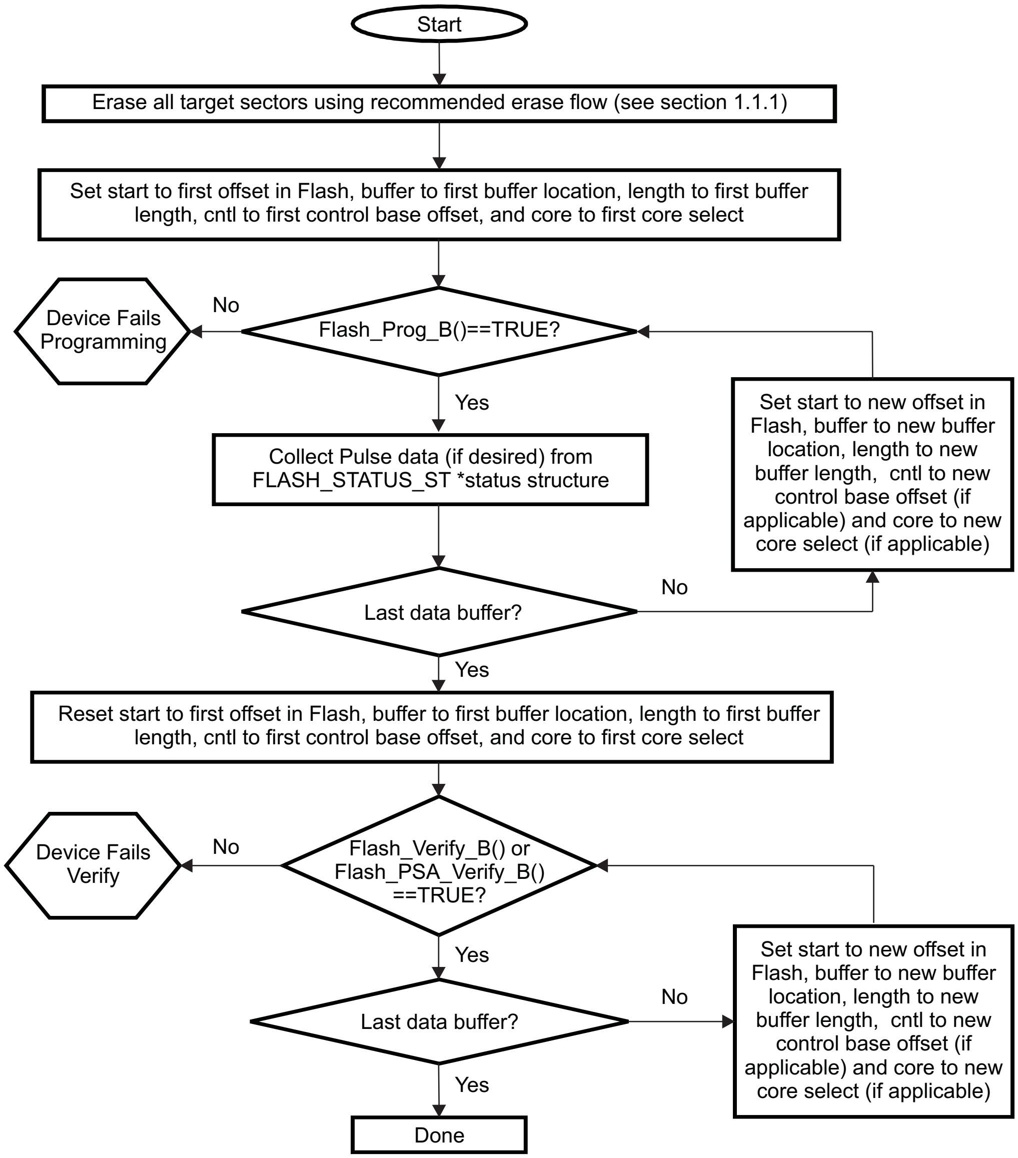
Start
Erase all target sectors using recommended erase flow (see section 1.1.1)
Set start to first offset in Flash, buffer to first buffer location, length to first buffer length, cntl to first control base offset, and core to first core select
Device Fails Programming
No
Flash_Prog_B()==TRUE?
Yes
Set start to new offset in Flash, buffer to new buffer location, length to new buffer length, cntl to new control base offset (if applicable) and core to new core select (if applicable)
Collect Pulse data (if desired) from FLASH_STATUS_ST *status structure
No
Last data buffer?
Yes
Reset start to first offset in Flash, buffer to first buffer location, length to first buffer length, cntl to first control base offset, and core to first core select
Device Fails Verify
No
Flash_Verify_B() or Flash_PSA_Verify_B() ==TRUE?
Yes
Set start to new offset in Flash, buffer to new buffer location, length to new buffer length, cntl to new control base offset (if applicable) and core to new core select (if applicable)
No
Last data buffer?
Yes
Done

