Application Note

CAN Flash Programming of C2000 Microcontrollers

Charles Roberson, Vamsikrishna Gudivada, and Skyler Baumer

ABSTRACT

Embedded processors often need to be programmed in situations where a JTAG debug probe cannot be used to program the target device. In these cases, the engineer needs to rely on programming solutions leveraging peripherals such as Controller Area Network (CAN) or Controller Area Network Flexible Data-Rate (CAN-FD), also known as Modular Controller Area Network (MCAN). C2000™ devices aid in this endeavor through the inclusion of several program loading utilities in Boot-ROM. These utilities are useful, but only solve half of the programming problem because they only facilitate loading application code into RAM. This application note builds on these ROM loaders by using a flash kernel. A flash kernel is loaded to RAM using a ROM loader - it is then executed and used to program the target device's on-chip Flash memory with the end application. This document details possible implementations for C2000 devices and provides PC utilities to evaluate the solution with.

Table of Contents

List of Figures

List of Tables

Trademarks

1 Introduction

The CAN flash kernel facilitates firmware updates of any C2000 MCU that features the CAN peripheral. It is copied into the device's RAM using the CAN bootloader in Boot-ROM. The kernel will then use the CAN peripheral to transfer the firmware from the host and program it in the Flash memory. The flash kernel consists of two pieces of software – a PC host CAN programmer and a Code Composer Studio™ (CCS) project for the C2000 MCU. The F28003x device is used as the basis for these projects and can be modified to work with other devices. If this project is to be used on any device, parameters related to the device operating frequency may need to be modified.

In summary, programming the flash requires two steps:

1. Use the CAN ROM bootloader to download the flash kernel to RAM.
2. Run the flash kernel in RAM to download the application to flash.

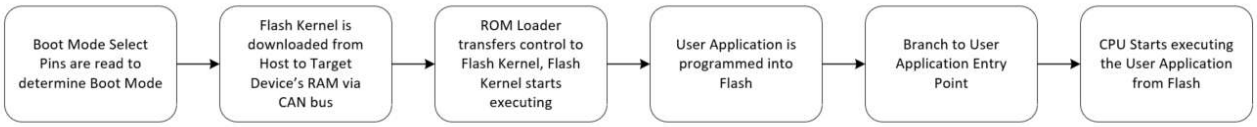

Figure 1-1. CAN Boot Flow

2 Programming Fundamentals

Before programming a device, it is necessary to understand how the non-volatile memory of C2000 devices works. Flash is a non-volatile memory that allows users to easily erase and re-program it. Erase operations set all the bits in a sector to '1' while programming operations selectively clear bits to '0'. Flash on certain devices can only be erased one sector at a time, but others have bank erase options.

Flash operations on all C2000 devices are performed using the CPU. Algorithms are loaded into RAM and executed by the CPU to perform any flash operation. For example, erasing or programming the flash of a C2000 device with Code Composer Studio™ entails loading flash algorithms into RAM and letting the processor execute them. There are no special JTAG commands that are used. All flash operations are performed using the flash application programming interface (API). Because all flash operations are done using the CPU, there are many possibilities for device programming. Irrespective of how the kernels and application are brought into the device, flash is programmed using the CPU.

Note

The term DCAN used in this report - DCAN flash kernels, DCAN flash programmer, and so forth refer to the Controller Area Network communication interface (CAN) [8]. **The DCAN flash programmer described in this document refers to the CAN module.**

The term MCAN used in this report - MCAN flash kernels, CAN flash programmer, and so forth refer to the Modular Controller Area Network (MCAN). MCAN is an interchangeable term with Controller Area Network Flexible Data-Rate (CAN-FD) [9]. **The CAN flash programmer described in this document refers to the MCAN module.**

In future releases, the naming convention for DCAN and MCAN flash projects will be made consistent for their related usage.

3 ROM Bootloader and Hex Utility Usage

At the beginning of device startup, the device boots and, based on the boot mode, decides if it should execute code already programmed into the Flash memory or load in code using one of the ROM loaders. This application note focuses on the boot execution path when the emulator (CCS) is not connected.

Note

This section is based on the TMS320F28003x device. Specific information for a particular device can be found in the *Boot ROM* chapter of the device-specific technical reference manual (TRM).

Table 3-1. Default Boot Modes for F28003x Devices

Boot Mode	GPIO24 (default boot mode select pin 1)	GPIO32 (default boot mode select pin 0)
Parallel I/O	0	0
SCI/Wait boot	0	1
CAN	1	0
Flash	1	1

After the boot ROM readies the device for use, it decides where it should start executing (flash or Boot ROM). In the case of a standalone boot, it does this by examining the state of two GPIOs (as seen in Table 3-1, the default choices are GPIO 24 and 32). In some cases, two values programmed into one time programmable (OTP) memory can be examined. In the implementations described in this application note, both the CAN and MCAN loader are used, so at power up GPIO 32 must be forced low and GPIO 24 must be forced high. If this is the case when the device boots, the CAN loader in ROM begins executing and waits to receive data from the host.

For CAN Boot mode (DCAN), the boot table for GPIO assignments is defined by Table 3-2. Table 3-3 shows the MCAN Boot Options. Note that the boot options for DCAN and MCAN use the same GPIO assignments to select the boot definition value. The common GPIO pairs are GPIO4 and GPIO5, as well as GPIO13 and GPIO12. For more information, read the boot options in the device-specific TRM.

