Programming TMS570 Flash Using Flash API
AEC Automotive Safety Application

ABSTRACT
The Texas Instrument's TMS570 products have Flash memory that can store data or machine code for execution. This document provides a high-level overview of the TMS570 Flash architecture and Flash API, focusing on the erasing and programming of TMS570 Flash using Flash API.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/spna117.zip.

Contents
1 Introduction .................................................................................................................. 1
2 TMS570 Flash Overview ................................................................................................... 1
3 Programming Flash Using Flash API .................................................................................... 4
4 Example ...................................................................................................................... 7
5 References ................................................................................................................... 8
Appendix A Flash API Function Description ........................................................................... 9

List of Figures
1 Flash Module Block Diagram .............................................................................................. 2
2 ECC Word Memory Map ................................................................................................... 2
3 Flash Cell Structure .......................................................................................................... 3
4 Flash Cell 0/1 Threshold in Different Reads ........................................................................ 3

List of Tables
1 Source Files in Project ....................................................................................................... 7
2 Project Files ..................................................................................................................... 7
3 Flash API Function Description ........................................................................................... 9
4 Flash API Function Parameter Description ........................................................................ 10

1 Introduction
TMS570 platform devices are implemented with embedded Flash memories. Flash API defines a set of software peripheral functions intended to program/erase the Flash module. This application report explains how to call those functions to program/erase the Flash memories.

2 TMS570 Flash Overview
TMS570 has integrated Flash memory for nonvolatile data storage. Usually, the Flash memory consists of several Flash banks, which is a group of Flash sectors. The smallest unit that can be erased in a Flash memory is a complete Flash sector. For example, TMS570LS20216S has four Flash banks: bank0 has 10 sectors and the other three banks have 4 sectors in each bank. See the device-specific data sheet for bank/sector configuration.
The Flash bank in TMS570LS20216S is 144 bits wide, including 128 bits for normal data width and two sets of 8 bit ECC check bits. Each 8 ECC check bits check 64 data bits and 19 address bits. Since ECC in this device contains address information, the ECC bits will be different if the same data is stored in a different location. Logically, this 8-bit ECC repeats itself four times as shown in Figure 2. In other words, physically, Flash ECC occupies only 1/8 of the corresponding Flash data; logically, the size of Flash ECC is half of the corresponding Flash data.
Figure 3 shows a typical Flash cell and its symbol. TMS570 Flash has three basic operations:

- **Erase** – Remove electrons from the floating gate, generating logical 1s in the memory.
- **Program** – Forcing electrons into the floating gate, generating logical 0s in the memory.
- **Read** – Accessing the contents of a Flash memory. It can either be a 1 or 0. It is done by applying voltage on the control gate and sensing the drain current (IFlash). Normal read, erase verify, program verify, and read margin tests are all variations of read. They are differentiated by the control gate voltage and the threshold of the drain current to determine logical 0 and 1. The current/voltage ratio threshold of different reads are sequenced as shown in Figure 4, where depletion means over-erased, the transistor is always on:

![Flash Cell Structure](image-url)

**Figure 3. Flash Cell Structure**

![Flash Cell 0/1 Threshold in Different Reads](image-url)

**Figure 4. Flash Cell 0/1 Threshold in Different Reads**
3 Programming Flash Using Flash API

Flash API defines a set of software peripheral functions intended to program/erase the Flash module. There are more than 50 functions, including many TMS470R1x legacy and diagnostic functions. Table 3 and Table 4 in the Appendix lists the description for those nine functions used in this document. With these nine functions, you can perform most of the Flash operations.

There are several steps required to program Flash using Flash API functions:
1. Include the Flash API head files and libraries into the project.
2. Compact the target Flash.
3. Erase the target Flash.
4. Program the target Flash.
5. Verify the target Flash.

