Application Report
SPNA117A–May 2010

Programming TMS570LS20x/10x 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.ti.com/lit/zip/SPNA117.

Contents

List of Figures

List of Tables

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 TMS570LS20x/10x Flash Overview

TMS570LS20x/10 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.

Code Composer Studio, nowECC are trademarks of Texas Instruments.

Copyright © 2010, Texas Instruments Incorporated

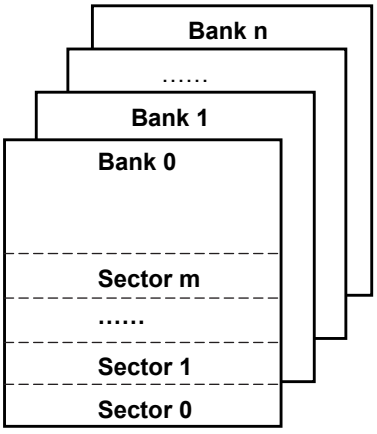
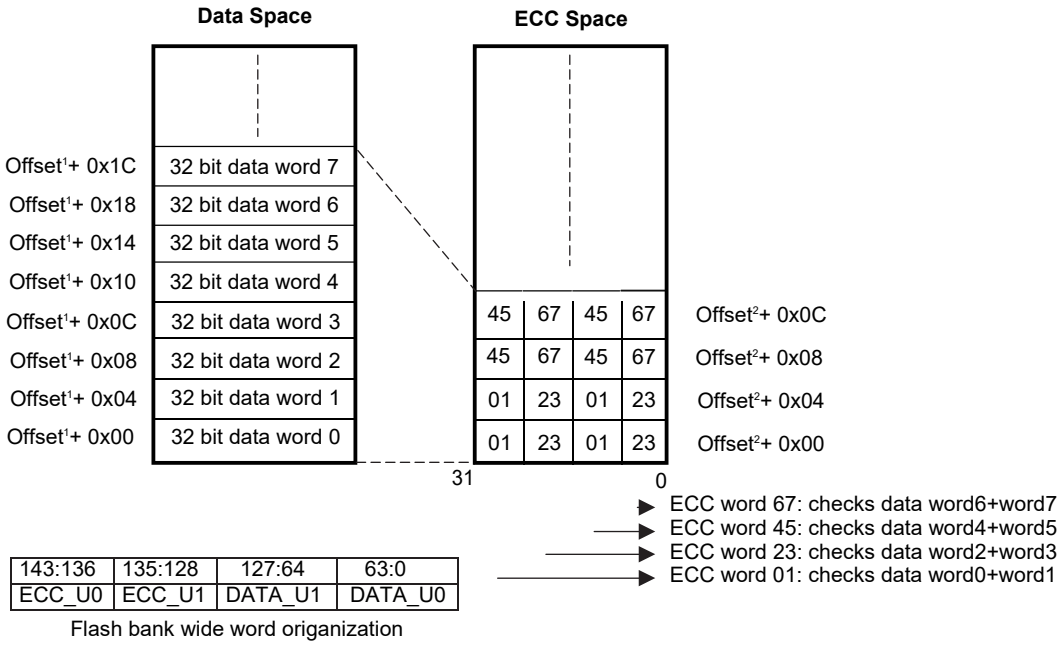


Figure 1. Flash Module Block Diagram

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.



Notes: 1) Offset is 0M for main data space, 6M for customer OTP space.
2) Offset is 4M for main data ECC space, 6M+32k for customer OTP ECC space.

Figure 2. ECC Word Memory Map

Copyright © 2010, Texas Instruments Incorporated

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:

