Application Report
SPRAAL3–August 2008

Flash Programming Solutions for the TMS320F28xxx DSCs

Tim Love and Pradeep Shinde

ABSTRACT

Flash programming is a process that occurs during all stages of a TMS320F28xxx digital signal controller (DSC) development cycle: firmware debug, prototyping, production, and field reprogramming. Several solutions are available to accommodate programming for all of these development stages. This application report presents solutions that are available and at what stage in the development cycle these solutions are useful.

Contents

List of Figures

1 Introduction

The internal Flash memory of the TMS320F28xxx DSCs is a great benefit because it is non-volatile memory that allows designers to store application code internal to the chip as opposed to interfacing external memory to store this code.

Flash memory is composed of an array of memory cells made from floating-gate transistors. Each cell of the Flash can store one bit of information. A cell with a charge on the floating gate contains a value of 0 and a cell with little or no charge on the floating gate contains a value of 1. This technology requires supply voltage for the Flash to be present at all times. All TMS320F28xxx devices contain the V_{DD3VFL} voltage pin that requires 3.3 V applied to program (write) and read the Flash.

Because of its technology, the Flash must go through an erase, program, and verify procedure to store application code within the memory. The algorithms used for this functionality are time critical algorithms that execute on the DSC from the internal random access memory (RAM). These algorithms must be configured for the proper central processing unit (CPU) frequency and should not be interrupted to ensure proper programming of the Flash. TI provides the Flash application programming interface (API) algorithms at [1], [2], [7] and [8]. All of the Flash programming solutions discussed in this application report use these algorithms to program the Flash seamlessly from your interface. Figure 1 shows the overall scheme of the Flash API being utilized by a) JTAG, b) Serial, and c) custom solutions.

The first step is to know and understand the TI provided Flash tools that are downloadable from the web at http://www.ti.com/. Follow the path: TI Home → Digital Signal Processing → Processor Platforms → C2000™ High Performance 32-Bit Controllers → select the F28x generation and click on the Flash Tools button.

Note: It is necessary to use the correct version of the Flash API that matches the F28x part and its silicon version.

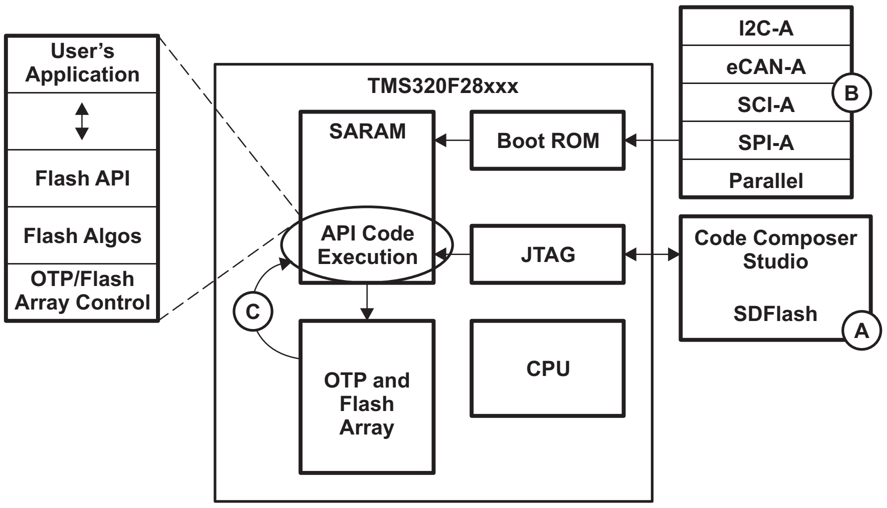


Figure 1. Flash API Code Execution

2 JTAG Solutions

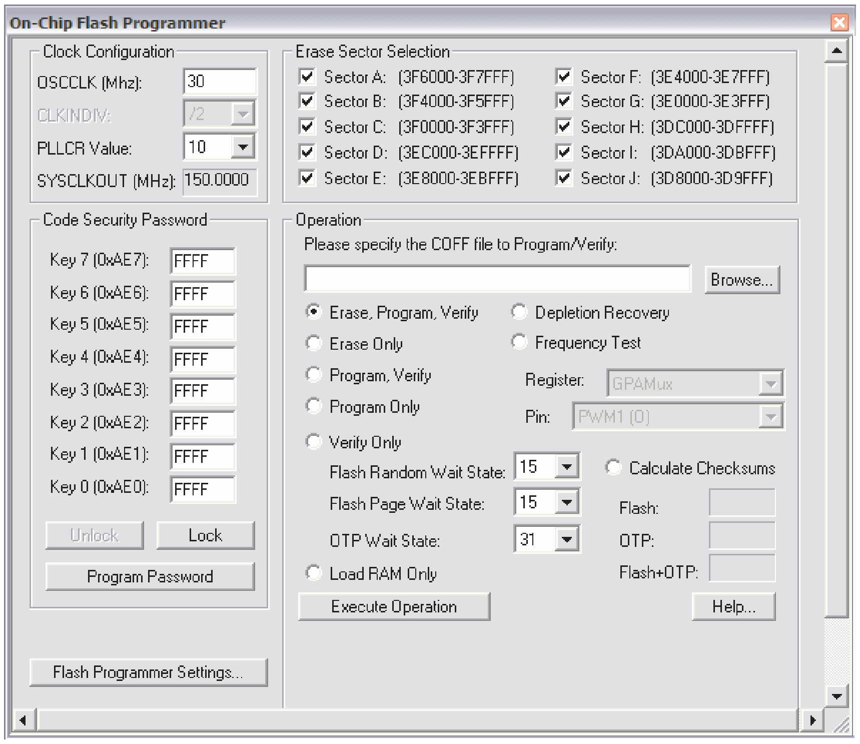
IEEE Standard 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture (JTAG) solutions can apply to all stages of the development cycle, but are predominately used in the firmware debug and prototyping stages as this method allows a designer to program the Flash and then debug it within the Code Composer Studio™ IDE. Several solutions exist including the Code Composer Studio On-Chip Flash Programmer, SD Flash, and Flasher-C2000. The Flash programming utility depends on the emulator being used.