Table 3-2. CAN Boot Options

Option	BOOTDEF Value	CANTXA GPIO	CANRXA GPIO
0 (default)	0x02	GPIO4	GPIO5
1	0x22	GPIO32	GPIO33
2	0x42	GPIO2	GPIO3
3	0x62	GPIO13	GPIO12

Table 3-3. MCAN Boot Options

Option	BOOTDEFx Value	CANTXA GPIO	CANRXA GPIO
0	0x08	GPIO4	GPIO5
1	0x28	GPIO1	GPIO0
2	0x48	GPIO13	GPIO12

The ROM loader requires data to be presented to it in a specific structure. The structure is common to all ROM loaders and is described in detail in the *Bootloader Data Stream Structure* section of [6] . You can easily generate your application in this format by using the hex2000 utility included with the TI C2000 compiler. This file format can even be generated as part of the Code Composer Studio build process by adding a post-build step with the following options:

```
"${CG_TOOL_HEX}" "${BuildArtifactFileName}" -boot -sci8 -a -o "${BuildArtifactFileBaseName}.txt"
```

Alternatively, you can use the TI hex2000 utility to convert COFF and EABI .out files into the correct boot hex format. To do this, you need to enable the C2000 Hex Utility under Project Properties. The command is below:

```
hex2000.exe -boot -sci8 -a -o <file.txt> <file.out>
```

As stated before, ROM loaders can only load code into RAM, which is why they are used to load in flash kernels, which will be described in DCAN Flash Kernel and MCAN Flash Kernel sections.

The flash kernel expects the firmware image to be in the same format, so the command above can be used for the firmware image hex file generation as well.

The MCAN ROM bootloader has three boot options and each option is mapped to different GPIO pins to be used for MCANRX and MCANTX functions. In order to configure the device for MCAN Boot Mode, the OTP of the device needs to be programmed to include MCAN Boot. For more details on how to program the one time programmable (OTP), see the *Boot ROM* chapter of the *TMS320F28003x Real-Time Microcontrollers Technical Reference Manual* (TRM).

4 DCAN Flash Kernel

DCAN Flash Kernel runs on:

- TMS320F28003x
- TMS320F28P65x
- TMS320F280015x

To find the location of the flash kernel project for these devices, see Troubleshooting.

4.1 Implementation

DCAN Flash Kernel is based off DCAN ROM loader sources. To enable this code to erase and program flash, flash APIs must be incorporated, which is done by linking the flash APIs. Before any application data is received, the F28P65x and F280015x (not the F28003x, this is done later) DCAN flash kernels erase the flash of the device readying it for programming. The F28P65x and F280015x DCAN flash kernel projects allow the user to specify which flash banks and flash sectors should be erased before the application is programmed. This is discussed in more detail in Custom Flash Bank and Sector Erase. After the appropriate locations in flash memory are erased, the application load begins. A buffer is used to hold the received contiguous blocks of application code. When the buffer is full or a new block of non-contiguous data is detected, the code in the buffer is programmed. This continues until the entire application is received.

After the DCAN module is initialized in the flash kernel, this module waits for the host to send in the firmware image. The flash kernel receives 8 bytes at a time from the host and places the contents into an intermediate RAM buffer. This buffer is then written into Flash in 128-bit or 512-bit increments. The F28P65x and F280015x DCAN flash kernel projects support 512-bit programming, while the F28003x uses 128-bit programming. If desired, there is also a 128-bit programming project available for the F28P65x device. The F280015x flash API supports 128-bit programming, but the flash kernel was implemented using 512-bit programming.

Before writing to a sector for the first time, the F28003x (not the F28P65x and F280015x, as flash is erased earlier in the process) Flash kernel checks to see if the sector has been erased, and if it has not been erased, the F28003x Flash kernel has the Flash API execute an erase operation. After this, a buffer is filled up with content to be written into Flash, and a program command is sent from the Flash API. Once the write has occurred, the Flash kernel has the Flash API verify that the segment was written into Flash at the correct address. Once the kernel has copied everything to Flash, the project jumps to the entry address of the image.

All of the sections of the firmware image stored in flash should be aligned according to the number of bits being programmed at once. If programming 128-bits at once (F28003x and F28P65x), the sections should be aligned to a 128-bit boundary. In the linker command file for the firmware image, all initialized sections need to be mapped to Flash sectors, and after each mapping, an ALIGN(8) directive needs to be added to ensure the 128-bit alignment. If programming 512-bits at once (F280015x and F28P65x), the sections should be aligned to a 512-bit boundary. In the linker command file for the firmware image, all initialized sections need to be mapped to Flash sectors, and after each mapping, an ALIGN(32) directive needs to be added to ensure the 512-bit alignment.

The protocol used to transfer the application data has been slightly modified from the DCAN ROM loader protocol. With the original DCAN ROM loader protocol, data is transmitted two bytes per frame at 100Kbps from the host to the target device. As the data is transmitted two bytes per frame at a time to the DCAN ROM loader, this increases the total time to download the kernel or image. By modifying the data length code (DLC) per frame to 8 bytes and increasing the bitrate to 1Mbps, this allows the PC side application to send many bytes at a time through the host, substantially decreasing the latency of communications. The changes required to increase the bitrate to 1Mbps are detailed in the Application Load section.