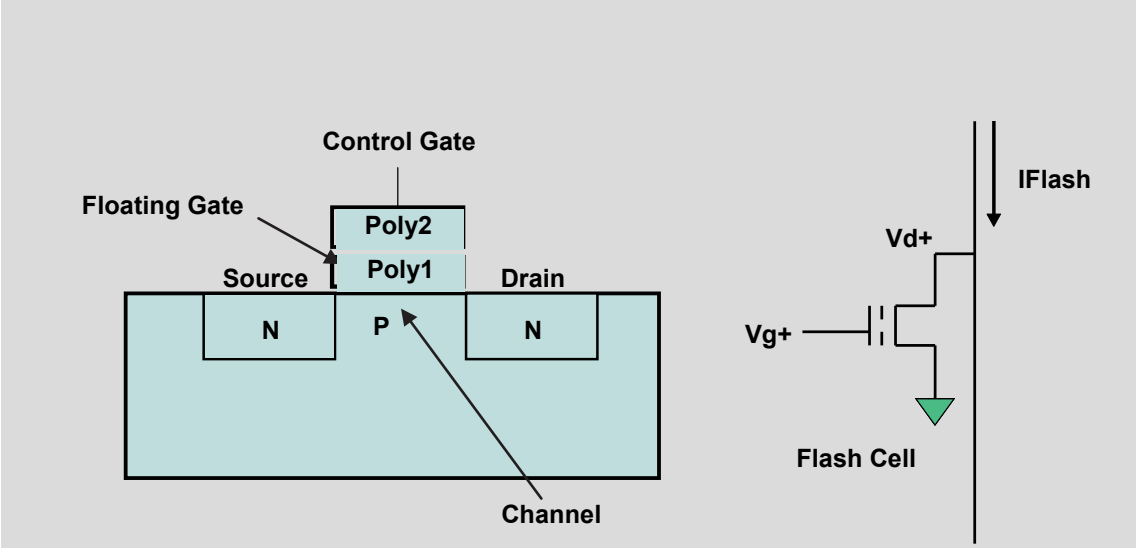


Figure 3. Flash Cell Structure

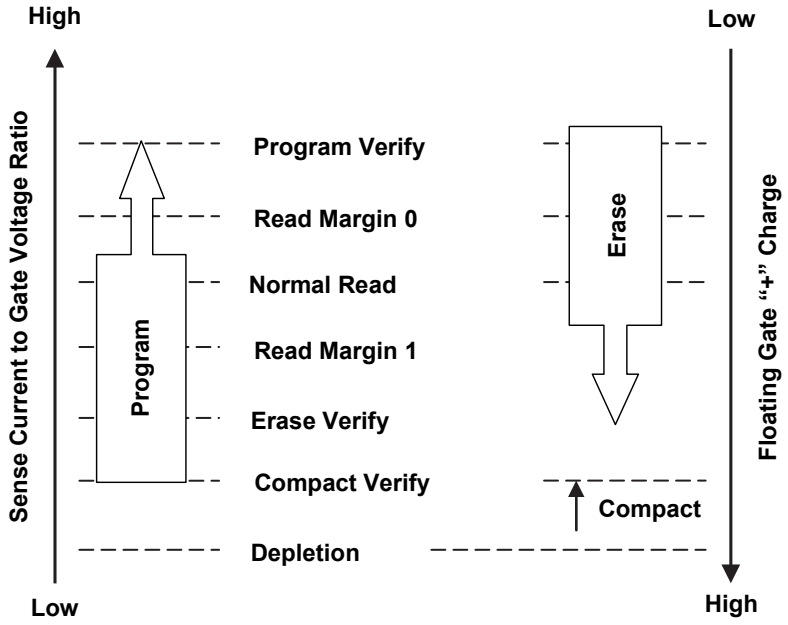


Figure 4. Flash Cell 0/1 Threshold in Different Reads

Copyright © 2010, Texas Instruments Incorporated

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:

```
//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; }
}
```


Copyright © 2010, Texas Instruments Incorporated

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 erasing:

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.

```
// 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;}
```

```
// 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.

Copyright © 2010, Texas Instruments Incorporated

The attached example uses following code to program 1k byte data into bank2 and bank3.

```
// 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)0xfff87000,&status);
if(!temp) {pstatus=1;}
temp=Flash_Prog_B((void *)0x00180000 ,(UINT32 *)(&Flash_Data+0x40), 0x300>>2,
       FLASH_CORE3, delay,(FLASH_ARRAY_ST)0xfff87000,&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)0xfff87000,&status);
if(!temp) {pstatus=0x20;}
temp=Flash_Prog_B((void *)0x004C0000 ,(UINT32 *)(&Flash_ECCData+0x20), 0x180>>2,
       FLASH_CORE3, delay, (FLASH_ARRAY_ST)0xfff87000,&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.

```
// 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)0xfff87000,&status);
if(!temp) {pstatus=0x1;}
temp=Flash_Verify_B((void *)0x00180000 ,(UINT32 *)((&Flash_Data)+0x40), 0x300>>2,
       FLASH_CORE3,(FLASH_ARRAY_ST)0xfff87000,&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)0xfff87000,&status);
if(!temp) {pstatus=0x40;}
temp=Flash_Verify_B((void *)0x004C0000 ,(UINT32 *)(&Flash_ECCData+0x20),
       0x180>>2, FLASH_CORE3, (FLASH_ARRAY_ST)0xfff87000,&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.

Copyright © 2010, Texas Instruments Incorporated

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.

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 1. Source Files in Project

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

It includes two project files:

Table 2. Project Files

File Name	Description
FlashEABI_Flash.pjt	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.
FlashEABI_RAM.pjt	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.

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
```


Copyright © 2010, Texas Instruments Incorporated

```
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
8- Call Flash_Compact_B() to compact bank1 unused sectors
9- Call Flash_Erase_B() to erase bank1 unused sectors
D- Dump 64 byte data from a certain location
```

