Application Report
SPRAAW0–November 2009

Programming External Nonvolatile Memory Using SDFlash for TMS320C28x Devices

Devin Cottier

ABSTRACT

The C28xxx generation of microcontrollers can connect to external nonvolatile memory through a variety of interfaces including parallel, serial peripheral interface (SPI), and inter-integrated circuit (I2C). This application report discusses how to write drivers for your external nonvolatile memory to allow for in-place programming using SDFlash, an application provided by Spectrum Digital, Inc. As a reference, it also presents a complete external memory design including schematic, layout, external memory drivers, and SDFlash programming algorithms.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/spraaw0.zip.

Contents

1 Introduction 1
2 SDFlash Overview 2
3 Installing the Sample Project 3
4 Using SDFlash 3
5 Writing Drivers for Your Memory 7
6 Example System: AT49BV802D 8M-Bit Memory and Modified C28346 Control Card 17
7 References 20

List of Figures

1 SDFlash Overview 2
2 Missing Title 3
3 Target Tab 4
4 Erase Tab 5
5 Programming Tab 6
6 Verify 7
7 Initializations 10
8 Erase 11
9 Program Flow 13
10 Verify Flow 15
11 Schematic 19
12 Layout Illustration of Board 19

1 Introduction

Interfacing to external nonvolatile memory is an important part of an embedded system, especially when using a RAM-based microcontroller such as the TMS320C2834x. It is possible to use the host C28x™ DSP device to program your external memory in place, instead of using a standalone Flash/EEPROM programmer. An easy way to do this is to use SDFlash, a GUI-based memory programming application provided by Spectrum Digital Inc.

Several SDFlash algorithms are available for programming specific external memories. These include C2834x algorithms for the Atmel memories AT49BV802D, AT25HP256/512, AT25256A, and AT25F1024A. These algorithms, along with documentation on how to use them, are available directly from Spectrum Digital's website at http://emulators.spectrumdigital.com/utilities/sdflash/.

Since there are many other possible external memories to choose from, this document describes how to create SDFlash algorithms. It discusses how SDFlash interfaces to the C28x device and the external memory and what is needed to build an SDFlash algorithm. This allows for in-place programming of your external memory using the SDFlash tool. The AT49BV802D algorithms, that connect to the external memory through the external interface peripheral (XINTF), are used frequently as an example.

2 SDFlash Overview

SDFlash is a generic front-end application owned by Spectrum Digital Inc (www.spectrumdigital.com). This application provides a generic interface to the IEEE Standard 1149.1-1990, IEEE Standard Test Access Port Bondary-Scan Architecture (JTAG) communications channel that can be used to support programming of nonvolatile memory.

To perform the required operations, SDFlash downloads an algorithm file onto the C28x device. This algorithm file is an executable (.out) file for the C28x device. The SDFlash algorithm project contains an SDFlash wrapper. The SDFlash wrapper acts as an interface for the SDFlash application and has well defined standard functions and variables that are accessed by SDFlash over the JTAG channel. In turn, these functions make calls to the low-level drivers that program the external memory.

NOTE: Because the SDFlash application is separate from the algorithm file, the version of the SDFlash application differs from the version of the algorithm file.

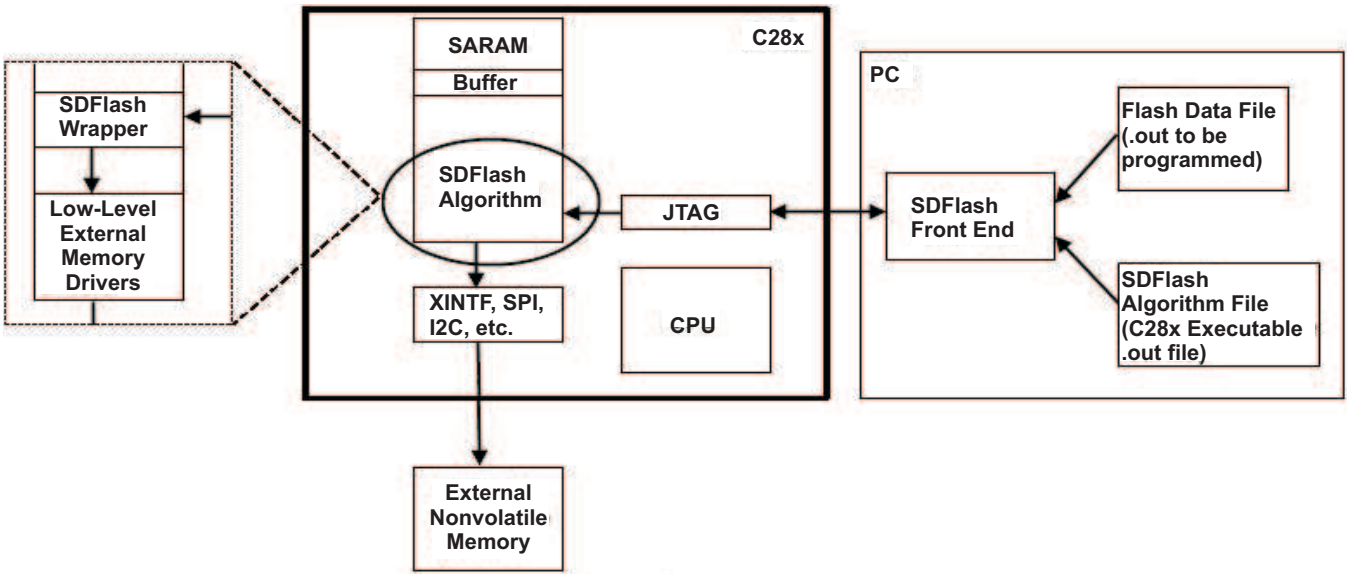


Figure 1. SDFlash Overview

The SDFlash application runs on the PC. You will choose the location of the .out file produced by compiling the SDFlash algorithm project. You will choose the location of the .out file containing the data to be programmed into memory and any subset of the operations to performed on the device: erase, program, and verify.

Then, SDFlash loads the algorithm .out file onto the host C28x device using the JTAG port. The algorithm loaded onto the C28x has two parts. The first part is the SDFlash wrapper that serves as the interface to the SDFlash application. This part of the algorithm contains wrapper functions that SDFlash can call and

wrapper variables to which SDFlash can read and write. The second part of the SDFlash algorithm consists of low-level drivers specific to the external memory being programmed. These drivers are called by the wrapper functions. The low-level drivers interface to the external memory through some peripheral interface such as external interface (XINTF), SPI, or I2C. They configure the communication peripheral and use it to send commands to the external memory device.

To perform an operation (erase, program, or verify), the SDFlash application sets the PC to point to the location of the SDFlash wrapper function corresponding to that operation. It also sets the value of some of the wrapper variables to tell the operation exactly what to do. Then, it runs the wrapper function. The wrapper function for the particular operation calls the low-level drivers for the external memory. The low-level drivers send commands to the external memory through the communication peripheral.

3 Installing the Sample Project

An installer for the example SDFlash algorithm files is included with this download and is named *sd2834x_XINTF_flash.exe*. Run the program to unpack and install the sample project.

This installer will by default place the sample project in the directory `C:\CCStudio_v3.3\specdig\sdflash\myprojects\` and the sample project folder will be named `tif2834x_XINTF_flash_v1`. The rest of this document will use `<ProjectDir>` to indicate the directory where the example project was installed. If you change the installation directory from the default, `<ProjectDir>` will indicate the directory where the example project was actually installed.

4 Using SDFlash

The following section gives a brief overview on how to configure SDFlash and run the erase, program, and verify functions. For a more detailed checklist for running SDFlash, see the documentation provided with the sample algorithm project. This can be found at *<ProjectDir>\tif2834x_XINTF_flash_v1\doc\SDFlash2834x_Flash_XINTF_Readme_V1_0.pdf*.