2.1 Code Composer Studio On-Chip Flash Programmer

The Code Composer Studio On-Chip Flash Programmer is a plug in for Code Composer Studio that enables Flash programming within the IDE using any emulator that supports C2000 as well the eZdsp™ development board, and interfaces directly with Code Composer Studio. This programmer is the most convenient JTAG option during the firmware debug and prototyping stages, since the programmer can be accessed directly from Code Composer Studio.

The programmer is available in the Tools Menu of Code Composer Studio. If using Code Composer Studio 3.1 or older, this programmer is available at F281x Flash Tools [1], F280x Flash Tools [2] or within the Update Advisor of Code Composer Studio. If using Code Composer Studio 3.3, the programmer is installed with the base install of Code Composer Studio and updated through Service Releases available from the Update Advisor.

Figure 2 shows the Code Composer Studio On-Chip Flash Programmer graphical user interface (GUI).



OSCCLK (Mhz): 30
CLKINDIV: /2
PLLCR Value: 10
SYSCLKOUT (MHz): 150.0000

[✓] Sector A: (3F6000-3F7FFF)
[✓] Sector B: (3F4000-3F5FFF)
[✓] Sector C: (3F0000-3F3FFF)
[✓] Sector D: (3EC000-3EFFFF)
[✓] Sector E: (3E8000-3EBFFF)
[✓] Sector F: (3E4000-3E7FFF)
[✓] Sector G: (3E0000-3E3FFF)
[✓] Sector H: (3DC000-3DFFFF)
[✓] Sector I: (3DA000-3DBFFF)
[✓] Sector J: (3D8000-3D9FFF)

Key 7 (0xAE7): FFFF
Key 6 (0xAE6): FFFF
Key 5 (0xAE5): FFFF
Key 4 (0xAE4): FFFF
Key 3 (0xAE3): FFFF
Key 2 (0xAE2): FFFF
Key 1 (0xAE1): FFFF
Key 0 (0xAE0): FFFF

Please specify the COFF file to Program/Verify:

[✓ Erase, Program, Verify] Erase Only Program, Verify Program Only Verify Only Load RAM Only
Depletion Recovery Frequency Test
Register: GPAMux
Pin: PWM1 (0)
Flash Random Wait State: 15
Flash Page Wait State: 15
OTP Wait State: 31
Calculate Checksums
Flash:
OTP:
Flash+OTP:

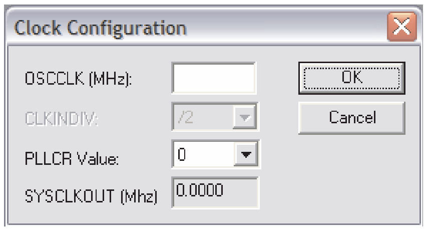
Figure 2. Code Composer Studio On-Chip Flash Programmer

2.1.1 On-Chip Flash Programmer Options

The On-Chip Flash Programmer has several options/features that can be utilized. Within the GUI, there are four visual sections and the Flash Programmer Settings button. The functionality of each portion is discussed in the following sections.

2.1.1.1 Clock Configuration

This section of the programmer configures the clocking for the Flash API algorithms. When opening the programmer, Code Composer Studio prompts you to configure these properties as shown in Figure 3.



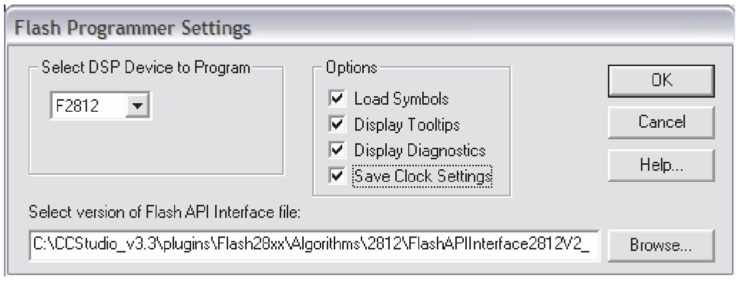
OSCCLK (MHz):
CLKINDIV: /2
PLLCR Value: 0
SYSCLKOUT (Mhz) 0.0000

Figure 3. Clock Configuration Setup

From this prompt, the input clock frequency will be specified and the PLLCR Value will be set. Based on these inputs, the programmer calculates the SYCLKOUT and configures the algorithms accordingly.

2.1.1.2 Flash Programmer Settings

The Flash Programmer Settings button opens the menu shown in Figure 4.



Select DSP Device to Program: F2812

Options:
- ✓ Load Symbols
- ✓ Display Tooltips
- ✓ Display Diagnostics
- ✓ Save Clock Settings

Select version of Flash API Interface file:

C:\CCStudio_v3.3\plugins\Flash28xx\Algorithms\2812\FlashAPIInterface2812V2_

Figure 4. Flash Programmer Settings Menu

The Flash API file is selected from this menu. When selecting this algorithm, the Browse button navigates you to the algorithms available on the system. Always choose the most recent algorithm.