Figure 2. Programming Flow Chart

1.2 Compiling New Source

The TMS470 devices offer flexibility of the memory map. Be careful to ensure the compiling and linking of new code functions properly with respect to the memory map and the existing code. The flash-memory API routines allow writing and erasing all memory locations within the flash memory. The MSM and flash-memory protection key areas are managed by the user to prevent accidental modification.

2 Programming Example Using the TI Flash-Memory APIs Loaded Into Flash Memory and Run From RAM

This example uses the flash-memory API modules. These modules are compiled into a user-defined segment, the API_SEGMENT. This segment allows the API routines to run from flash memory or RAM, depending on the application. The linker command file allocates the flash memory and RAM space. The main_B1M_program_flash_01.c program is an example for reference purposes that demonstrates the flash-memory APIs running out of RAM to reprogram the flash memory.

2.1 Overview – RAM to Flash Memory

This example is compiled on the IAR Embedded Workbench 4.30a tool. The JTAG debugger is used to load the program into flash memory. When executed, the main program copies the API routines from the flash-memory segment to the RAM segment and reprograms the fifth sector of flash memory with new values.

Using the modified tms470r1b1m_lnk_ram.xcl linker command file, the cstartup.r79 file and the tms470r1b1m_low_level_init_flash.c file, together these map the flash memory at 0x0000 0000, the RAM at 0x0040 0000, and the HET RAM at 0x0080 0000. The linker command file defines the API_SEGMENT as the RAM segment and the API_SEGMENT_D as the flash-memory segment.

Details on the TI flash-memory API routines are in the *TMS470R1x F05 Flash Reference Guide* (SPNU213). The API routines provide status results on the flash-memory programming. The data from the status registers within the API routines is transmitted via the UART2 on the B1M EVM (TMDS FET470R1B1M) to a terminal software such as the HyperTerminal™ program running on a PC for diagnostic purposes. These results are removed by commenting the TEST #define statement. The code is significantly smaller when used in the final version using only the necessary code to compact, erase, and program the flash memory.

This example compacts, erases, programs, verifies, calculates the parallel signature analysis (PSA), verifies the PSA, and gets the version number to demonstrate the use of the components of the API routines. The final version created by the user does not need all these steps for ISP.

Invoking the ISP calls the routines in RAM and reprograms the flash memory with new data or code. The primary functions of the ISP program are compacting, erasing, programing, and verifying the flash memory. Additionally, it demonstrates calculating the PSA, verifying the PSA and retrieving the flash-memory API version designator. Status is transmitted to the UART throughout the programming cycle.

2.2 Modifications to the Linker and ISP Module Files

The flash-memory API modules are compiled with the object files linked into the user-defined segment called API_SEGMENT. The linker file of the ISP modules requires the following declarations to create the API_SEGMENT in RAM and the API_SEGMENT_D in flash memory. The RAM segment is loaded or copied at run time from the flash-memory segment. The linker creates a link to the ISP files in RAM. The –Q linker designator automatically sets up the copy initialization of segments. This causes the linker to generate a new segment (initializer_segment) into which it places all data content of the segment. The application at runtime copies the contents of initializer_segment in flash memory to the segment in RAM.