3.1 Include the Flash API Files

After the F035a Flash API is installed, the following files will be in the directory:

- Three header files: f035.h, flash470.h and Flash470ErrorDefines.h. Include these files in the project.
- Three .lib files: pf035a_api.lib, pf035a_api_tiabi.lib, pf035a_api_eabi.lib. Except for the names, the first two libraries are identical. Include either file in the project if it is built using tiabi options (in Code Composer Studio™ 3.3, Project → Build Options, under tab Compiler and Linker; in Code Composer Studio 4.x, Project → Properties, under tab Tool Settings, click Runtime Model Options). Otherwise, if eabi is used, include pf035a_api_eabi.lib in the project. These two ‘abi’s specify different standard conventions for file formats, data types, register usage, stack frame organization, and function parameter passing of an embedded software program.

The attached example uses eabi. After including all the head files and library files, the application can call the API function defined in flash470.h.

3.2 Compact

A device may contain depleted (over erased) columns and/or marginally erased bits. The application code should validate the target Flash before erasing. Otherwise, the leakage current caused by the depleted bits might confuse the sense amplifier in the other. The Flash API provides the following function to the compact depleted Flash sectors.

Flash_Compact_B()

This function validates the Flash data and ECC/parity together. For example, if this function is called to validate the Flash data of sector 0 in bank1, it validates both this sector and the corresponding ECC/parity area.

The attached example uses the following code to compact bank2 and bank3:

```c
//Compact bank 2 and bank3 the Flash
pstatus=0x0;
for(i=14;i<NUMBEROFSECTORS;i++)
{
    temp=Flash_Compact_B(sector[i].start, sector[i].bank,
    (FLASH_SECT)sector[i].sectorNumber, delay,
    (FLASH_ARRAY_ST)sector[i].FlashBaseAddress,&status);
    if(!temp) { pstatus=1;
        break; }
}
```
3.3 **Erase**

The target Flash is ready for erasing after it is validated by the compact function. After erasing, the target Flash reads as all '0xFFFFFFFF's. This state is called as *blank*. Flash API provides one bank erase and two sector functions to erase the target Flash. Similar to the compact function, all the erase function erases the data and the ECC/parity together.

`Flash_Erase_Bank_B(); // erase the target bank`

`Flash_Erase_B(); // erase the target sector`

`Flash_Erase_Sector_B(); // erase the target sector`

`Flash_Erase_Bank_B()` and `Flash_Erase_B()` support disabling preconditioning, i.e., program to 0’s prior to applying erase pulses. Obviously, disabling preconditioning can save erase time on the *blank* banks. However, the application makes sure that the target Flash is *blank* before disabling preconditioning. Flash API provides the following function to determine if the Flash bank is *blank* before disabling preconditioning:

`Flash_Bank_B()`

This function can also be used to verify the Flash has been properly erased.

The attached example calls `Flash_Erase_Bank_B()` to erase bank2 and `Flash_Erase_B()` to erase bank3.

```c
// Erase Antares Flash Bank 2 using Bank Erase
pstatus=0x0;
status.stat1 = 0;//enable preconditioning
temp=Flash_Erase_Bank_B(bank[2].start,bank[2].length,bank[2].bankNumber, delay, (FLASH_ARRAY_ST)bank[2].FlashBaseAddress,& status);
if(!temp) { pstatus=1;}
```

```c
// Erase Antares Flash bank 3 using Sector Erase
pstatus=0x0;
status.stat1 = 0; //enable preconditioning
for(i=18;i<NUMBEROFSECTORS;i++)
{
    temp=Flash_Erase_B(sector[i].start,sector[i].length,sector[i].bank, (FLASH_SECT)sector[i].sectorNumber, delay, (FLASH_ARRAY_ST)sector[i].FlashBaseAddress,& status);
    if(!temp) { pstatus=1;
        break;
    }
}
```

3.4 **Program**

Flash API provides the following function to program the Flash:

`Flash_Prog_B()`

This function programs the target Flash from the starting address *start*. The data buffers being programmed to Flash should not cross boundaries between banks that are 32-bit aligned. In the attached example, programming 1k byte data into Flash calls the `Flash_Prog_B()` twice because it crosses the boundary of bank2 and bank3.