The device can be selected from this menu. This defaults to the device that Code Composer Studio is configured for. The other options available are:

- Load Symbols - Enables/Disables Code Composer Studio to Load Symbols directly after programming Flash to enable debugging.
- Display Tooltips – Enables/Disables details within the programmer when hovering over an option.
- Display Diagnostics – Enables/Disables output screen that displays the status of programming.
- Save Clock Settings – Enables/Disables the option to save the clock settings entered in Section 2.1.1.1.

2.1.1.3 Erase Sector Selection

This section of the programmer allows you to select all or just certain sections of the Flash memory to erase. This option is helpful because it allows sectors to keep their contents if there is code programmed to that sector that should not be erased. Unused sectors can be deselected and will not be erased, which is also beneficial for decreasing overall erase time.

2.1.1.4 Operation

The operation portion of the programmer contains several features:

- Specifying a COFF File – The .out file to program to the Flash is input in this section. If a project is open and built in Code Composer Studio, the resulting .out file is automatically specified.
- Erase, Program, and Verify Operations – These buttons allow the erase, program, and verify operations to simultaneously run or execute each operation individually.
- Depletion Recovery – This option invokes the depletion recovery algorithm that looks for sectors that are in depletion and attempts to recover them.
- Frequency Test – The clock configuration specified from Section 2.1.1.1 can be verified on the general-purpose input/output (GPIO) pin of choice.
- Calculate Checksums – Performs a checksum of the Flash, one-time programmable (OTP), and Flash + OTP.
- Load RAM Only – This option loads initialized sections that are specified to run from RAM.
- Wait States – During programming, the Wait States for the Flash and OTP are set with these options.
- Execute Operation – This button will execute whatever option is selected.
- Help – This button will open the Help Files with all documentation associated for the programmer.

2.1.1.5 Code Security Password

The Code Security Password portion of the programmer directly accesses the Code Security Module (CSM) of the Flash memory. The CSM consists of eight 16-bit memory locations that enables the Flash to be password protected. For complete details regarding the CSM, see the *Code Security Module (CSM)* section of [10], [11] and [12].

This portion of the programmer contains the following elements:

- KEY0 – KEY7- 16-bit password locations.
- Unlock – Unlocks Flash if CSM is secured and passwords are written in KEY0 – KEY7.
- Lock – Locks the Flash if the CSM has previously been programmed with the passwords written in KEY0 – KEY7.
- Program Password – Programs the passwords to the CSM that is written in KEY0 – KEY7. The Flash is required to be erased for this process. If the Flash has not been erased when the Program Password button is pressed, the Flash Plug In will prompt you to erase the Flash.

There are several precautions to be aware of when using the CSM. For further details, see Section 6.3.

2.1.2 Programming Example

The process of programming and debugging from the Flash only contains a few steps. For this example, the TMS320F28335 eZdsp, Code Composer Studio 3.3, and Flash example from *Running an Application from Internal Flash Memory on the TMS320F28xxx DSP* (SPRA958) [3] were used. This process is universal for all TMS320F28xxx DSCs.

1. Connect the target board to the PC using the JTAG emulator and power the board with the appropriate power connector.
2. Start Code Composer Studio with the appropriate emulation driver selected in the Code Composer Studio setup utility.
3. Open and Build the project by selecting Project → Open followed by Project → Rebuild All.
4. Open the On-Chip Flash Programmer from the Tools Menu.

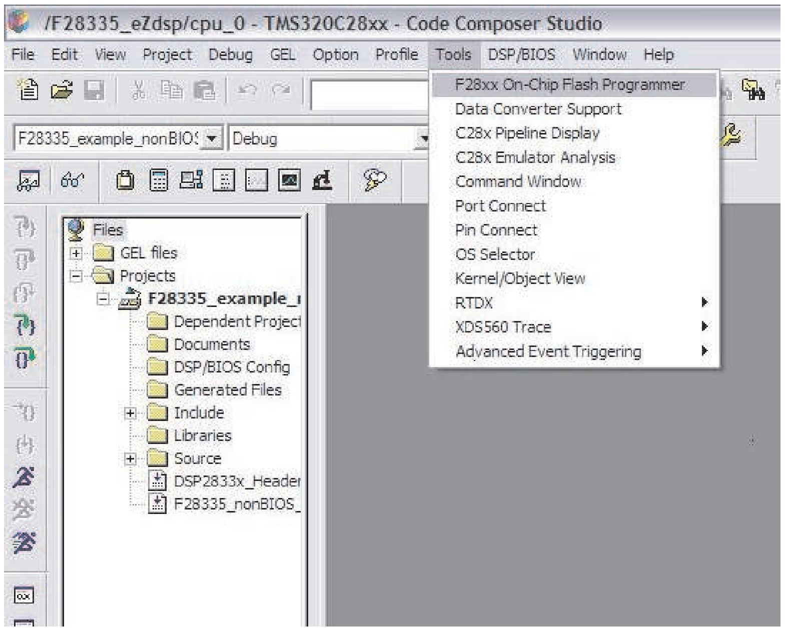
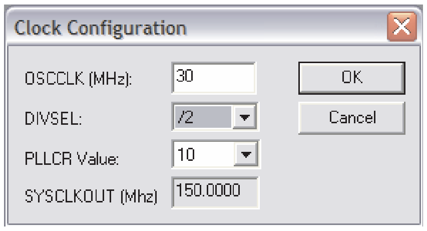


Figure 5. Invoking On-Chip Flash Programmer

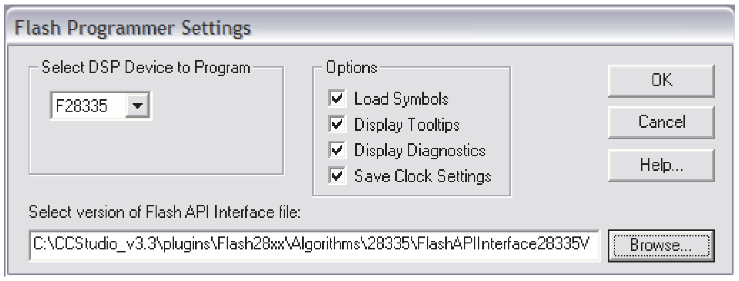
5. Configure Clock Settings for the target board as described in Section 2.1.1.1.