4.1.1 Custom Flash Bank and Sector Erase

The F280015x and F28P65x (both the 128-bit and 512-bit programming projects) DCAN flash kernels allow the user to specify which flash banks and flash sectors should be erased before the application is loaded to flash. This section will discuss how it is implemented for the F28P65x device, but the F280015x implementation is very similar.

Within the flash_kernel_ex5_can_flash_kernel.c file, there are four 32-bit unsigned integer arrays that control the flash banks and sectors erased by the kernel.

```
uint32_t Application_Flash_Banks[5] = {0,1,2,3,4};
uint32_t WE_Protection_A_Masks[5] = {0,0,0,0,0};
uint32_t WE_Protection_B_Masks[5] = {0,0,0,0,0};
uint32_t WE_Protection_OTP_Masks[5] = {0,0,0,0,0};
```

Within Application_Flash_Banks, the flash bank numbers should be entered. Within WE_Protection_A_Masks, the Write/Erase Protection Masks corresponding the sectors 0-31 for each bank in Application_Flash_Banks should be entered. As shown above, flash sectors 0-31 of each bank will be erased. Within WE_Protection_B_Masks, the Write/Erase Protection Masks corresponding the sectors 32-127 for each bank in Application_Flash_Banks should be entered. As shown, flash sectors 32-127 will be erased in each bank. For more information about these masks, see the device-specific Flash API Guide. Similarly, WE_Protection_OTP_Masks should be filled with the desired masks for each bank's OTP. Note that configuring this array will not result in OTP being erased, but rather enable the device to program a flash bank's OTP during the application load.

Additionally, WE_Protection_A_Masks and WE_Protection_B_Masks is used when programming the application to flash. The proper masks are determined based on the target address of the data being programmed.

Once these arrays are configured as desired, recompile the flash kernel and generate the new kernel image.

4.1.2 Application Load

This section walks through the entire flow of programming an application into flash using the DCAN boot mode.

Ensure the device is ready for DCAN communications by resetting the device while ensuring the boot mode pins are in the proper state to select DCAN Boot mode. These are the steps that follow:

1. The device enters the DCAN Boot loader and waits to receive message frames in Mailbox 1. Acceptable messages have a message identifier (MSGID) value of 0x1 for boot-loader communication. For more on mailboxes and MSGID, refer to the DCAN chapter of the device-specific TRM [6].
2. The flash kernel is transferred to the device with 2 bytes of data per frame. The host programmer will transmit frames to the device, checking if data bytes 3 and 4 are non-zero values. Bytes 3 and 4 of the text file must be replaced with the hex value calculated from the final result of the bit timing register value (*CAN_CALC_BTRREG)* in order of least significant byte followed by the most significant byte. If the host programmer recognizes a bit-timing change with bytes 3 and 4, the host programmer will then send the bit-timing change to the device and re-initialize itself (skipping the following 7 reserved words). The device will increase the bitrate to the desired bit-timing and continue to receive frames until the kernel has finished download.

3. The ROM transfers control and the flash kernel begins to execute. There is a small delay in which the kernel must prepare the device for flash programming before it is read to begin communications, and in this time the kernel configures the PLL, flash wait states, and so forth.
   a. The F28P65x and F280015x devices will erase the user-designated flash banks and sectors at this point.
   b. The F28003x kernel erases at a later point
4. The kernel enters DCAN Boot mode and waits to receive message frames in Mailbox 1. The *CAN_CALC_BTRREG* value (*bootloader_can_timing.h)* is adjusted within the project to 1Mbps and the DCAN message buffer size is adjusted by the kernel from 2 bytes per frame to 8 bytes per frame, to allow for a faster download of the application.
5. The host programmer will delay for 5 seconds before sending the application image at 1 Mbps and payload of 8 bytes per frame.
6. At the beginning of the download process, a key, a few reserved fields, and the application entry point are read.
   a. The F28003x kernel begins to erase the flash at this point. Erasing flash can take a few seconds, so it is important to note that while it looks like the application load may have failed, it is likely that the flash is just being erased.
7. Once the flash is erased, the application load continues by transferring frames of data into blocks of application code, and programming that into flash 128-bits or 512-bits at a time.
   a. The F280015x and F28P65x flash kernels program 512-bits at a time
   b. The F28003x flash kernel and 128-bit version of the F28P65x flash kernel program 128-bits at a time
8. After a block of data is programmed into flash, the kernel continues to receive messages to program the next block of data. This process continues until the entire application has been programmed into flash.

Now that the application is programmed into flash, the flash kernel attempts to run the application by branching to the entry point that was transferred to it at the start of the application load process. A device reset is needed for this.

5 MCAN Flash Kernel

MCAN Flash Kernel runs on:

- TMS320F28003x
- TMS320F28P65x
- TMS320F28P55x

To find the location of the flash kernel projects for these devices, see Troubleshooting.

5.1 Implementation

The flash kernel project is modeled after the MCAN ROM bootloader. It goes straight into the MCAN_Boot function which has been modified to write to Flash. The MCAN module initialization for the flash kernel is the same as the bootloader – the clock source for the MCAN module, the nominal and data bit rates, GPIO pins, and so forth, are set by the kernel on initialization according to the boot mode. Before any application data is received, the F28P55x and F28P65x (not the F28P65x, this is done later) MCAN flash kernel erases the flash of the device, readying it for programming. Additionally, the F28P55x MCAN flash kernel project allows the user to specify which flash banks and flash sectors should be erased before the application is programmed. This is discussed in more detail in Custom Flash Bank and Sector Erase. After the appropriate locations in flash memory are erased, the application load begins.