SDFlash is included with Code Composer Studio™ software. It can be found in the directory `<CCSbase>\specdig\SDflash\` where `<CCSbase>` is `c:\CCStudio_v3.3\` or whatever directory Code Composer Studio was installed to.

To open the SDFlash application, run the executable file *<CCSbase>\specdig\SDflash\bin\SDFlash.exe*.

For SDFlash to program the external nonvolatile memory, it must have the following information:

- Location of the programming algorithm file (C28x executeable .out file)
- Location of the Flash data file – that is the data (.out file) to program into the external memory device
- Which JTAG driver to use
- Information about the JTAG scan chain

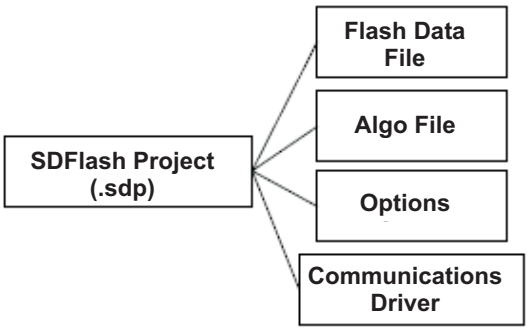


Figure 2. Missing Title

All of this information is stored in an SDFlash project file (.sdp) that can be edited though the SDFlash GUI interface. A sample SDFlash project has been provided in this download.

To open a .sdp project, click File → Open project and select the project you want to open. The example project for the C2834x algorithms to program the AT49BV802D through the XINTF can be found at `<ProjectDir>\tif2834x_XINTF_flash_v1\Sample_DSP2834x_XINTF_ExtFlash.sdp`.

To configure your project click on Project → Settings.

This brings up a configuration window with four tabs.

The first tab is labeled *Target*. Under *Driver* select the driver for your JTAG emulator. The emulator drivers are stored in default directory `\<CCSbase>\drivers\`. Also select the emulator address corresponding to your JTAG emulator.

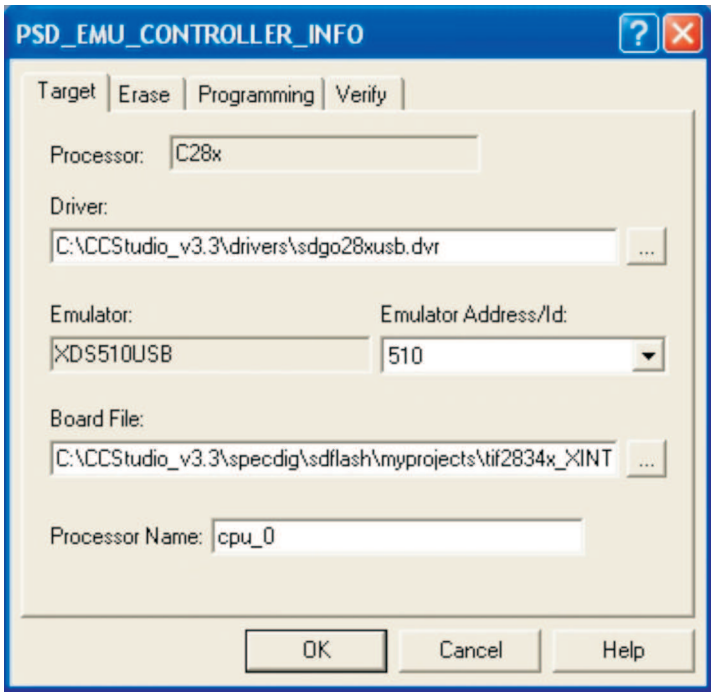


Figure 3. Target Tab

The second tab is labeled *Erase*. For the field *Algorithm File* locate the .out file produced by compiling your algorithm project. Timeout is how many seconds SDFlash will wait before failing the erase operation. User Options 1 – 4 are 16-bit hex numbers that are passed onto the SDFlash wrapper functions. The wrapper function may not do anything with some or all of this data depending on the particular algorithm.

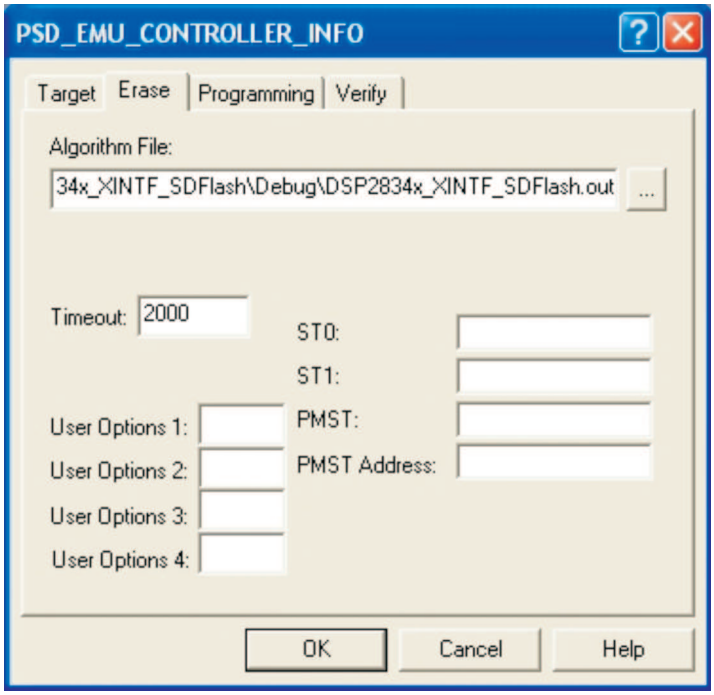


Figure 4. Erase Tab

The third tab is labeled *Programming*. The *Flash Data File* field should be directed to the .out file that contains the data to be programmed into the external memory. The *Algorithm File* field is the same as in *Erase*. The option fields function the same way as in the *Erase* tab. However, values different from those in the erase tab are entered that are specific to the *program* part of the algorithm.

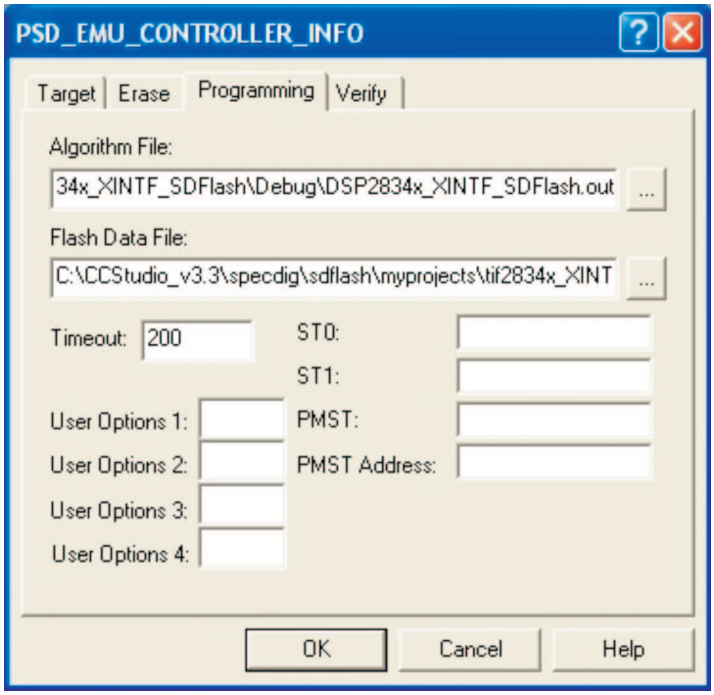


Figure 5. Programming Tab

The fourth tab is labeled *Verify*. Select the *Algorithm File*, timeout, and options in the same way they were determined for the previous tabs.