OSCCLK (MHz): 30
DIVSEL: /2
PLLCR Value: 10
SYSCLKOUT (Mhz) 150.0000

Figure 6. Example Clock Configuration Settings

6. Select the correct algorithm for the programmer as described in Section 2.1.1.2.



Select DSP Device to Program: F28335

Options:
- ✓ Load Symbols
- ✓ Display Tooltips
- ✓ Display Diagnostics
- ✓ Save Clock Settings

Select version of Flash API Interface file: C:\CCStudio_v3.3\plugins\Flash28xx\Algorithms\28335\FlashAPIInterface28335V

Figure 7. Example Flash Programmer Settings

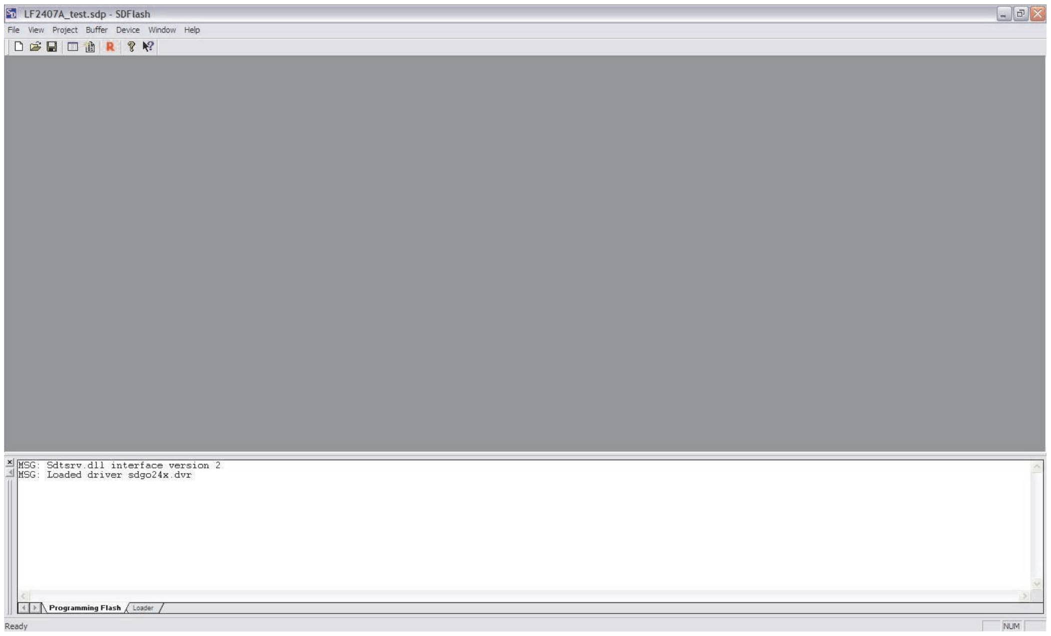
7. Select the sectors that will be erased/programmed as described in Section 2.1.1.3.
8. Select Execute Operation as the Erase, Program, Verify option is default and the .out file should be specified already from building the project.

Once this process has finished, the program is stored on the Flash and the board is ready for standalone operation or debugging, through Code Composer Studio, by using the method described in Section 6.1.

2.2 SD Flash

SD Flash is a free, standalone programmer that enables programming with a Spectrum Digital emulator without requiring the need of Code Composer Studio. Because of this, this programmer is useful for all development stages of a product as well. The utility, algorithms, example projects, and full documentation explaining how to use the utility are available at the SD Flash Utility home page [4].

The SD Flash GUI is shown in Figure 8.


LF2407A_test.sdp - SDFlash
File View Project Buffer Device Window Help
Programming Flash
Loader
Ready
NUM

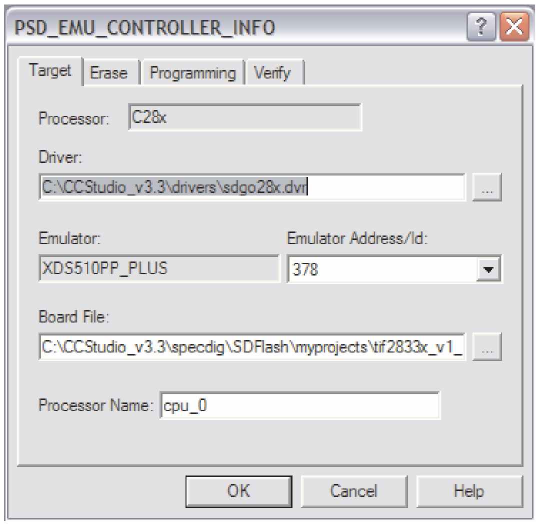
```
MSG: Sdtsrv.dll interface version 2
MSG: Loaded driver sdgo24x.dvr
```

Figure 8. SD Flash

SD Flash uses projects to configure all settings for the JTAG connection, erase, programming, and verify options. A new project can be created or an existing project can be opened from the File menu. Once the project is opened, the settings are configured by selecting Project -> Settings.

2.2.1 Target

Figure 9 shows the target options. The target tab configures the JTAG connection for the device options.