The flash kernel receives 64 bytes at a time from the host and places the contents into an intermediate RAM buffer. This buffer is then written into Flash in 128-bit or 512-bit increments. The F28003x and F28P65x flash kernels write 128 bits at a time, while the F28P55x flash kernel writes 512 bits at a time. Before writing to a sector for the first time, the F28003x checks to see if the sector has been erased, and if it has not been erased, the Flash kernel has the Flash API execute an erase operation (The F28P55x and F28P65x flash kernels erase flash beforehand, as described above). After this, a buffer is filled up with content to be written into Flash, and a program command is sent from the Flash API. Once the write has occurred, the Flash kernel has the Flash API verify that the segment was written into Flash at the correct address. Once the kernel has copied everything to Flash, the project jumps to the entry address of the image.

All of the sections of the firmware image stored in flash should be aligned according to the number of bits being programmed at once. If programming 128-bits at once (F28003x and F28P65x), the sections should be aligned to a 128-bit boundary. In the linker command file for the firmware image, all initialized sections need to be mapped to Flash sectors, and after each mapping, an ALIGN(8) directive needs to be added to ensure the 128-bit alignment. If programming 512-bits at once (F28P55x), the sections should be aligned to a 512-bit boundary. In the linker command file for the firmware image, all initialized sections need to be mapped to Flash sectors, and after each mapping, an ALIGN(32) directive needs to be added to ensure the 512-bit alignment.

The protocol used to transfer the application data follows the MCAN ROM loader protocol. With the original MCAN ROM loader protocol, nominal bitrate used is 1Mbps and transmits 64 bytes per frame from the host to the target device for nominal bit timing. The data bitrate used by the protocol is 2Mbps for data bit timing.

5.1.1 Custom Flash Bank and Sector Erase

The F28P55x MCAN flash kernel allows the user to specify which flash banks and flash sectors should be erased before the application is loaded to flash. Within the flash_kernel_ex4_can_flash_kernel.c file, there are four 32-bit unsigned integer arrays that control the flash banks and sectors erased by the kernel.

```
uint32_t Flash_Banks_To_Erase[5] = {0,1,2,3,4};
uint32_t CMD_WE_Protection_A_Masks[5] = {0,0,0,0,0};
uint32_t CMD_WE_Protection_B_Masks[5] = {0,0,0,0,0};
uint32_t CMD_WE_Protection_UO_Masks[5] = {0,0,0,0,0};
```

Within Flash_Banks_To_Erase, the flash bank numbers should be entered. Within CMD_WE_Protection_A_Masks, the Write/Erase Protection Masks corresponding the sectors 0-31 for each bank in Flash_Banks_To_Erase should be entered. As shown above, flash sectors 0-31 of each bank will be erased. Within CMD_WE_Protection_B_Masks, the Write/Erase Protection Masks corresponding the sectors 32-127 for each bank in Flash_Banks_To_Erase should be entered. As shown, flash sectors 32-127 will be erased in each bank. For more information about these masks, see the device-specific Flash API Guide. Similarly, CMD_WE_Protection_UO_Masks should be filled with the desired masks for each bank's OTP. Note that configuring this array will not result in OTP being erased, but rather enable the device to program a flash bank's OTP during the application load.

Additionally, CMD_WE_Protection_A_Masks and CMD_WE_Protection_B_Masks is used when programming the application to flash. The proper masks are determined based on the target address of the data being programmed.

Once these arrays are configured as desired, recompile the flash kernel and generate the new kernel image.

5.1.2 Application Load

This section walks through the entire flow of programming an application into flash using the MCAN boot mode.

Ensure the device is ready for MCAN communications by resetting the device while ensuring the boot mode pins are in the proper state to select MCAN Boot mode. These are the steps that follow:

1. The device enters the MCAN Boot loader and waits to receive message frames in the RX Message Buffer. Acceptable messages have a MSGID value of 0x1 for boot-loader communication.
2. The flash kernel is transferred to the device with 64 bytes of data per frame at 1Mbps (nominal bitrate). The host programmer will transmit frames to the device until the kernel has finished download.
3. The ROM transfers control and the flash kernel begins to execute. There is a small delay in which the kernel must prepare the device for flash programming before it is read to begin communications, and in this time the kernel configures the PLL, flash wait states, and so forth.
   a. The F28P55x kernel will erase the user-designated flash banks/sectors at this point.
   b. The F28P65x and F28003x kernels erase flash at a later point.
4. The kernel enters MCAN Boot mode and waits to receive message frames in the RX Message Buffer. The bit-rate switching (BRS) value is adjusted within the project and allows the bitrate to be increased to 2Mbps (data bitrate), to allow for a faster download of the application.
5. The host programmer delays for 5 seconds before sending the application image at 2Mbps and payload of 64 bytes per frame.

6. At the beginning of the download process, a key, a few reserved fields, and the application entry point are read.
   a. The F28P65x kernel begins to erase the flash at this point. Erasing flash can take a few seconds, so it is important to note that while it looks like the application load may have failed, it is likely that the flash is just being erased.
7. Once the flash is erased, the application load continues by transferring frames of data into blocks of application code, and programming that into flash 128-bits or 512-bits at a time.
   a. The F28P55x flash kernel programs 512-bits at a time
   b. The F28003x and F28P65x flash kernels program 128-bits at a time
      i. Before programming data, the F28003x kernel checks each flash sector to see if it has been erased . If it is has not been previously erased, the sector is erased and the application data is programmed.
8. After a block of data is programmed into flash, the kernel continue to receive messages to program the next block of data. This process continues until the entire application has been programmed into flash.