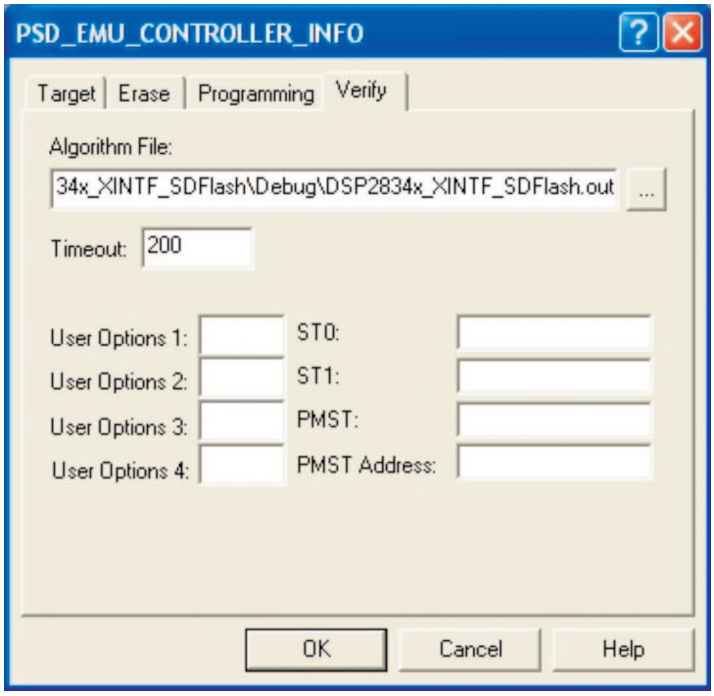


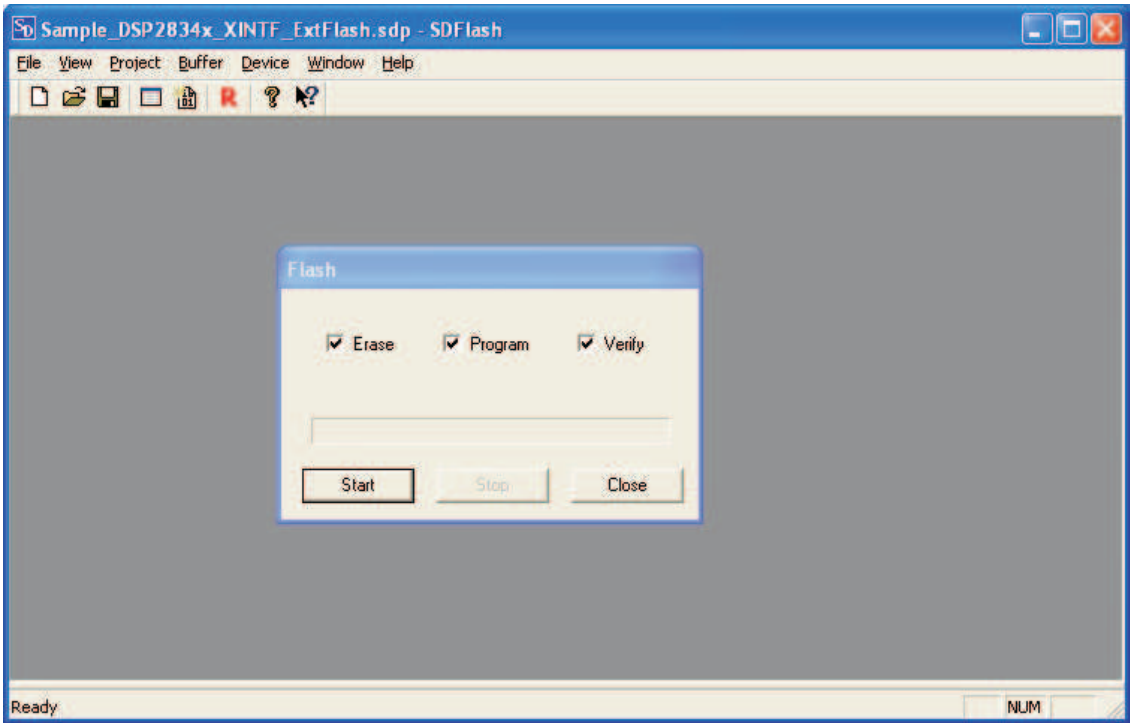
Figure 6. Verify

The algorithm file for the provided sample project is `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_XINTF_SDFlash\Debug\DSP2834x_XINTF_SDFlash.out`.

The Flash data file for the provided sample project is `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_GpioToggle_ExtFlash\Debug\GpioFlash.out`.

Once you have configured your project, click on Device → Flash.

This will bring up a window with three check boxes labeled *Erase*, *Program*, and *Verify*. Select which of these operations you want to perform and then press *Start* to continue. The operations are executed from left to right. The device contains the correct data in the external Flash memory once the operations have been performed.



5 Writing Drivers for Your Memory

5.1 Low-Level Drivers

First, you need to write drivers to interface directly to your intended external memory through your intended communication channel (XINTF, SPI, I2C…etc). These drivers need to be able to configure the C28x peripheral used to communicate with the memory; they need to be able to read data from the memory, write data to the memory, and erase the memory; and they need to be able to perform any other operations on the external memory device necessary for your application (set different modes, lock and unlock, reset…etc).

To write these drivers you need Code Composer Studio and a JTAG emulator. Use these tools to develop and test your low-level drivers separately from SDFlash. Once the drivers are working, they can later be integrated into the SDFlash algorithm, where their functions will be called by the functions in the SDFlash wrapper. Note that SDFlash is in no way required to write the low-level drivers; design your drivers to meet the needs of your end application.

Since there are so many choices of external memories, this guide will not try to develop a generic solution to low-level interfacing to an external memory. Moving forward, it is assumed that the low-level drivers for the external memory are in a working condition and capable of basic operations on the external memory. However, an example of a low-level external memory driver, DSP2834x_AT49BV802D.c, is included in this download. This C2834x driver is used to connect through the XINTF to an 8M-bit parallel Flash memory produced by Atmel (AT49BV802D). It can be found in the folder `<ProjectDir>\tif2834x_XINTF_flashv1\common\source\`.

Feel free to modify this example driver as necessary for your application or use it as a template for your own drivers. Note that the requirements for this driver were to be able to write to the external Flash memory to preload program code and then to execute directly from external Flash through the XINTF. Since write operations were not intended to occur during application execution, calls to the write functions block execution until the entire write is complete. If your application intends to log data to the external memory during application program execution, this may not be a good solution. It may be necessary to structure your drivers to include function callbacks using pointers-to-functions or to include non-blocking functions with periodic status checks from the calling program.

5.2 SDFlash Algorithm

The SDFlash algorithm is a C28x program loaded by SDFlash onto the C28x device. Then, SDFlash executes the functions in the SDFlash wrapper, which in turn, call the low-level drivers that program the external memory.

5.2.1 SDFlash Wrapper

The SDFlash wrapper defines specific functions and variables that the SDFlash application expects to be in the algorithm. These functions and variables are the only way the SDFlash application interacts with the C28x device. The following functions should be implemented in the algorithm project.