Processor: C28x
Driver: C:\CCStudio_v3.3\drivers\sdgo28x.dvr
Emulator: XDS510PP_PLUS
Emulator Address/Id: 378
Board File: C:\CCStudio_v3.3\specdig\SDFlash\myprojects\tif2833x_v1_
Processor Name: cpu_0

Figure 9. SD Flash Target Options

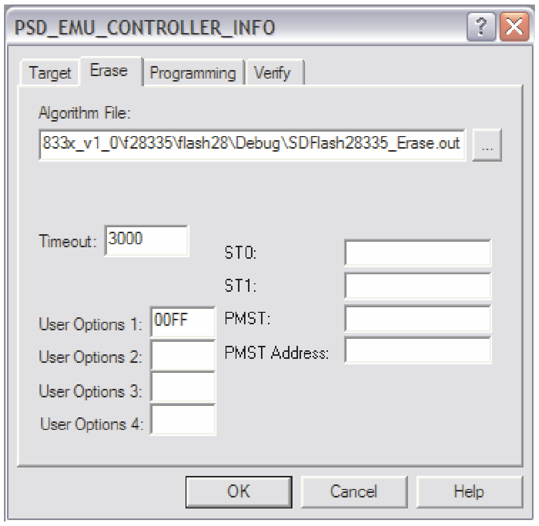
The options within this tab are as follows:

- Driver – Selects the Code Composer Studio emulation driver used to communicate with the device.
- Emulator Address/ID – Selects the address for the JTAG emulator. The default is 0x378 for the XDS510PP and 510 for XDS510USB.

- Board File - Provides information to the SD Flash regarding the number of devices on the JTAG scan chain.
- Processor Name – Used to place a generic name for the device within the project.

2.2.2 Erase

Figure 10 shows the erase tab that configures all options for the erase process.



Algorithm File: 833x_v1_0\f28335\flash28\Debug\SDFlash28335_Erase.out

Timeout: 3000

User Options 1: 00FF
User Options 2:
User Options 3:
User Options 4:

ST0:
ST1:
PMST:
PMST Address:

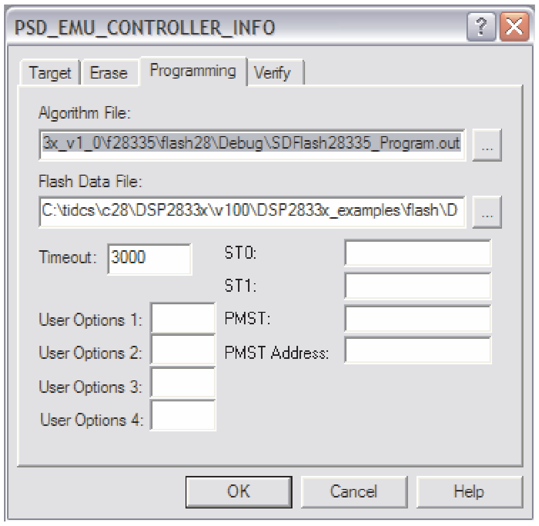
Figure 10. SD Flash Erase Options

The options within this tab are as follows:

- Algorithm File – Specify erase algorithm for project. The file should be selected for the specific device being used.
- Timeout – Set timeout for the erase process in terms of seconds.
- User Options 1 - Specify which sectors to erase. 00FF is default and will specify all sectors.
- User Options 2/User Options 4 – Not Used.
- User Options 3 – Run frequency toggle test.
- ST0/ST1/PMST/PMST Address – Default is blank and should not be used.

2.2.3 Programming

Figure 11 shows the programming tab that configures all options for the programming process.



Algorithm File: 3x_v1_0\f28335\flash28\Debug\SDFlash28335_Program.out

Flash Data File: C:\tidcs\c28\DSP2833x\v100\DSP2833x_examples\flash\D

Timeout: 3000

ST0:

ST1:

User Options 1:

PMST:

User Options 2:

PMST Address:

User Options 3:

User Options 4:

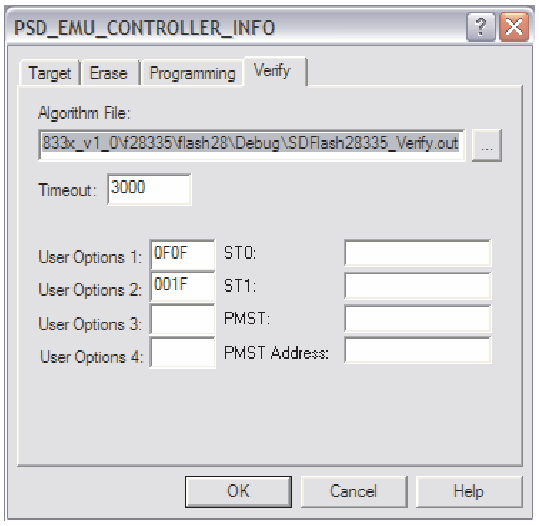
Figure 11. SD Flash Programming Options

The options within this tab are as follows:

- Algorithm File – Specify programming algorithm for project. The file should be selected for the specific device being used.
- Flash Data File – Specify .out file produced from Code Composer Studio project to program to the Flash.
- Timeout – Set timeout for the programming process in terms of seconds.
- User Options 3 – Run frequency toggle test.
- User Options 1/User Options 2/User Options 4 – Not Used.
- ST0/ST1/PMST/PMST Address – Default is blank and should not be used.

2.2.4 Verify

Figure 12 shows the verify tab that configures all options for the verify process .