2.2.1 Linker Modification Run From RAM

```
//************************************************
// FLASH API RAM and FLASH Segments
// Load from FLASH run in RAM
//************************************************

-Z(DATA)API_SEGMENT=RAMSTART-RAMEND
-Z(CONST)API_SEGMENT_D=ROMSTART-(KEYSTART-1),(KEYEND+1)-ROMEND
-QAPI_SEGMENT=API_SEGMENT_D
```

2.2.2 Main Routine Segment Initialization

```
void main(void)
{
  copy_to_ram(void)   // Copies the API modules from FLASH to RAM
}

//------------------------------------------------------------------------------
// This module copies the API modules from FLASH to RAM
//------------------------------------------------------------------------------
#pragma segment="API_SEGMENT"
#pragma segment="API_SEGMENT_D"
void copy_to_ram(void)
{
  int *i, *j;
  for(i = (int*)__segment_begin("API_SEGMENT"),
      j = (int*)__segment_begin("API_SEGMENT_D");
      i < (int*)__segment_end("API_SEGMENT");
    *i++ = *j++);
}
```

The user places ISP modules in RAM in a similar way. Placing the "#pragma location='API_SEGMENT'" statement before the user’s ISP routines causes the linker to make a copy of the ISP in flash memory and reserves the space in RAM for the code to be copied into at run time.

2.2.3 Implementing the #pragma location="API_SEGMENT"

```
#pragma location="API_SEGMENT"
void Your_ISP_Routine_01 (void)
{
  ...
  ...
  ...
}

#pragma location="API_SEGMENT"
void Your_ISP_Routine_02 (void)
{
  ...
  ...
  ...
}
```

2.3 Creating the Flash-Memory Programming Example

To create the program in the example, the following files are added to the project and compiled:

- Main program: main_B1M_program_flash_01.c
- Startup program: cstartup.s79
- Mapping file: tms470r1b1m_low_level_init_flash.c
- Flash-memory API files: blank.r79, compact.r79, erase.r79, fver.r79, feed_dog.r79, init_state.r79, match_key_B.r79, prog.r79, psa.r79, psa_calc.r79, sector_select.r79, track_pulses.r79, verify.r79, verify_psa.r79
- Linker command file: tms470r1b1m_lnk_flash_ram_to_flash.xcl