Function Name	Description
void PRG_init(void)	This function is called by the SDFlash application before attempting to erase, program, and verify. This is where to initialize the clock and PLL settings and to call the low-level drivers that setup the XINTF, SPI, or I2C .
void PRG_erase(void)	This function is called by the SDFlash application if the *erase* box was checked. The low-level drivers to erase the external memory should be called in this function. Erasing the memory before programming may not be necessary for all types of nonvolatile memory.
void PRG_program(void)	This function is called by the SDFlash application if the *program* box was checked. SDFlash populates a data buffer variable, a start address variable, and a data length variable to specify what data should be programmed and to where. This function should call the low-level program drivers with this information. Depending on the length of the data buffer and the number of disjoint sections of memory to be programmed, SDFlash will repeatedly call this function to program chunks of memory until all sections have been programmed.
void PRG_verify(void)	This function is called by the SDFlash application if the *verify* box was checked. Whereas PRG_program writes data to the external memory, PRG_verify reads the data already in the external memory and compares it to the expected contents. SDFlash populates a data buffer variable, a start address variable, and a data length variable to specify what data should be expected and where to expect it. This function should call the low-level verify drivers with this information. This function is repeatedly called by the SDFlash application to verify chunks of memory until all memory has been verified.
void PRG_exit(void)	This function passes control from the running C28x code back to the SDFlash application by means of a software breakpoint.

The following variables are used to pass data back and forth between the algorithm and the SDFlash application:

Function Name	Description
volatile Uint16 *PRG_bufaddr	Pointer to the data buffer (that contains data to be programmed to external memory or data to be used for verification). This variable is set by the algorithm during PRG_init to let the SDFlash application know where the data buffer is.
volatile Uint16 PRG_bufsize	Size of the data buffer. This is set during PRG_init by the algorithm to let the SDFlash application know how large the data buffer is.
volatile Uint16 PRG_devsize	Size of the external memory. Set by the algorithm.
volatile Uint16 *PRG_paddr	Programming address. This pointer is set by SDFlash prior to running PRG_program or PRG_verify to let the algorithm know where to start programming the data in the data buffer to or where to start reading data from for verification.
volatile Uint16 PRG_page	Which page data should be written to for programming or read from for verification. Set by the SDFlash application.
volatile Uint16 PRG_length	The number of words placed in the data buffer. Set by the SDFlash application prior to running PRG_program or PRG_verify to let the algorithm know how many words to program or verify.
volatile Uint16 PRG_status	Status of the algorithm. Set by algorithm and read by SDFlash.
volatile Uint16 PRG_options	User defined options entered in GUI of the SDFlash application. This is a 16 bit hexadecimal value typed into the SDFlash application's GUI under a particular tab: erase, program, or verify. Can be used to mask sectors to be erased, program a pattern instead of the .out file data for testing, to verify only certain sectors…etc.
volatile Uint16 PRG_options2	More options
volatile Uint16 PRG_options3	More options
volatile Uint16 PRG_options4	More options

5.2.2 SDFlash Algorithm Flow

The following examples demonstrate how the SDFlash application, running on the PC, programs the external memory that is connected to the C28x device.

Initializations: This is the first thing that happens before erase, program, or verify is attempted. The SDFlash application:

1. Loads the SDFlash algorithm, that includes the SDFlash wrapper and low-level drivers, onto the C28x device via the JTAG. It then sets the C28x program counter (PC) to the location of PRG_int() and starts the C28x running. The PRG_init does various initializations to variables in the SDFlash wrapper (e.g. *PRG_bufaddr and PRG_bufsize). Then, the PRG_init:
2. Calls the initialization function(s) of the low-level drivers for the external memory. The low-level drivers:
3. Configure the peripheral used to communicate with the external memory device and perform any other initializations required for the external memory drivers. Execution returns to PRG_init. The PRG_init:
4. Sets PRG_status as appropriate to indicate success or failure of the initializations and then calls PRG_exit to end initializations. PRG_exit:
5. Cleans up and executes a software breakpoint that lets the SDFlash application know that PRG_init is complete. The SDFlash application checks the value of PRG_status to ensure that no errors have occurred. It then moves on to executing erase, program, and/or verify, depending on what the SDFlash user has selected.

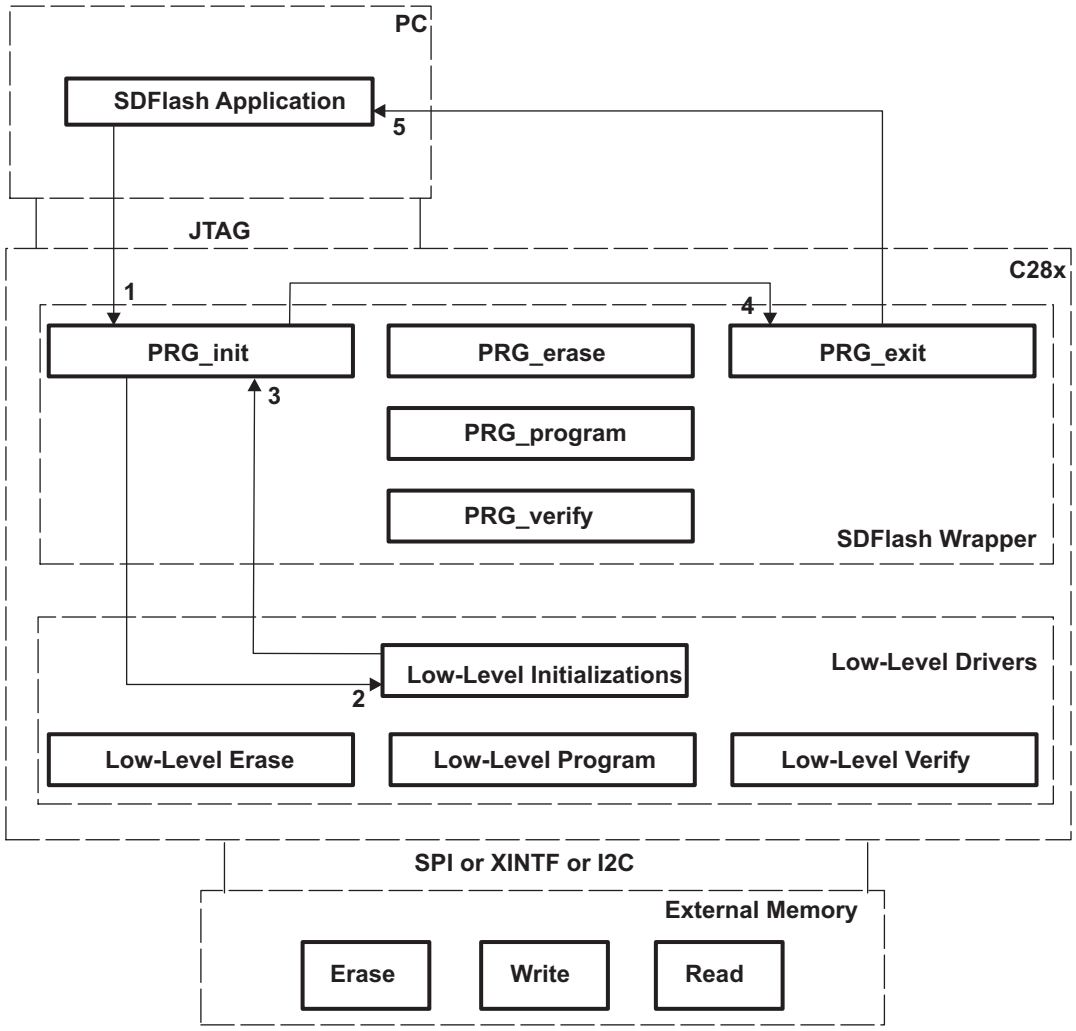


Figure 7. Initializations

Erase: Now that the device is initialized SDFlash will attempt to erase the external memory, if you selected this option. If you did not select to erase memory, SDFlash moves on to program and then to verify. Algorithm code is already loaded on the C28x from initializations and is sitting at a software breakpoint. SDFlash:

1. Sets the C28x PC to the beginning of PRG_erase. SDFlash also populates the PRG_options variables with whatever you have entered. SDFlash starts execution of PRG_erase. PRG_erase:
2. Calls the erase functions of the low-level external memory drivers using the information in the PRG_options variables as appropriate (e.g., to tell the low-level drivers to only erase certain sectors). The low-level drivers:
3. Communicate with the external memory device through a communication peripheral (e.g., XINTF) issuing erase commands to the external memory device. The external device:
4. Indicates the status of the erase operation. Then, external memory drivers:
5. Return the status of the erase operation to PRG_erase. PRG_erase:
6. Sets PRG_status as appropriate to indicate whether the erase operation was successful or not. PRG_erase then calls PRG_exit to return control to SDFlash. PRG_exit:
7. Cleans up and executes a software breakpoint that lets the SDFlash application know that PRG_erase is complete. The SDFlash application checks the value of PRG_status to ensure that no errors have occurred. It then moves on to executing program and/or verify, depending on what the SDFlash user has selected.