Algorithm File: 833x_v1_0\f28335\flash28\Debug\SDFlash28335_Verify.out

Timeout: 3000

User Options 1: 0F0F

ST0:

User Options 2: 001F

ST1:

User Options 3:

PMST:

User Options 4:

PMST Address:

Figure 12. SD Flash Verify Options

The options within this tab are as follows:

- Algorithm File – Specify verify algorithm for project. The file should be selected for the specific device being used.
- Timeout – Set timeout for the verify process in terms of seconds.
- User Options 1/User Options 2 – Specify wait states used for the Flash and OTP during the verify operation.
- User Options 3 - Run frequency toggle test.
- User Options 4 – Not Used.
- ST0/ST1/PMST/PMST Address – Default is blank and should not be used.

2.2.5 Programming Example

The process of programming the Flash only contains a few steps. For this example, the TMS320F28335 eZdsp, SampleF28335usb.sdp (provided SD Flash project), and Flash example from *Running an Application from Internal Flash Memory on the TMS320F28xxx DSP* (SPRA958) [3] were used. This process is universal for all TMS320F28xxx DSCs.

1. Download the SD Flash utility and algorithms from SD Flash [4] .
2. Connect the target board to the PC using the JTAG emulator and power the board with the appropriate power connector.
3. Open SD Flash.
4. Open the provided SD Flash project file by selecting File → Open Project.
5. Modify any project options described in Section 2.2.1, Section 2.2.2, Section 2.2.3, and Section 2.2.4 as needed. (In this example, the .out file was changed in the Flash Data File to use the .out file from *Running an Application from Internal Flash Memory on the TMS320F28xxx DSP* (SPRA958) [3].)
6. Save the project file by selecting File → Save Project As.
7. Reset the device by selecting Device → Reset.
8. Program the Flash by selecting Device → Flash. A separate menu will open as shown in Figure 13. All options or only certain options can be selected for the flashing process.

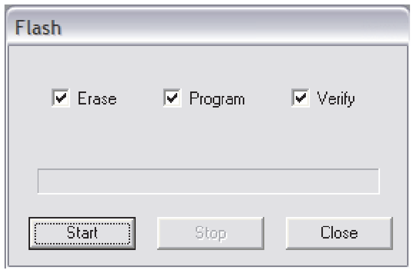


Figure 13. Flash Window

Once this process has finished, the program is stored on the Flash and the board is ready for standalone operation or debugging through Code Composer Studio by using the method described in Section 6.1. The documentation supplied with SD Flash explains all the steps described in this section in detail and provides further information regarding clock rate configuration, CSM, depletion recovery, etc. It is recommended to read this documentation for complete information regarding SD Flash.

2.3 *Flasher-C2000*

Flasher-C2000 is a standalone programmer that enables programming using a Signum™ JTAGjet-TMS-C2000 emulator without the need of Code Composer Studio, making this programmer useful for all development stages of a product. The utility can be purchased from the Signum JTAGjet-TMS-C2000 home page at http://www.signum.com/Signum.htm?p=c2000.htm [5]. For further information regarding this programmer please contact the Signum Systems Corp.

3 Serial Solution

SD Flash also has free RS-232 algorithms used to program the Flash through the SCI-A peripheral using the serial communications interface (SCI) Boot ROM option. This programming solution is useful for production and field reprogramming stages of the development cycle since SD Flash is standalone and the only connection needed is a standard RS-232 cable.

This solution contains the same graphic interface as shown in Section 2.2 and uses the same options. The only difference is that the RS-232 interface is used instead of JTAG. Using this method, the steps for programming this Flash are the same as Section 2.2.5 with the only exception of connecting the RS-232.

1. Connect a serial cable between the SCI-A port and the host PC.
2. Set the F28xxx device for boot to the SCI-A serial bootload.
3. Pull the XMP/MC pin low for access to Boot ROM.
4. Disconnect any JTAG emulator that may be connected to the board.

Full documentation describing these steps and other information regarding this programming solution, the utility, and examples are available for download at http://emulators.spectrumdigital.com/utilities/sdflash/ [4].

4 Embedded Solution

Some applications require re-programming all or part of the Flash memory or OTP block to reconfigure the application code or to store data (e.g., calibration parameters) in the non-volatile memory. This operation is also referred to as in-circuit programming; in this context, CPU programming the Flash on the system.

4.1 Required Set Up

Since the Flash API program is required for any Flash operation, this has to be made available during embedded programming. The Flash API needs to be included in the application software that resides in the Flash. As discussed earlier, the Flash routines (erase, write operations) have to follow strict timing requirements. Therefore, they cannot be run from Flash due to its slower speed. At run time, entire Flash API needs to be copied to RAM; code/data to be written also needs to reside in the RAM.

It is important that the Flash programming voltage pin V_{DD3VFL} is connected to the 3.3-V supply rail. Note that the Flash erase and write operations draw additional current during this operation. To get the typical value of the current drawn from V_{DD} and V_{DDIO} supply rails, see the Flash Parameter Table in the device-specific data sheets. The system power supply must be capable of supporting in-excess of this current requirement for the reliable operations.

4.2 Flash API

This section takes a closer look at the Flash API since it is resident and called for by the CPU for various Flash operations.

The API library includes functions to erase, program and verify the Flash array. The smallest amount of memory that can be erased at a time is a single sector. The Flash API erase function includes the Flash pre-conditioning and a separate *clear* step is not required. The program function operates on both the Flash array and the OTP block. The program function can only change bits from a 1 to a 0. Bits cannot be moved from a 0 back to a 1 by the programming function. The programming function operates on a single 16-bit word at a time.

For complete details on all options and operating procedures of the Flash API library, see the API documentation included in the API zip file.

4.3 *Flash API Checklist*

The following data is taken from the API document as a reference and for completeness.