Now that the application is programmed into flash, the flash kernel attempts to run the application by branching to the entry point that was transferred to it at the start of the application load process. A device reset is needed for this.

6 Example Implementation

The kernels described above are available in C2000Ware under examples folder for the specific device within the examples directory. For example, the DCAN flash kernel for F28003x is found at *C2000Ware_x_x_xx_xx* > *driverlib* > *f28003x* > *examples* > *flash* . The related host application is found in C2000Ware (*C2000Ware_x_xx_xx_xx* > *utilities* > *flash_programmers* > *dcan_flash_programmer*). The source and executable are found in the *dcan_flash_programmer* folder. This section details the dcan_flash_programmer: how to build, run and use it with DCAN Flash Kernel. Similarly, the MCAN Flash kernel will be described for usage with the can_flash_programmer.

Note

The flash kernel of the appropriate device must be supplied to the host application tool being used to program the flash. The host programmer starts the same way independent of the kernel or device. It first loads the kernel to the device through the CAN/MCAN ROM bootloader. After this, the tool's functionality differs depending on the device and kernel being used.

6.1 Device Setup

6.1.1 Flash Kernels

Flash kernel source and project files for Code Composer Studio (CCS) are provided in C2000Ware, in the corresponding device's examples directory. Load the project into CCS and build the project. These projects have a post-build step in which the compiled and linked .out file is converted into the correct boot hex format needed by either the CAN or MCAN ROM bootloader and is saved as the example name with a .txt extension.

6.1.2 Hardware

The hardware components needed to run the examples are a C2000 device connected to a CAN transceiver, and a PEAK PCAN-USB Pro FD Analyzer. For controlCards, a custom-designed CAN transceiver board needs to be used, as well as the HSEC-180-pin ControlCard Docking Station. The custom-designed transceiver board is connected to the ControlCard using four connections: ground, 3.3V, CANTX and CANRX.

The LaunchPad™ devices contain an on-board CAN transceiver. The PEAK PCAN-USB Pro FD Analyzer is connected to the LaunchPad through the ground, CAN-Lo and CAN-Hi connections. The onboard CAN Routing switch needs to be set low for the transceiver to communicate using the GPIOs.

After building the kernels in CCS, it is important to setup the device correctly to be able to communicate with the host PC running the host programmer. The first thing to do is make sure the boot mode value is configured correctly to boot the device to CAN/MCAN Boot mode. Next, connect the MCAN TX and RX pins using a transceiver to the CAN port on the host side.

6.2 Host Application: *dcan_flash_programmer*

6.2.1 Overview

The host is responsible for sending the DCAN kernel image and flash (firmware) image to the MCU. The PEAK PCAN-USB Pro FD CAN bus Analyzer is used as the host. The flash programmer project is built and run on *Visual Studio 2019*. The host programmer uses the *PCAN_Basic* API from PEAK. The *PCAN_Basic* API can be used to send and receive CAN frames on the CAN analyzer.

Note

The PEAK PCAN-USB Pro FD CAN bus Analyzer is backwards compatible to receive both classical CAN frames as well as CAN-FD frames.

On the F28003x device, the clock to the MCAN module is switched to the external clock source by the Boot ROM. The external clock is 20MHz in the LaunchPad and the ControlCard. The Boot ROM configures the nominal bit rate to be 100Kbps. The host CAN programmer configures the PEAK CAN analyzer to have the same clock and nominal bit rate value.

The host initializes the analyzer for CAN usage, and sends the kernel over in 2-byte increments, and then sends over the image in 8-byte increments, with a delay of 10 ms between each frame to give the Flash API time to program the data it is receiving into Flash. Once the firmware image has been written, the host CAN programmer exits.

The command line PC utility is a programming solution that can easily be incorporated into scripting environments for applications like production line programming. It was written using Microsoft Visual Studio® in C++. The project and its source can be found in C2000Ware (*C2000Ware_x_xx_xx_xx* > *utilities* > *flash_programmers* > *dcan_flash_programmer*).

To use this tool to program the C2000 device, ensure that the target board has been reset and is currently in the CAN boot mode and connected to the PC COM port. The host programmer will take in kernel and application files as inputs on the command line. There are also options for quiet or verbose output, and an option to wait on exit before closing the CAN flash programmer application. The command line usage of the tool is described below:

```
dcan_flash_programmer.exe –d <device> -k <kernel file> -a <app file> [-q] [-w] [-v]
```

-d <device>	- The name of the device to connect and load to: F28003x
-k <file>	- The file name for the CPU1 flash kernel. This file must be in the ASCII boot format.
-a <file>	- The application file name to download or verify to CPU1. This file must be in the ASCII SCI boot format.
-? Or -h	- Show help.
-q	- Quiet mode. Disable output to stdout.
-w	- Wait for a key press before exiting.
-v	- Enable verbose output.

Note

Both the flash kernel and flash application MUST be in the SCI8 boot format. The CAN/MCAN ROM loaders follow the same 8-bit channel boot format as the SCI ROM loader, which uses *-sci8* format option.

6.2.2 Building and Running dcan_flash_programmer Using Visual Studio

dcan_flash_programmer.cpp can be compiled using Visual Studio.