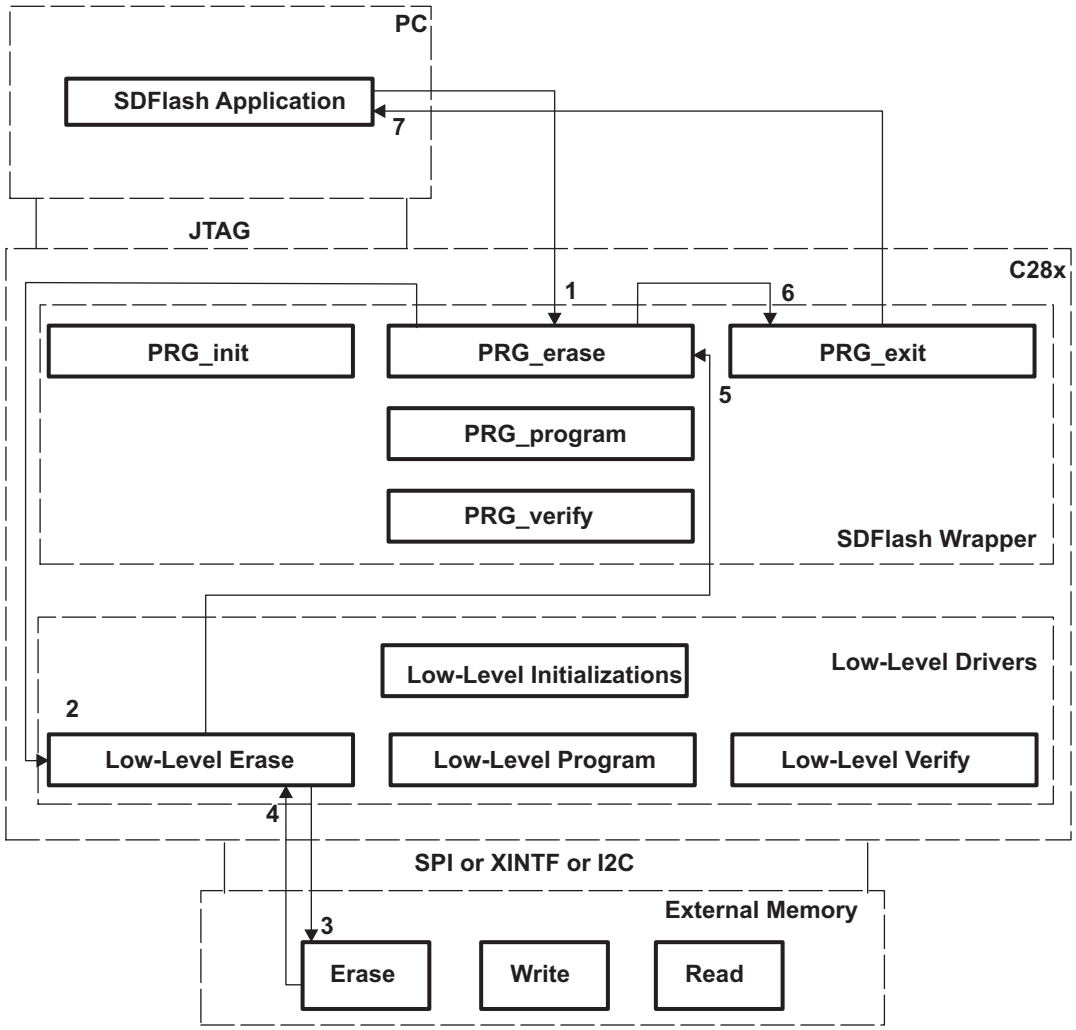


Figure 8. Erase

Program: Now that the external memory has been erased if necessary, data can be programmed into the device. If you have not selected the program option, SDFlash moves on to verify. Algorithm code is already loaded on the C28x from previous operations and is sitting at a software breakpoint. SDFlash:

1. Sets the C28x PC to the beginning of PRG_program. SDFlash also populates the PRG_options variables with whatever you have entered. SDFlash populates the buffer specified by *PRG_bufaddr with data to be programmed to the external memory and also sets *PRG_paddr with the desired location to program this data. PRG_length is set to indicate how many words were placed in the buffer. SDFlash starts execution of PRG_program. PRG_program:
2. Uses the information in the PRG_options variables along with the data in the data buffer, *PRG_paddr and PRG_length to appropriately call the low-level drivers for programming. The low-level drivers:
3. Communicate with the external memory device through the communication peripheral causing data to be written to the memory. Memory:
4. Returns the status of the operation to the low-level drivers. Then, the low-level drivers:
5. Return the status of the programming operation to PRG_program. PRG_program:
6. Sets PRG_status as appropriate to indicate if the programming operation was successful or not. PRG_program then calls PRG_exit to return control to the SDFlash application. PRG_exit:
7. Cleans up and executes a software breakpoint that lets the SDFlash application know that PRG_program is complete. The SDFlash application checks the value of PRG_status to ensure that no errors have occurred. It then moves on to execute verify if you have selected it.

NOTE: Depending on how data is passed from PRG_program to the low-level drivers, some subset of the above examples are repeated for each word in the data buffer. If PRG_program passes single words to low-level drivers, then steps 2,3,4 and 5 are repeated. If PRG_program passes the entire data buffer to the low-level drivers, only steps 3 and 4 are repeated for each word.

NOTE: When SDFlash calls PRG_program, one contiguous chunk of memory with size less than or equal to the size of the data buffer is programmed into the external memory. SDFlash repeatedly calls PRG_program with different data buffer contents until all desired memory locations are programmed.