Different from the compact and erase function, the program function programs the Flash data and ECC/parity separately. Generate the Flash ECC/parity part, i.e., through nowECC™ tool, to generate the ECC part from an input file and program the ECC/parity into the target Flash ECC/parity area.
The attached example uses following code to program 1k byte data into bank2 and bank3.

```c
// Program the 1k byte data from RAM 0x9000 to Flash 0x0017FF00.
pstatus=0x0;
temp=Flash_Prog_B((void *)0x0017FF00, (UINT32 *)&Flash_Data, 0x100>>2, FLASH_CORE2,
delay, (FLASH_ARRAY_ST)0xffff87000, &status);
if(!temp) {pstatus=1;}
temp=Flash_Prog_B((void *)0x00180000, (UINT32 *)((&Flash_Data)+0x40), 0x300>>2,
FLASH_CORE3, delay, (FLASH_ARRAY_ST)0xffff87000, &status);
if(!temp) {pstatus=1;}

// Program Flash ECC
pstatus=0x0;
temp=Flash_Prog_B((void *)0x004bff80, (UINT32 *)&Flash_ECCData, 0x80>>2, FLASH_CORE2,
delay, (FLASH_ARRAY_ST)0xffff87000, &status);
if(!temp) {pstatus=0x20;}
temp=Flash_Prog_B((void *)0x004C0000, (UINT32 *)((&Flash_ECCData)+0x20), 0x180>>2,
FLASH_CORE3, delay, (FLASH_ARRAY_ST)0xffff87000, &status);
if(!temp) {pstatus=0x20;}
```

### 3.5 Verify

After program, the application code should perform verify on all buffers using either Flash_Verify_B() or Flash_PSA_Verify_B(). These two functions verify proper programming by using normal read, read-margin 0, and read-margin 1.

The attached example use the following code to verify 1k byte data into bank2 and bank3.

```c
// Verify that the 1k byte data from RAM 0x9000 to Flash 0x0017FF00.
pstatus=0x0;
temp=Flash_Verify_B((void *)0x0017FF00, (UINT32 *)&Flash_Data, 0x100>>2, FLASH_CORE2,
(FLASH_ARRAY_ST)0xffff87000, &status);
if(!temp) {pstatus=0x1;}
temp=Flash_Verify_B((void *)0x00180000, (UINT32 *)((&Flash_Data)+0x40), 0x300>>2,
FLASH_CORE3, (FLASH_ARRAY_ST)0xffff87000, &status);
if(!temp) {pstatus=0x1;}

// Verify Flash ECC
pstatus=0x0;
temp=Flash_Verify_B((void *)0x004bff80, (UINT32 *)&Flash_ECCData, 0x80>>2, FLASH_CORE2,
(FLASH_ARRAY_ST)0xffff87000, &status);
if(!temp) {pstatus=0x40;}
temp=Flash_Verify_B((void *)0x004C0000, (UINT32 *)((&Flash_ECCData)+0x20),
0x180>>2, FLASH_CORE3, (FLASH_ARRAY_ST)0xffff87000, &status);
if(!temp) {pstatus=0x40;}
```

### 3.6 Notes

To avoid ECC/parity malfunctions, TI strongly recommends that you disable Flash ECC/parity during Flash compacting, erasing, programming and verifying. TI also recommends that you disable all the interrupts and exceptions during these operations because the Flash API might miss counting compact/erase pulses if the CPU is serving interrupt.

F035 Flash offers the possibility of erasing/programming one bank while executing code in another bank. No attempts should be made to read from Flash locations in the same bank as the area being operated on by the Flash State Machine until the command completes. For example, appropriate code in bank0 sector 0 can erase bank2 but can NOT erase bank0 sector 3.
3.7 **One Time Programmable (OTP) Flash Sector**

There are four customer OTP sectors mapping to the 6Mbyte offset from the Flash-based address; they increment by 2k for each. These sectors can only be programmed once. Do not attempt to erase the OTP sectors. Flash API provides the following function to program the customer OTP area.

```c
OTP_Prog_B(); // Program customer OTP
```

You can perform the blank check verify using the same function as normal Flash, i.e., `Flash_Blank_B()` for blank check, `Flash_PSA_Verify_B()` and `Flash_Verify_B()` for verify.

4 **Example**

The attached example is built with *eabi*; communicate with PC through SCI using RS232 protocols. You can type the command in Hyper Terminal to execute compact, erase, blank check, program and verify Flash API function.