Integration of the Flash APIs into user software requires that the system designer implement operations to satisfy several key requirements. The following checklist gives an overview of the steps required to use the API. These steps are further discussed in detail in the reference section indicated. This checklist applies to all of the F2823x Flash APIs and was taken from the F2823x Flash API documentation. This general checklist applies to all F28xxx Flash API Libraries and can be found in each library's respective documentation.

Before using the API, do the following:

1. Modify Flash2823x_API_Config.h for your target operating conditions.
2. Include Flash2823x_API_Library.h in your source code.
3. Add the proper Flash API library to your project.
   - When using the Flash API, build your code with the large memory model.
   - The API Library is built in 28x Object code (OBJMODE = 1, AMODE = 1)

Do the following in your application before calling any Flash API functions:

1. Initialize the PLL control register (PLLCR) and wait for the PLL to lock.
2. Make sure the PLL is not running in limp mode. If the PLL is in limp mode, do not call any of the API functions as the device will not be running at the proper frequency.
3. The API must execute from zero-wait state internal SARAM (optional). If the API is to be copied from Flash/OTP into internal SARAM memory, then follow the instructions in this section.
4. Initialize the 32-bit global variable Flash_CPUScaleFactor.
5. Initialize the global function pointer Flash_CallbackPtr to point to the application's callback function. Alternatively, set the pointer to NULL.
6. Disable the global interrupts before calling an API function (optional).
7. Understand the API restrictions that are detailed in this section before making any APIcalls.
8. Run the frequency toggle test to confirm proper frequency configuration of the Flash API (optional).

Note: The ToggleTest function will execute forever. You must halt the processor to stop this test.

9. Unlock the CSM (optional).
10. Call the Flash API Functions that are described in the API Reference.

The called Flash API function will do the following:

- The watchdog timer is disabled.
- Checks the PARTID (memory location 0x0882) register to make sure the part is the correct device
- Checks the contents of 0x3FFFB9 in the boot ROM for API version vs. silicon revision compatibility.
- Performs the called operation and:
  – Disables and restores global interrupts (via INTM, DBGM, XNMICR) around time critical code segments.
  – Invokes the callback function if Flash_CallbackPtr is not NULL
- Returns success or an error code. These are defined in F2823x_API_Library.h.

Your code should then do the following:

1. Check the return status against the error codes.
2. Re-enable the watchdog timer (optional).

4.4 Flash API Do’s and Don’t’s

- API Do’s
  - Execute the Flash API code from zero-wait state internal SARAM memory. The F2823x and F2833x devices contain both zero wait state (L0-L3) and one wait state SARAM (L4-L7). The Flash APIs should be run from L0-L3 SARAM.
  - Configure the API for the correct CPU frequency of operation.
  - Follow the checklist of the Flash API documentation to integrate the API into an application.
  - Initialize the PLLCR and wait for the PLL to lock before calling an API function.
  - Initialize the API callback function pointer (Flash_CallbackPtr). If the callback function is not going to be used, then it is best to explicitly set the function pointer to NULL as described in the API documents in section *Initialize the Callback Function Pointer*. Failure to initialize the callback function pointer can cause the code to branch to an undefined location.
  - Carefully review the API restrictions for the callback function, interrupts, and watchdog that are described in the document.
- API Don’ts
  - Don not execute the Flash APIs from the Flash or OTP. If the APIs are stored in Flash or OTP memory, they must first be copied to internal zero-wait state SARAM before they are executed.
  - Do not execute any interrupt service routines (ISRs) that can occur during an erase, program or depletion recovery API function from the Flash or OTP memory blocks. The Flash and OTP are not available for program execution or data storage until the API function completes and exits.
  - Do not execute the API callback function from Flash or OTP. When the callback function is invoked by the API during the erase, program or depletion recovery routine, the Flash and OTP are not available for program execution or data storage. The Flash and OTP will be available only after the API function completes and exits.
  - Do not stop the erase, program or depletion recovery functions while they are executing (for example, don’t stop the debugger within API code, don’t reset the part, etc).
  - Do not execute code or fetch data from the Flash array or OTP while the Flash and/or OTP is being erased, programmed or during depletion recovery.

5 Production Solutions

Programming the application code on to fresh parts is done by non-engineering staff and needs to be done at a faster speed. Various solutions are available based on the production volumes.

For smaller volumes, a good option for in-house programming is the Serial Utility available from third parties like Spectrum Digital and Signum. These are standalone software utilities (running on PC) that are not linked to Code Composer Studio. They normally accept the COFF (.out) file of the project and communicate with the CPU via SCI/RS232 port. Check the websites of these vendors for the details.

Texas Instruments distributors Arrow and Avnet provide the services to program the devices with the application image file. This would be useful for medium volumes (> 1K).

For larger quantities, consider the standalone Flash programmers available from vendors like BP Micro and Data I/O. They constantly add new devices to the list of parts which can be programmed.

All of these options are accessible at F281x Flash tools [1] and F280x Flash tools [2].

6 Debugging, Useful Documentation, and Precautions

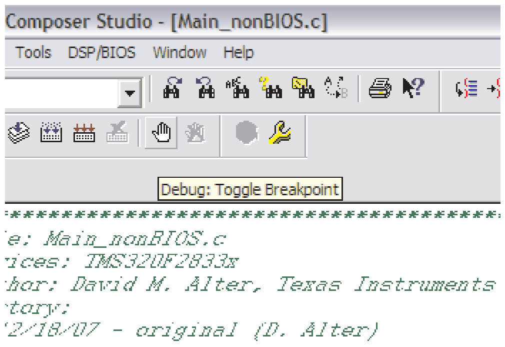
The Flash programming solutions presented in this application report enable a designer to store and run application code on the device. Along these same lines, a designer can use Code Composer Studio to debug directly from the Flash through JTAG. There are also several useful documents, precautions, etc., to take into consideration when using the Flash memory and programming solutions.