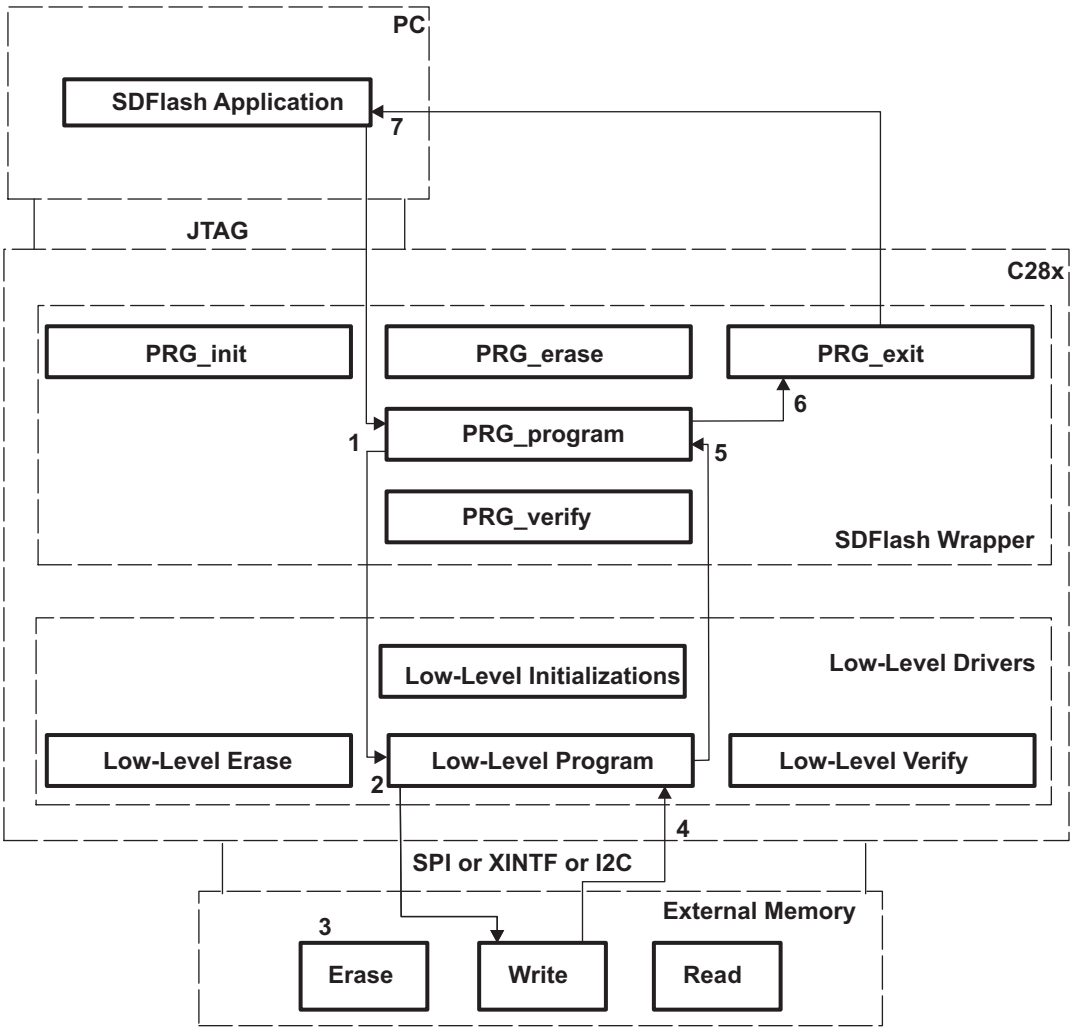


Figure 9. Program Flow

Verify: Verify is executed after program if you have selected it in the SDFlash application GUI. Verify ensures that data has been correctly programmed into the external memory. Algorithm code is already loaded on the C28x from previous operations and is sitting at a software breakpoint. SDFlash:

1. Sets the C28x PC to the beginning of PRG_verify. SDFlash also populates the PRG_options variables with whatever you have entered. SDFlash populates the buffer specified by *PRG_bufaddr with data to be used for verification against the external memory and also sets *PRG_paddr with the desired location in external memory to verify. PRG_length is set to indicate how many words were placed in the buffer. SDFlash starts execution of PRG_verify. PRG_verify:
2. Uses the information in the PRG_options variables along with the data in the data buffer, *PRG_paddr and PRG_length to appropriately call the low-level drivers for reading/verification. The low-level drivers:
3. Issue read commands to the external memory. Then, memory:
4. Returns the data at the specified location. The low-level drivers:
5. Compare what is in the memory with what is expected and return the results to PRG_verify. PRG_verify:
6. Sets PRG_status as appropriate to indicate if the verification operation was successful or not. PRG_verify then calls PRG_exit to return control to the SDFlash application. PRG_exit:
7. Cleans up and executes a software breakpoint that lets the SDFlash application know that PRG_verify is complete. The SDFlash application checks the value of PRG_status to ensure that no errors have occurred.

NOTE: Depending on how data is passed from PRG_verify to the low-level drivers some subset of the above steps are repeated for each word in the data buffer. If PRG_verify passes single words to low-level drivers, then steps 2,3,4 and 5 are repeated. If PRG_verify passes the entire data buffer to the low-level drivers, only steps 3 and 4 are repeated for each word.

NOTE: When SDFlash calls PRG_verify, one contiguous chunk of memory with size less than or equal to the size of the data buffer is checked against the external memory. SDFlash will repeatedly call PRG_verify with different data buffer contents until all desired memory locations are verified.

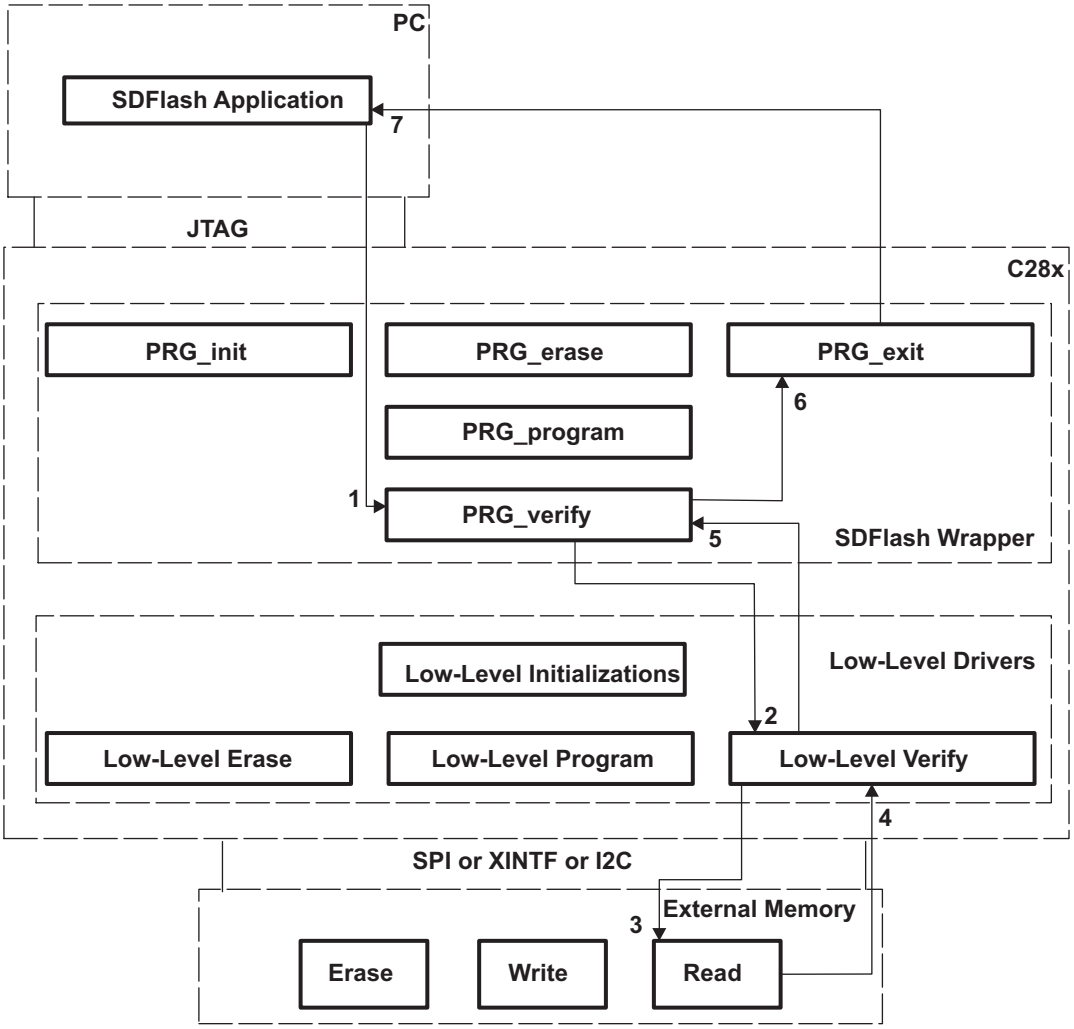


Figure 10. Verify Flow

5.2.3 Building Your SDFlash Algorithm

Once you have the low-level drivers for your external Flash memory working, you can build the SDFlash algorithm that uses those low-level drivers. This is done inside of Code Composer Studio. To build your own SDFlash algorithm, you need to implement the functions and define the variables specified in the SDFlash wrapper. Included with this download is a header file that defines all the functions and variables that you need. This header file can be found at `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_XINTF_SDFlash\include\SDFlash2834x_Wrapper.h`.