1. Navigate to the *dcan_flash_programmer* directory.
2. Double click the *dcan_flash_programmer.sln* to open the Visual Studioproject.
3. When Visual Studio opens, select Build → BuildSolution.
4. After Visual Studio completes the build, select Debug → *dcan_flash_programmer* → properties.
5. Select Configuration Properties →Debugging.
6. Select the input box next to the Command Arguments.

7. Type the arguments in the following format. The arguments are described in Section 6.2.1
   a. Format: `-d <device> -k <file> -a <file`
   b. `-d f28003x -k C:\Documents\flash_kernel.txt -a C:Documents\test.txt`
8. Click Apply and OK.
9. Select Debug → Start Debugging to begin running the project.

6.2.3 Running dcan_flash_programmer for F28003x

1. Navigate to the folder containing the compiled *can_flash_programmer* executable.
2. Run the executable *can_flash_programmer.exe* with the following command:

```
dcan_flash_programmer.exe -d f28003x -k <flash_kernel.txt> -a <file>
```

This first loads the *flash_kernel* into RAM of the device using the bootloader. Then, the kernel executes and loads and programs flash with the file specified by the '-a' command line argument, as seen in Figure 6-1 and Figure 6-2.

This automatically connects to the device, performs an auto baud lock, downloads the CPU1 kernel into RAM and executes it. Now, the CPU1 kernel is running and waiting for a packet from the host.

Figure 6-1. DCAN Flash Programmer Prompt After Downloading Flash Kernel to RAM

Figure 6-2. DCAN Flash Programmer After Downloading Flash Application

6.2.4 Using the Project With DCAN Bootloader

This section details the steps needed to run the DCAN Flash Kernel on the ControlCard:

1. Enter a command with the parameters as described below:
   a. Example: *dcan_flash_programmer.exe -d f28003x -k flash_kernel_ex5_dcan_flash_kernel.txt -a led_ex1_blinky.txt -v*
2. After the kernel is downloaded, it moves the application file to flash and confirms that the application load has completed.

For the LaunchPad, the following steps must be taken:

1. Set LaunchPad SW4 position to OFF (down). This is done to route the CANTX and CANRX signals to the header instead of the transceiver.
2. Open up a command window and navigate to where *dcan_flash_programmer.exe* is.
3. Enter a command with the parameters as described below:
   a. Example: *dcan_flash_programmer.exe -d f28003x -k flash_kernel_ex5_dcan_flash_kernel.txt -a led_ex1_blinky.txt -v*
4. After the kernel is downloaded, it transfers the firmware to flash and confirms that the application load has completed.

6.2.5 Using the Project With CCS

1. In CCS, import and build the CPU1 kernel project.
2. Launch the target configuration file.
3. Connect to CPU1.
4. Load the gel file provided in the project folder to the project. Right click on CPU1 in the target configuration and select "Open GEL Files View".
5. In the "GEL Files" tab, click on GEL Files. Right click in the "Script" window and select "Load GEL...". Navigate to the project folder and load the gel file.

6. In emulation mode, the following memory locations need to be set to enable CAN boot mode:
   a. Location 0xD00 with 0xFFFF
   b. Location 0xD01 with 0x5AFF
   c. Location 0xD04 with 0x00XX where XX is the boot mode for CAN Boot – 0x02, 0x22, 0x42 or 0x62. The SENDTEST CAN Boot modes of 0x82, 0xA2, 0xC2 and 0xE2 use the same pins as the first four configurations, respectively, and they also send out two CAN frames. In evaluation mode, using one of the SENDTEST modes ensures that the CAN module does not time out before the host starts sending the flash kernel over. To learn more about the SENDTEST modes, consult the DCAN boot source file located in C2000Ware (*C2000Ware_x_xx_xx_xx* > *driverlib* > *f28003x* > *examples* > *flash* > *DCAN_Boot.c*).

Figure 6-3 shows an example implementation of these memory locations. Once these locations have been programmed, reset the device and click resume. The F28003x device should now be waiting in CAN boot mode in ROM.

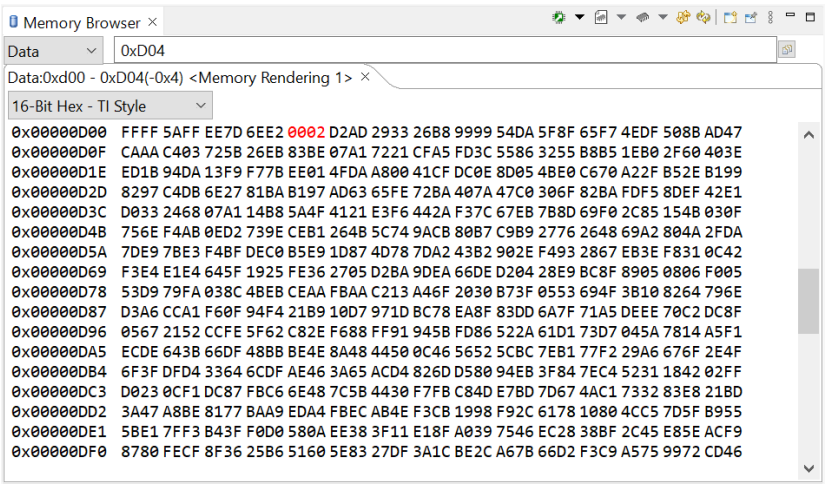

Figure 6-3. Example Memory Window in CCS (GPIO pins 4/5, CAN Boot Mode)

The PEAK CAN analyzer needs to be connected to the PC as well. No initialization needs to be done beforehand.