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
- Cortex-R4 Processor http://www.arm.com/products/processors/cortex-r/cortex-r4.php

Copyright © 2010, Texas Instruments Incorporated

Appendix A Flash API Function Description

Table 3. Flash API Function Description

Function	Description
Flash_Compact_B() `BOOL Flash_Compact_B (UINT32 *pu32Start, FLASH_CORE oFlashCore, FLASH_SECT oFlashSector, UINT32 u32Delay, FLASH_ARRAY_ST oFlashControl, FLASH_STATUS_ST *oFlashStatus);`	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.
Flash_Erase_B() `BOOL Flash_Erase_B(UINT32 *pu32Start, UINT32 u32Length, FLASH_CORE oFlashCore, FLASH_SECT oFlashSector, UINT32 u32Delay, FLASH_ARRAY_ST oFlashControl, FLASH_STATUS_ST *oFlashStatus);`	This function is used to erase the targeted sector and to collect pulse count information. It allows for the disabling of reconditioning during erase.
Flash_Erase_Sector_B() `BOOL Flash_Erase_Sector_B(UINT32 *pu32Start, UINT32 u32Length, FLASH_CORE oFlashCore, FLASH_SECT oFlashSector, UINT32 u32Delay, FLASH_ARRAY_ST oFlashControl);`	This function erases a sector, and preconditioning is enabled by default and cannot be disabled as is possible using Flash_Erase_B or Flash_Erase_Bank_B.
Flash_Erase_Bank_B() `BOOL Flash_Erase_Bank_B(UINT32 *pu32Start, UINT32 u32Length, FLASH_CORE oFlashCore, UINT32 u32Delay, FLASH_ARRAY_ST oFlashControl, FLASH_STATUS_ST *oFlashStatus);`	This function is used to erase the targeted bank and to collect pulse count information. It allows for the disabling of reconditioning during erase.
Flash_Blank_B() `BOOL Flash_Blank_B(UINT32 *pu32Start, UINT32 u32Length, FLASH_CORE oFlashCore, FLASH_ARRAY_ST oFlashControl, FLASH_STATUS_ST *oFlashStatus );`	This function verifies that the Flash has been properly erased starting from the address passed in the parameter *start*. *length* words are read, starting at the starting address.
Flash_Prog_B() `BOOL Flash_Prog_B (UINT32 *pu32Start, UINT32 *pu32Buffer, UINT32 u32Length, FLASH_CORE oFlashCore, UINT32 u32Delay, FLASH_ARRAY_ST oFlashControl, FLASH_STATUS_ST *poFlashStatus );`	This function programs the Flash from the starting address *start* for *length* 32 bit words. The user code must make sure that the areas to be programmed are already erased before calling this routine.
Flash_Verify_B() `BOOL Flash_Verify_B(UINT32 *start, UINT32 *buff, UINT32 length, FLASH_CORE core, FLASH_ARRAY_ST cntl, FLASH_STATUS_ST *status, );`	This function verifies proper programming by using normal read, read-margin 0, and read-margin 1 modes. Verification starts from the start address *start* and check *length* words from the start address.
Flash_PSA_Verify_B() `BOOL Flash_PSA_Verify_B(UINT32 *start, UINT32 length, UINT32 psa, FLASH_CORE core, FLASH_ARRAY_ST cntl, FLASH_STATUS_ST *status );`	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 *start* and check *length* words from the start address.

Copyright © 2010, Texas Instruments Incorporated

Table 3. Flash API Function Description (continued)

Function	Description
OTP_Prog_B() `BOOL OTP_Prog_B (UINT32 *pu32Start, UINT32 *pu32Buffer, UINT32 u32Length, FLASH_CORE oFlashCore, UINT32 u32Delay, FLASH_ARRAY_ST oFlashControl, FLASH_STATUS_ST *poFlashStatus, );`	Similar to Flash_Prog_B, except that it is used to program OTP.

Table 4. Flash API Function Parameter Description

Parameter	Type	Description
`pu32Start / start`	UINT32	Points to the first word in the Flash that are compact/erased/programmed or verified
`pu32Buffer / buff`	UINT32	Pointer to the starting address of a buffer with data to program or verified
`u32Length /length`	UINT32	Number of 32-bit words to program/verify. This parameter in erase function can be ignored.
`oFlashCore / core`	FLASH_CORE	Bank select (0-7) of region being programmed/verified.
`u32Delay`	UINT32	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.
`oFlashControl / ctnl`	FLASH_ARRAY_ST	Flash control base address of module. 0xfff87000 for TMS570LS20216S.
`poFlashStatus / status`	FLASH_STATUS_ST	Pointer to status structure for storing statistical information.
`UINT32`	psa	The expected PSA value against which the actual PSA values is compared.

Copyright © 2010, Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2019, Texas Instruments Incorporated