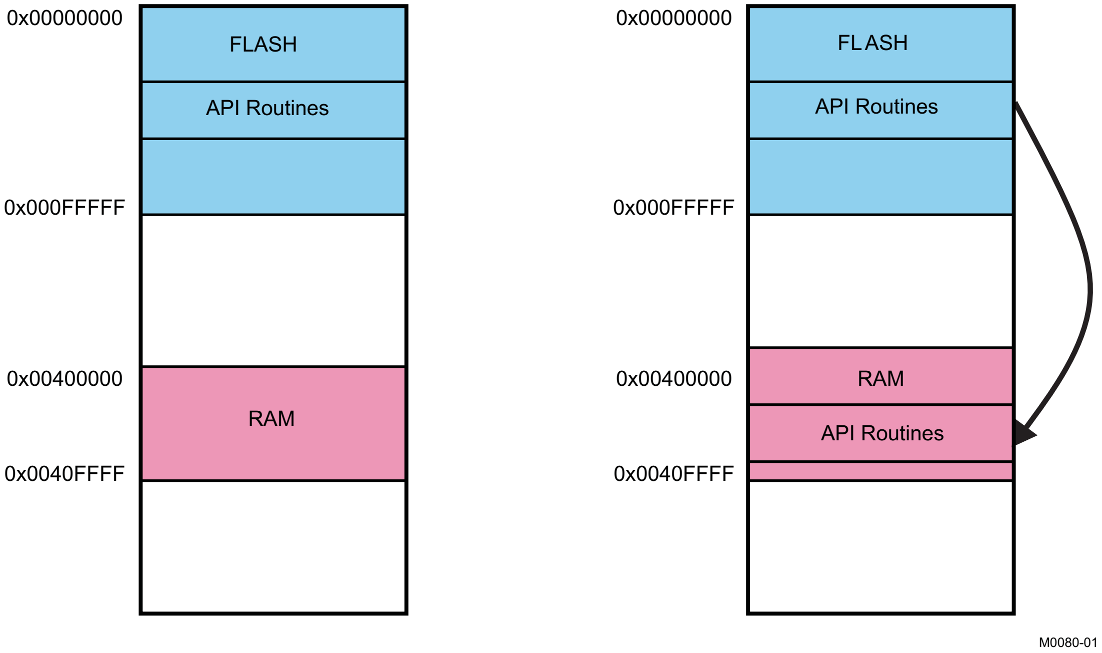

Figure 3. Diagram Showing the API Routines in Flash Memory and Copied to RAM

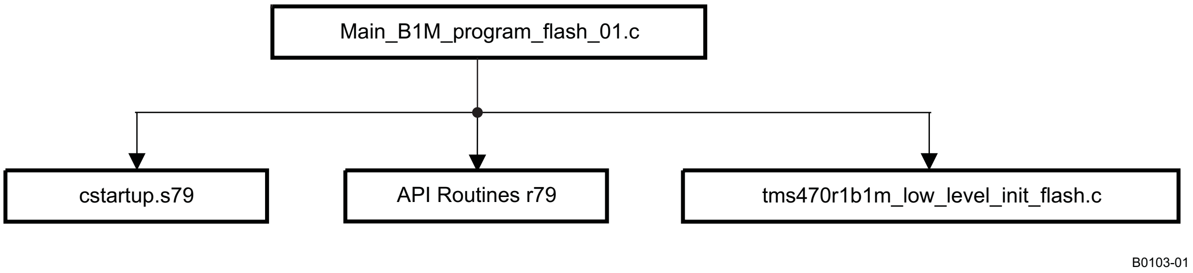
Main_B1M_program_flash_01.c
cstartup.s79
API Routines r79
tms470r1b1m_low_level_init_flash.c
B0103-01

Figure 4. Files for Example Program

3 Programming Example Using the TI Flash-Memory APIs Loaded and Run From Bank-0 and Reprogramming Bank-1

This example uses the flash-memory API modules. These modules are compiled into a user-defined segment, the API_SEGMENT. This segment allows the API routines to run from flash memory or RAM, depending on the application. The linker command file allocates the flash memory and RAM space. The main_B1M_program_flash_02.c program is an example for reference purposes that demonstrates the flash-memory APIs running out of flash-memory bank-0 to reprogram the flash memory in bank-1.

3.1 Overview – Bank-0 to Bank-1

This example is compiled on the IAR Embedded Workbench 4.30a tool. The JTAG debugger loads the program into flash memory. When run, the main program executes the flash-memory APIs running out of flash-memory bank-0 to reprogram the flash memory in bank-1 at address 0x0008 0000 with new values.

Using the modified tms470r1b1m_lnk_ram_B0_to_B1.xcl linker command file, the cstartup.r79 file, and the tms470r1b1m_low_level_init_flash.c file, together these map the flash memory at 0x0000 0000, the RAM at 0x0040 0000, and the HET RAM at 0x0080 0000. The linker command file defines the API_SEGMENT as the flash-memory segment.

Details on the TI flash-memory API routines are in the *TMS470R1x F05 Flash Reference Guide* (SPNU213). The API routines provide status results on the flash-memory programming. The data from the status registers within the API routines is transmitted via the UART2 on the B1M EVM (TMDS FET470R1B1M) to terminal software such as the HyperTerminal program running on a PC for diagnostic purposes. These results are removed by commenting the TEST #define statement. The code is significantly smaller when used in the final version, with only the code necessary to compact, erase, and program the flash memory.