6.1 Debugging in Code Composer Studio Environment

After the Flash process has completed using one of the presented Flash solutions, the program can be debugged directly from the Flash within Code Composer Studio. All standard debug options can be used when debugging from Flash: Run, Halt, Single Step, Reset CPU, etc.

The only step needed to program from the Flash is to select File → Load Symbols → Load Symbols Only and select the .out file that was programmed to the Flash.

One limitation to debugging is the use of breakpoints. Since debugging is occurring from the Flash memory, you are limited to two hardware breakpoints. These breakpoints can be set just like software breakpoints by selecting the Toggle Breakpoint option (see Figure 14).


Composer Studio - [Main_nonBIOS.c]
Tools DSP/BIOS Window Help
Debug: Toggle Breakpoint

```
*****************************************
e: Main_nonBIOS.c
ices: TMS320F2833x
hor: David M. Alter, Texas Instruments
tory:
2/18/07 - original (D. Alter)
```

Figure 14. Toggle Breakpoint

Note: Code Composer Studio automatically sets two breakpoints for end of program and CIO preventing you from having any breakpoints for debugging. These breakpoints can be enabled by navigating to Option → Customize → Program/Project/CIO and enabling the two checkboxes titled Do Not Set End of Program Breakpoint At Load and Do Not Set CIO Breakpoint At Load.

6.2 Useful Documentation

The documentation provided with the programming solutions is very useful for describing how to use the tool and options that need to be set within the tool. Before the tool can be used, however, the project must be configured to run from Flash. There is a detailed document available from TI that describes how to convert a project from RAM based to Flash based and supplies examples showing a converted project. For more details, see *Running an Application from Internal Flash Memory on the TMS320F28xxx DSP* (SPRA958) [3].

Another document from TI concerns loading code into the Flash and then copying the complete code into RAM at run time for faster execution. For more details, see *Copying Compiler Sections From Flash to RAM on the TMS320F28xxx DSCs* (SPRAAU8) [6]. [3] also discusses copying code sections from Flash to RAM at runtime for just specific code functions not the complete code.

Finally, TI has a document displaying the use of the SCI Boot option of the TMS320F281x Boot ROM to perform serial programming of the Flash using the Flash API. For more details, see *TMS320F281x Boot ROM Serial Flash Programming* (SPRAAQ2) [9]. This is a very useful document to demonstrate in field reprogramming.

6.3 Precautions

There are a few precautions that you should be aware of to keep from permanently locking or leaving the Flash in an unknown state:

- Do not stop the erase process in the middle of execution as this could permanently lock the CSM and could also result in depletion of the Flash cells. The erase process first programs all cells as zeroes and then removes the charge from the bits in the sector until all bits are erased. If the execution stops after the CSM locations are programmed as zeroes but before the charge is removed, it will become permanently locked. All zeroes in the CSM locations is a chip feature to allow permanent security. Depletion can also occur as a result of not allowing the erase procedure to perform its post condition step, which ensures that bits are not left in the depleted state.

Note: If Flash is in depletion mode, the depletion recovery functionality provided with the Code Composer Studio On-Chip Flash Programmer and SD Flash can be used to attempt to bring the Flash out of depletion.

- Ensure that the CSM password locations located at addresses 0x3F7FF8 – 0x3F8000 are not used for code allocation within the linker command file of the Code Composer Studio project. If code is loaded to the CSM password locations, the password will be written with the project code and the password will be unknown causing the Flash to be permanently locked.
- When using the Flash API, you cannot modify the Flash memory while running from Flash requiring that the algorithms be ran from internal RAM.
- It is important that the Flash programming voltage pin V_{DD3VFL} is connected to the 3.3-V supply rail at all times. This voltage is needed to read/program the Flash. Please note that Flash erase and write operations draw additional current during this operation. To get the typical value of the current drawn from V_{DD} and V_{DDIO} supply rails, see the Flash Parameter Table in the device-specific data sheets. The system power supply must be capable of supporting in-excess of this current requirement for the reliable operations.

7 Conclusion

This application report shows that there are several stages in the development cycle and provides Flash programming solutions to accommodate these stages. These options include: JTAG, Serial, Embedded, and production solutions. There is recommended documentation to aid in the development as well as some guidelines and pitfalls to watch for. With all of this information, you will be well equipped to handle the Flash memory of the TMS320F28xxx DSCs.

8 References

1. F281x Flash Tools: http://focus.ti.com/dsp/docs/dspplatformscontento.tsp?sectionId=2&familyId=1406&tabId=2026
2. F280x Flash Tools: http://focus.ti.com/dsp/docs/dspplatformscontento.tsp?sectionId=2&familyId=510&tabId=517
3. *Running an Application from Internal Flash Memory on the TMS320F28xxx DSP* (SPRA958)
4. SD Flash: http://emulators.spectrumdigital.com/utilities/sdflash/
5. Signum: http://www.signum.com/Signum.htm?p=c2000.htm
6. *Copying Compiler Sections From Flash to RAM on the TMS320F28xxx DSCs* (SPRAAU8)
7. *Download: TMS320F2823x Flash APIs* (SPRC665)
8. *Download: TMS320F2833x Flash APIs (v2.00)* (SPRC539)
9. *TMS320F281x Boot ROM Serial Flash Programming* (SPRAAQ2)
10. *TMS320x281x DSP System Control and Interrupts Reference Guide* (SPRU078)
11. *TMS320x280x, 2801x, 2804x DSP System Control and Interrupts Reference Guide* (SPRU712)
12. *TMS320x2833x, 2823x System Control and Interrupts Reference Guide* (SPRUFB0)