It includes following source files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intvecs.asm</td>
<td>Interrupt vectors setup file</td>
</tr>
<tr>
<td>Boot.asm</td>
<td>Initialize stack, call _init and _main</td>
</tr>
<tr>
<td>Startup.c</td>
<td>Initialize global variables</td>
</tr>
<tr>
<td>Data_To_Flash.asm</td>
<td>1k byte Data to be programmed into 0x17FF00</td>
</tr>
<tr>
<td>Data_To_FlashECC.asm</td>
<td>0.5k byte ECCData to be programmed into 0x004BFF80</td>
</tr>
<tr>
<td>SCI.C</td>
<td>RS232 functions</td>
</tr>
<tr>
<td>Main.c</td>
<td>Demonstrates the Flash compact, erase, blank check, program and verify Flash API function</td>
</tr>
</tbody>
</table>

It includes two project files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlashEABI_Flash.pjt</td>
<td>Build FlashEABI_Flash.out using linker_flash.cmd as the link file. User shall program this .out file into TMS570LS20216S Flash to run the demo.</td>
</tr>
<tr>
<td>FlashEABI_RAM.pjt</td>
<td>Build FlashEABI_RAM.out using linker_RAM.cmd as the link file. User shall load this .out file into TMS570LS20216S SRAM to run the demo.</td>
</tr>
</tbody>
</table>

4.1 **Running the Demo**

4.1.1 **Hardware**

1. Connect the TI EVM board USB port to a PC.
2. Configure the right COM port and the baud rate to be 19.2 kHz, no parity, one stop bit.

4.1.2 **Software**

1. Program `FlashEABI_Flash.out` to the Flash device using the nowFlash tool; reset the device. Or,
2. Load `FlashEABI_RAM.out` into the RAM device using Code Composer Studio; click *run*.

The following menu prompts are shown from HyperTerminal:

- 0 - Call `Flash_Compact_B()` to compact bank2 and bank3
- 1 - Call `Flash_Erase_Bank_B()` to erase bank2
- 2 - Call `Flash_Erase_B()` to erase bank3
- 3 - Call `Flash_Blank_B()` to blank check bank2 and bank3
- 4 - Call `Flash_Prog_B()` to program 1k byte data to Flash 0x0017FF00
- 5 - Call `Flash_Verify_B()` to program 1k byte data to Flash
- 6 - Call `Flash_Prog_B()` to program 0.5k byte data to FlashECC 0x004BFF80
- 7 - Call `Flash_Verify_B()` to program 0.5k byte data to FlashECC
3. Input the function index in HyperTerminal to run the function.
   For example, you can input the command 1, 2, 3, 4, 5, 6, 7 sequentially to program the 1kbyte data and the according ECC bits to Flash. Function 8 and 9 can only be correctly executed if the code is in RAM. If it runs from Flash, it will cause an exception and require a reset because the code tries to erase the Flash sector in the same bank.

5 References

- TMS470/570 Platform F035a Flash Application Programming Interface User’s Specification, v1.03 (SPNU493)
- TMS570LS20216 Technical Reference Manual (SPNU489)
- CoreSight and Trace for Cortex-R Series Processors http://www.arm.com/products/system-ip/debug-trace/coresight-for-cortex-r.php
## Appendix A Flash API Function Description