In addition to the included SDFlash wrapper header file, this download contains a full working SDFlash algorithm project for Code Composer Studio v 3.3. This project provides everything necessary to download code into an Atmel 8M-bit parallel Flash memory (AT49BV802D) using the XINTF of the C2834x. This project can be found in `<ProjectDir>\tif2834x_XINTF_flashv1\DSP2834x_XINTF_SDFlash\`.

Use this project as a template for building your SDFlash algorithm to do in place programming of your external memory. Modify the contents of PRG_init (defined as an assembly function that branches to main(), so modify the main function) to setup your external memory device. Modify PRG_erase, PRG_program, and PRG_verify to call the appropriate drivers for your external memory device.

In the end you will have a Code Composer Studio project that contains several things:

- A header file that defines the functions and variables in the SDFlash wrapper
- A source file or files that implements the functions in the SDFlash wrapper
- Header file(s) and source file(s) for your low-level external memory drivers
- Additional header and source files for your peripheral (e.g., DSP2834x_Xintf.c)
- Optional – Files that create additional layers of abstraction between SDFlash wrapper and low-level memory drivers (see next section)

All of these things are written using Code Composer Studio.

Now that you have all the files for your SDFlash algorithm, you can begin to test and debug them. First compile your algorithm project in Code Composer Studio and then close Code Composer Studio (Code Composer Studio and SDFlash cannot be open at the same time or they will compete for use of the JTAG port). Next configure SDFlash to use the .out file from your algorithm as the *Algorithm File* in the *Erase*, *Programming*, and *Verify* tabs (see Section 3). In the *Programming* tab set the *Flash Data File* field to be whatever you want to program into external memory (your application code linked to external memory or a test program linked to external memory). Now, check which of the erase, program, and/or verify operations you want to perform and press *Start* to continue (see Section 3). The SDFlash application will download and run your algorithm code.

Now that the algorithm code has been run, close SDFlash and open Code Composer Studio while maintaining power to the C28x device. Connect to the C28x device and load the algorithm .out file to the C28x device (but do not run the algorithm). Now, use watch windows and memory windows to observe the results produced when SDFlash ran the algorithm.

You may want to test and debug within Code Composer Studio If you see that the memory and registers are not sufficient to debug the algorithms, after running SDFlash. Load the algorithm .out file to your C28x device using Code Composer Studio and then use the *Set PC to Cursor* command to run arbitrary sections of code (e.g., PRG_init or PRG_program). Manually set the SDFlash wrapper variables before running to emulate the actions of the SDFlash application. You can also set _c_int00, the starting point of code when using Code Composer Studio, to point to a function like PRG_init, and then add calls from PRG_init to other PRG_ functions.

5.3 Abstraction Layer

The example SDFlash algorithm uses an abstraction layer to allow different low-level drivers to be interchanged without modifying the SDFlash wrapper source code. This is accomplished by defining initialization, erase, program, and verify functions which are called from the SDFlash wrapper and must be implemented either in the low-level drivers for a particular memory or in a file that, in turn, calls the drivers for a particular memory. In this way code pertaining to the wrapper is kept inside the wrapper and code pertaining to the external memory drivers is completely outside the SDFlash wrapper.

The functions of this abstract interface are defined in the file `<ProjectDir>\tif2834x_XINTF_flashv1\DSP2834x_headers\include\DSP2834x_ExtFlashAPI.h` and are implemented directly in the external memory drivers in `<ProjectDir>\tif2834x_XINTF_flashv1\DSP2834x_common\isource\AT49BV802D.c`.

If you implement the functions found in ExtFlashAPI.h in your external memory drivers, you will not have to make changes to the SDFlash wrapper (DSP2834x_XINTF_SDFlash.c, DSP2834x_XINTF_SDFlash_boot.asm, and SDFlash2834x_Wrapper.h) except to change the include statements to import your drivers into the file.

The abstract interface also allows you to swap out your external memory device drivers for debug code. If desired, implement the functions defined in the abstract interface in a separate debug driver file. If you include this file in the SDFlash project instead of your memory drivers, you will be able to do things like toggle a general-purpose input/output (GPIO) to insure your drivers are getting called or to print out what is being passed as parameters to the drivers.

6 Example System: AT49BV802D 8M-Bit Memory and Modified C28346 Control Card

6.1 Overview

The following sections outline a complete design for attaching an external nonvolatile memory to a C28346 device. The external memory used is an 8M-bit parallel Flash memory produced by Atmel (AT49BV802D). The board used is a modified version of the C28346 DIMM-168 control card base. Low level drivers for the external Flash memory are provided as well as an SDFlash algorithm that uses them.

The example board was designed using ExpressSCH and ExpressPCB, that are available for free from www.expresspcb.com. The design is based off of the design for the C28346 control card and docking station, which is available from www.ti.com/f28xkits under *baseline software* for the C28346 control card and docking station, see *C2000 Baseline Software for ControlCARD Kits* (SPRC675).

6.2 Schematic

This circuit removes some of the GPIO headers on the DIMM-168 control card docking station and adds the AT49BV802D Flash memory onto zone 6 of the XINTF. A 16-bit wide data bus is used.

- XA1 to XA20 are connected to A0 to A18 on the external memory device
- XD0 to XD15 is connected to the I/O0 to I/O15 pins on the external memory device
- XZCS6n is connected to CE on the external memory device
- XRDn is connected to OE on the external memory device
- XWE0n is connected to WE on the external memory device

The schematic files included in this download are named *DIM168FlashDockingStn_Schematic.sch* (ExpressSCH version) or *DIM168FlashDockingStn_Schematic.pdf* (portable document version).