6.3 Host Application: *can_flash_programmer* [MCAN]

6.3.1 Overview

The host is responsible for sending the MCAN kernel image and flash (firmware) image to the MCU. The PEAK PCAN-USB Pro FD CAN bus Analyzer is used as the host. The flash programmer project is built and run on *Visual Studio 2019*. The host programmer uses the *PCAN_Basic* API from PEAK. The *PCAN_Basic* API can be used to send and receive CAN-FD frames on the CAN analyzer.

On the F28003x device, the clock to the MCAN module is switched to the external clock source by the Boot ROM. The external clock is 20MHz in the LaunchPad and the ControlCard. The Boot ROM configures the nominal bit rate to be 1Mbps, and the data bit rate to be 2Mbps. The host CAN programmer configures the PEAK CAN analyzer to have the same clock, nominal and data bit rate values.

The host initializes the analyzer for CAN-FD usage, sends the kernel over in 64-byte increments, and then sends over the image in 64-byte increments, with a delay of 100 ms between each frame to give the Flash API time to program the data it is receiving into Flash. Once the firmware image has been written, the host CAN programmer exits.

The command line PC utility is a programming solution that can easily be incorporated into scripting environments for applications like production line programming. It was written using Microsoft Visual Studio® in C++. The project and its source can be found in C2000Ware (*C2000Ware_x_xx_xx_xx* > *utilities* > *flash_programmers* > *can_flash_programmer*).

To use this tool to program the C2000 device, ensure that the target board has been reset and is currently in the CAN boot mode and connected to the PC COM port. The host programmer will take in kernel and application files as inputs on the command line. There are also options for quiet or verbose output, and an option to wait on exit before closing the CAN flash programmer application. The command line usage of the tool is described below:

```
can_flash_programmer.exe –d <device> -k <kernel file> -a <app file> [-q] [-w] [-v]
```

-d <device>	- The name of the device to connect and load to: F28003x, F28P65x, F28P55x
-k <file>	- The file name for the CPU1 flash kernel. This file must be in the ASCII boot format.
-a <file>	- The application file name to download or verify to CPU1. This file must be in the ASCII SCI boot format.
-? Or -h	- Show help.
-q	- Quiet mode. Disable output to stdout.
-w	- Wait for a key press before exiting.
-v	- Enable verbose output.

Note

Both the flash kernel and flash application MUST be in the SCI8 boot format. The CAN/MCAN ROM loaders follow the same 8-bit channel boot format as the SCI ROM loader, which uses *-sci8* format option.

6.3.2 Building and Running can_flash_programmer Using Visual Studio

can_flash_programmer.cpp can be compiled using Visual Studio.

1. Navigate to the *can_flash_programmer* directory.
2. Double click the *can_flash_programmer.sln* to open the Visual Studio project.
3. When Visual Studio opens, select Build → BuildSolution.
4. After Visual Studio completes the build, select Debug → *can_flash_programmer* → properties.
5. Select Configuration Properties →Debugging.
6. Select the input box next to the Command Arguments.
7. Type the arguments in the following format. The arguments are described in Section 6.3.1.
   a. Format: `-d <device> -k <file> -a <file`
   b. `-d f28003x -k C:\Documents\flash_kernel.txt -a C:Documents\test.txt`
8. Click Apply and OK.
9. Select Debug → Start Debugging to begin running the project.

6.3.3 Running can_flash_programmer for F28003x

1. Navigate to the folder containing the compiled *can_flash_programmer* executable.
2. Run the executable *can_flash_programmer.exe* with the following command:

```
can_flash_programmer.exe –d f28003x –k <flash_kernel.txt> -a <file>
```

This first loads the *flash_kernel* into RAM of the device using the bootloader. Then, the kernel executes and loads and programs flash with the file specified by the '-a' command line argument, as seen in and .

This automatically connects to the device, performs an auto baud lock, downloads the CPU1 kernel into RAM and executes it. Now, the CPU1 kernel is running and waiting for a packet from the host.

Figure 6-4. CAN Flash Programmer Prompt After Downloading Flash Kernel to RAM

Figure 6-5. CAN Flash Programmer After Downloading Flash Application

6.3.4 Using the Project With MCAN Bootloader

This section details the steps needed to run the MCAN Flash Kernel on the ControlCard:

1. Enter a command with the parameters as described below:
   a. Example: *can_flash_programmer.exe -d f28003x -k flash_kernel_ex4_can_flash_kernel.txt -a led_ex1_blinky.txt -v*
2. After the kernel is downloaded, it moves the application file to flash and confirms that the application load has completed.

For the LaunchPad, the following steps must be taken:

1. Set LaunchPad SW4 position to OFF (down). This is done to route the CANTX and CANRX signals to the header instead of the transceiver.
2. Open up a command window and navigate to where *can_flash_programmer.exe* is.
3. Enter a command with the parameters as described below:
   a. Example: *can_flash_programmer.exe -d f28003x -k flash_kernel_ex4_can_flash_kernel.txt -a led_ex1_blinky.txt -v*
4. After the kernel is downloaded, it transfers the firmware to flash and confirms that the application load has completed.