This example compacts, erases, programs, verifies, calculates the Parallel Signature Analysis (PSA), verifies the PSA, and gets the version number to demonstrate the use of the components of the API routines. The final version created by the user does not need all these steps for ISP.

Invoking the ISP calls the routines in RAM and reprograms the flash memory with new data or code. The primary functions of the ISP program are compacting, erasing, programming, and verifying the flash memory. Additionally, the ISP calculates the PSA, verifies the PSA, and retrieves the flash-memory API version designator. Status is transmitted to the UART throughout the programming cycle.

3.2 Modifications to the Lnker and ISP Module Files

The flash-memory API modules are compiled with the object files linked into the user-defined segment called API_SEGMENT. The linker command file of the ISP modules requires the following declarations to create the API_SEGMENT in flash memory.

3.2.1 Linker Modification Run From Flash Memory

```
//*************************************************
// FLASH API RAM and FLASH Segments
// Run from FLASH Bank 0
// Program Bank 1
//*************************************************
-DBank0_START=0x00000000
-DBank0_END=0x0007FFFF
-DBank1_START=0x00080000
-DBank1_END=0x000FFFFF
-Z(DATA)API_SEGMENT=ROMSTART-(KEYSTART-1),(KEYEND+1)-Bank0_END
```

3.2.2 Implementing the #pragma location="API_SEGMENT"

The user places ISP modules in flash memory in a similar way. Placing the #pragma location="API_SEGMENT" statement before the user ISP routines causes the linker to put the routines in the specified location in flash memory. This is the case to place reprogramming algorithms in one bank while modifying the other flash-memory bank.

```
#pragma location="API_SEGMENT"
void Your_ISP_Routine_01 (void)
{
  ...
  ...
  ...
}

#pragma location="API_SEGMENT"
void Your_ISP_Routine_02 (void)
{
  ...
  ...
  ...
}
```

3.3 Creating the Flash-Memory Programming Example

To create the program in the example, the following files are added to the project and compiled:

- Main program: main_B1M_program_flash_02.c
- Startup program: cstartup.s79
- Mapping file: tms470r1b1m_low_level_init_flash.c
- Flash-memory API files: blank.r79, compact.r79, erase.r79, fver.r79, feed_dog.r79, init_state.r79, match_key_B.r79, prog.r79, psa.r79, psa_calc.r79, sector_select.r79, track_pulses.r79, verify.r79, verify_psa.r79
- Linker command file: tms470r1b1m_lnk_flash_b0_to_b1.xcl

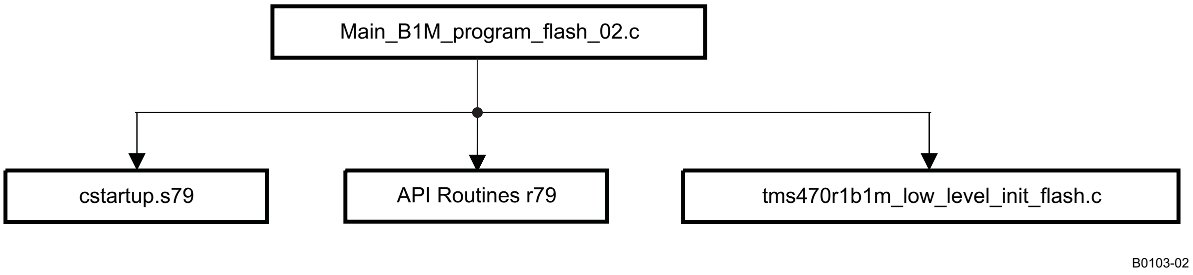
Main_B1M_program_flash_02.c
cstartup.s79
API Routines r79
tms470r1b1m_low_level_init_flash.c
B0103-02

Figure 5. Files for Example Program

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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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