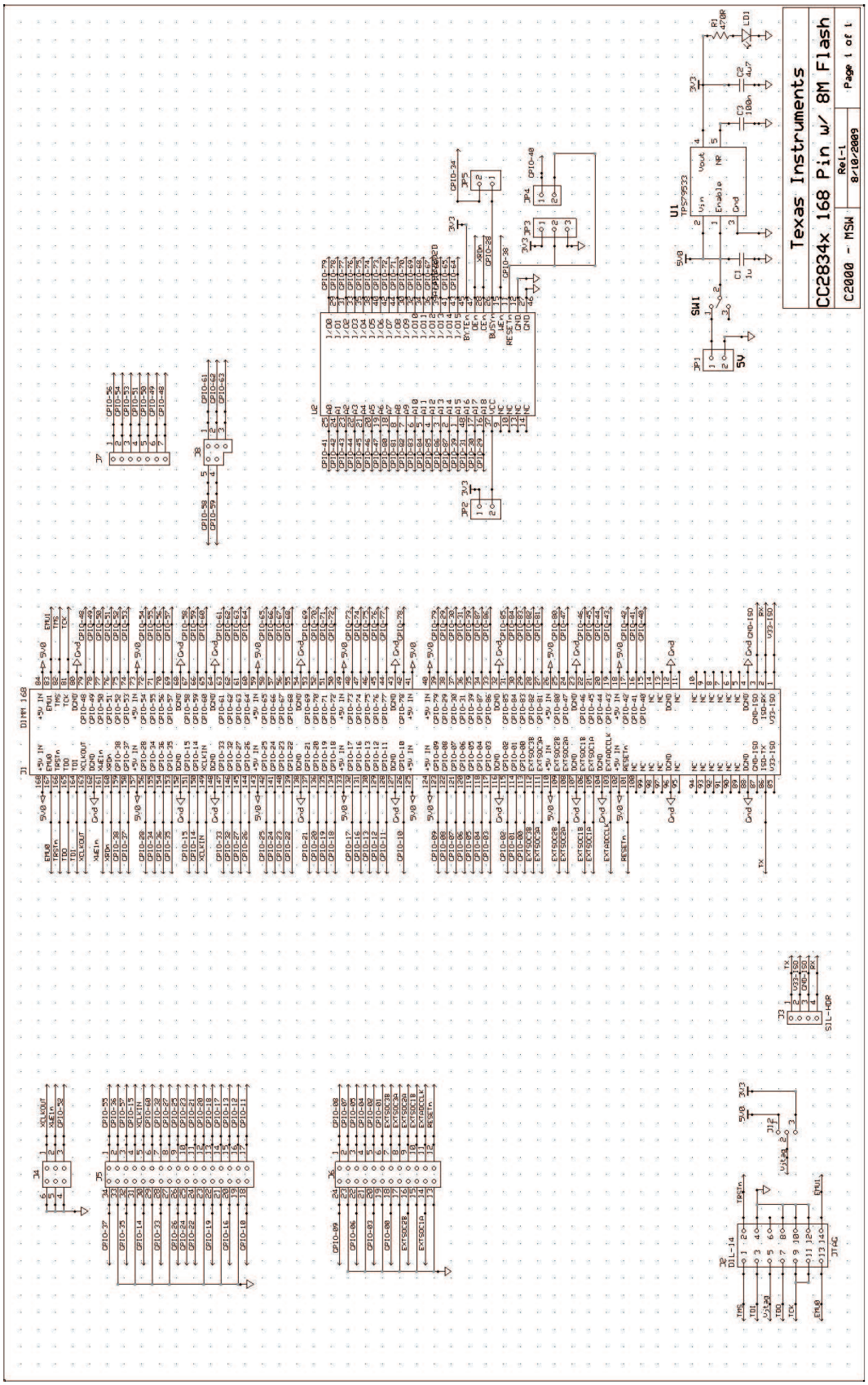


Figure 11. Schematic

6.3 Layout

The layout files included in this download are named *DIM168FlashDockingStn_Layout.pcb* (ExpressPCB version) or *DIM168FlashDockingStn_Layout.pdf* (portable document version). This express PCB file is everything necessary to have ExpressPCB manufacture the board, if desired. For more information on how to have a board manufactured, see the ExpressPCB website at: www.expressPCB.com.

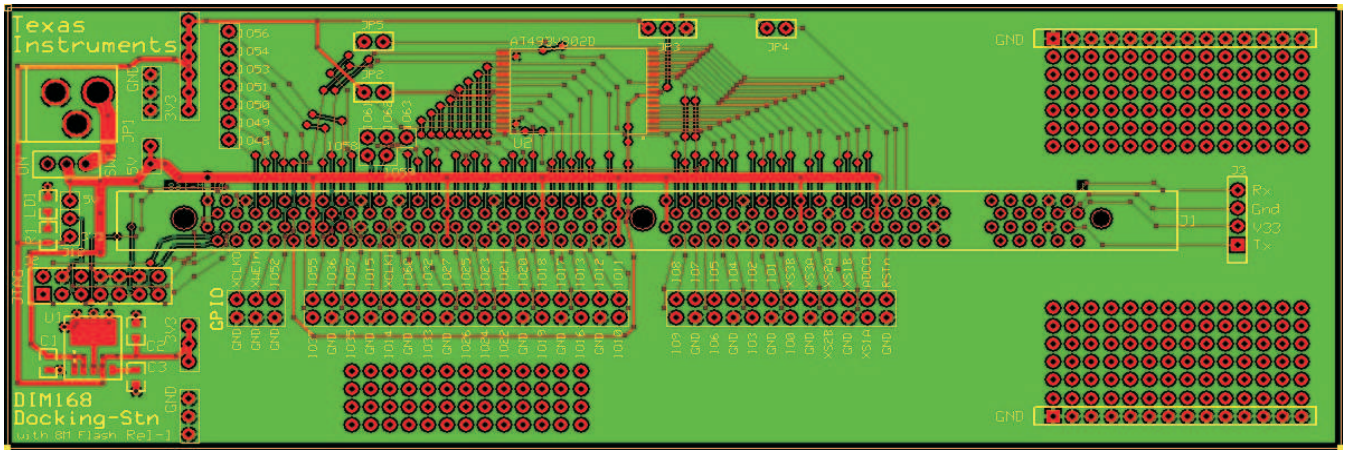


Figure 12. Layout Illustration of Board

6.4 AT49BV802D Drivers

The source code for the C2834x drivers for the AT49BV802D external Flash memory are located in the file `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_common\source\DSP2834x_AT49BV802D.c`. This source file uses the header file `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_headers\include\DSP2834x_AT49BV802D.h` and the header file for abstract external Flash interface at `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_headers\include\DSP2834x_ExtFlashAPI.h`.

These drivers setup the XINTF and configure the GPIO mux for x16 XINTF operation. The drivers perform operations by writing command sequences to the external memory device (these command sequences can be found in the data sheet for the external Flash memory on the manufacturer's website).

6.5 SDFlash Algorithms

The SDFlash Algorithms are composed of several files:

- Low-level drivers discussed in the previous section
- SDFlash2834x_Wrapper.h. This file defines all the functions and variables that must be implemented in the SDFlash wrapper. Located in the `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_XINTF_SDFlash\include\` directory
- DSP2834x_XINTF_SDFlash.c. This is the main file that implements the SDFlash wrapper. Note that the main() function is effectively PRG_init(). This file is located in `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_XINTF_SDFlash\source\`"
- DSP2834x_XINTF_SDFlash_Boot.asm. This assembly file contains the entry point for the program – PRG_init. This entry point calls main() in DSP2834x_XINTF_SDFlash.c This file also contains the PRG_exit function.

These files are all part of a Code Composer Studio v3.3 project *DSP2834x_XINTF_SDFlash.pjt* that can be found in `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_XINTF_SDFlash\`.

6.6 GPIO Toggle Project

In addition to the SDFlash algorithm project that is provided, there is also an example project that has been designed to run from external memory. This project links into zone 6 of the XINTF and can be run directly from external memory through the XINTF. The project causes the GPIO pins to toggle. Use this project as an example of what SDFlash would be used to load to the external memory and how to port your application to run from external memory.

The example project, *GpioFlash.pjt* can be found in the directory `<ProjectDir>\tif2834x_XINTF_flash_v1\DSP2834x_GpioToggle_ExtFlash`.

For a much more thorough discussion of porting your application to run from nonvolatile memory, see *Running an Application from Internal Flash Memory on the TMS320F28xxx DSP* (SPRA958).

6.7 More Examples

SDFlash algorithms for various devices can be downloaded from Spectrum Digital's website: http://emulators.spectrumdigital.com/utilities/sdflash/. Of particular interest is *C2834x SDFlash Algos V1.0 for programming external SPI Serial EEPROM or Flash connected via SPI-A*. These algorithms serve as additional examples of programming external memory devices; in this case, through the SPI instead of the XINTF. Example algorithms are provided for both Flash memories and EEPROM in this download.

The parallel Flash programming algorithms provided with this download are also available from Spectrum Digital's website, http://emulators.spectrumdigital.com/utilities/sdflash/, under the heading *C2834x SDFlash Algos V1.0 for programming external Flash connected via XINTF*.

7 References

- *Running an Application from Internal Flash Memory on the TMS320F28xxx DSP* (SPRA958)
- *C2000 Baseline Software for ControlCARD Kits* (SPRC675)

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
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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