### Table 3. Flash API Function Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Flash_Compact_B()</code></td>
<td>This function adjusts depleted (over-erased) Flash memory bits in the target sector so they are not in depletion. This function only performs compaction on the target sector, so compaction of N sectors requires N calls.</td>
</tr>
<tr>
<td><code>Flash_Erase_B()</code></td>
<td>This function is used to erase the targeted sector and to collect pulse count information. It allows for the disabling of reconditioning during erase.</td>
</tr>
<tr>
<td><code>Flash_Erase_Sector_B()</code></td>
<td>This function erases a sector, and preconditioning is enabled by default and cannot be disabled as is possible using <code>Flash_Erase_B</code> or <code>Flash_Erase_Bank_B</code>.</td>
</tr>
<tr>
<td><code>Flash_Erase_Bank_B()</code></td>
<td>This function is used to erase the targeted bank and to collect pulse count information. It allows for the disabling of reconditioning during erase.</td>
</tr>
<tr>
<td><code>Flash_Blank_B()</code></td>
<td>This function verifies that the Flash has been properly erased starting from the address passed in the parameter <code>start</code> <code>length</code> words are read, starting at the starting address.</td>
</tr>
<tr>
<td><code>Flash_Prog_B()</code></td>
<td>This function programs the Flash from the starting address <code>start</code> for <code>length</code> 32 bit words. The user code must make sure that the areas to be programmed are already erased before calling this routine.</td>
</tr>
<tr>
<td><code>Flash_Verify_B()</code></td>
<td>This function verifies proper programming by using normal read, read-margin 0, and read-margin 1 modes. Verification starts from the start address <code>start</code> and check <code>length</code> words from the start address.</td>
</tr>
<tr>
<td><code>Flash_PSA_Verify_B()</code></td>
<td>This function verifies proper programming by using normal read, read-margin 0, and read-margin 1 modes, and generating a 32 bit PSA checksum for the data in the region in each mode. Verification starts from the start address <code>start</code> and check <code>length</code> words from the start address.</td>
</tr>
</tbody>
</table>
Table 3. Flash API Function Description (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTP_Prog_B()</td>
<td>Similar to Flash_Prog_B, except that it is used to program OTP.</td>
</tr>
</tbody>
</table>

```c
BOOL OTP_Prog_B (UINT32 *pu32Start,
 UINT32 *pu32Buffer, UINT32
 u32Length, FLASH_CORE oFlashCore,
 UINT32 u32Delay, FLASH_ARRAY_ST
 oFlashControl, FLASH_STATUS_ST
 *poFlashStatus, );
```

Table 4. Flash API Function Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pu32Start / start</td>
<td>UINT32</td>
<td>Points to the first word in the Flash that are compact/erased/programmed or verified</td>
</tr>
<tr>
<td>pu32Buffer / buff</td>
<td>UINT32</td>
<td>Pointer to the starting address of a buffer with data to program or verified</td>
</tr>
<tr>
<td>u32Length / length</td>
<td>UINT32</td>
<td>Number of 32-bit words to program/verify. This parameter in erase function can be ignored.</td>
</tr>
<tr>
<td>oFlashCore / core</td>
<td>FLASH_CORE</td>
<td>Bank select (0-7) of region being programmed/verified.</td>
</tr>
<tr>
<td>u32Delay</td>
<td>UINT32</td>
<td>Flash delay parameter determines the compact/erase/program pulse width. Should be half of the HCLK frequency (in MHz). For example, if HCLK is 80 MHz, this parameter should be set to (ceiling)(80/2) = 40.</td>
</tr>
<tr>
<td>oFlashControl / ctnl</td>
<td>FLASH_ARRAY_ST</td>
<td>Flash control base address of module. 0xff87000 for TMS570LS20216S.</td>
</tr>
<tr>
<td>poFlashStatus / status</td>
<td>FLASH_STATUS_ST</td>
<td>Pointer to status structure for storing statistical information.</td>
</tr>
<tr>
<td>UINT32</td>
<td>psa</td>
<td>The expected PSA value against which the actual PSA values is compared.</td>
</tr>
</tbody>
</table>
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 neither designed nor intended 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:

<table>
<thead>
<tr>
<th>Products</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplifiers</td>
<td>Audio</td>
</tr>
<tr>
<td>Data Converters</td>
<td>Automotive</td>
</tr>
<tr>
<td>DLP® Products</td>
<td>Communications and Telecom</td>
</tr>
<tr>
<td>DSP</td>
<td>Computers and Peripherals</td>
</tr>
<tr>
<td>Clocks and Timers</td>
<td>Consumer Electronics</td>
</tr>
<tr>
<td>Interface</td>
<td>Energy</td>
</tr>
<tr>
<td>Logic</td>
<td>Industrial</td>
</tr>
<tr>
<td>Power Mgmt</td>
<td>Medical</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>Security</td>
</tr>
<tr>
<td>RFID</td>
<td>Space, Avionics &amp; Defense</td>
</tr>
<tr>
<td>RF/IF and ZigBee® Solutions</td>
<td>Wireless</td>
</tr>
</tbody>
</table>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2010, Texas Instruments Incorporated