6.3.5 Using the Project With CCS

1. In CCS, import and build the CPU1 kernel project.
2. Launch the target configuration file.
3. Connect to CPU1.
4. Load the gel file provided in the project folder to the project. Right click on CPU1 in the target configuration and select "Open GEL Files View".
5. In the "GEL Files" tab, click on GEL Files. Right click in the "Script" window and select "Load GEL...". Navigate to the project folder and load the gel file.
6. In emulation mode, the following memory locations need to be set to enable MCAN boot mode:
   a. Location 0xD00 with 0xFFFF
   b. Location 0xD01 with 0x5AFF
   c. Location 0xD04 with 0x00XX where XX is the boot mode for MCAN Boot – 0x08, 0x28, or 0x48. The SENDTEST MCAN Boot modes of 0x68, 0x88, and 0xA8 use the same pins as the first three configurations, respectively, and they also send out two CAN-FD frames. In evaluation mode, using one of the SENDTEST modes ensures that the MCAN module does not time out before the host starts sending the flash kernel over. To learn more about the SENDTEST modes, consult the MCAN boot source file located in C2000Ware (*C2000Ware_x_xx_xx_xx* > *driverlib* > *f28003x* > *examples* > *flash* > *MCAN_Boot.c*).

Figure 6-6 shows an example implementation of these memory locations. Once these locations have been programmed, reset the device and click resume. The F28003x device should now be waiting in MCAN boot mode in ROM.

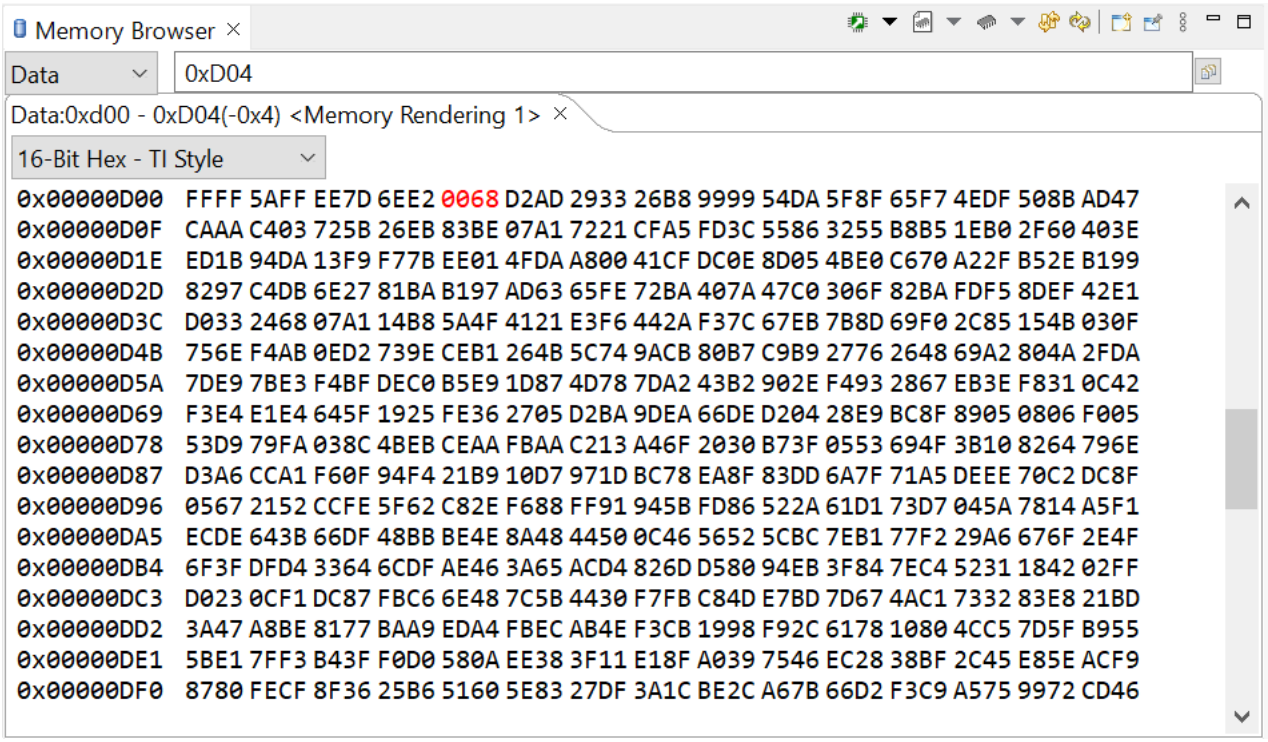

Figure 6-6. Example Memory Window in CCS (GPIO pins 4/5, SENDTEST MCAN Boot Mode)

The PEAK CAN analyzer needs to be connected to the PC as well. No initialization needs to be done beforehand.

After this, the following command should be entered into the command line in the directory where the *can_flash_programmer* is:

```
can_flash_programmer.exe -d f28003x -k -a
```

The host initializes the CAN analyzer and opens up the flash kernel file to transfer to the MCAN bootloader. Once the entire file has been sent over, the host delays for 5 seconds before sending over the firmware image.

6.4 Application Load: CPU2 Image

For the F28P65x device there are multiple Flash Banks with unique starting addresses, so programming the CPU2 image can be done by combining both CPU1 application image with CPU2 application image.

6.4.1 Combining Two Images (.txt)

Each generated application .txt file has a header at the top and a terminating header at the bottom of the file. These files cannot be combined using text editors as is due to the presence of multiple start and terminating headers.

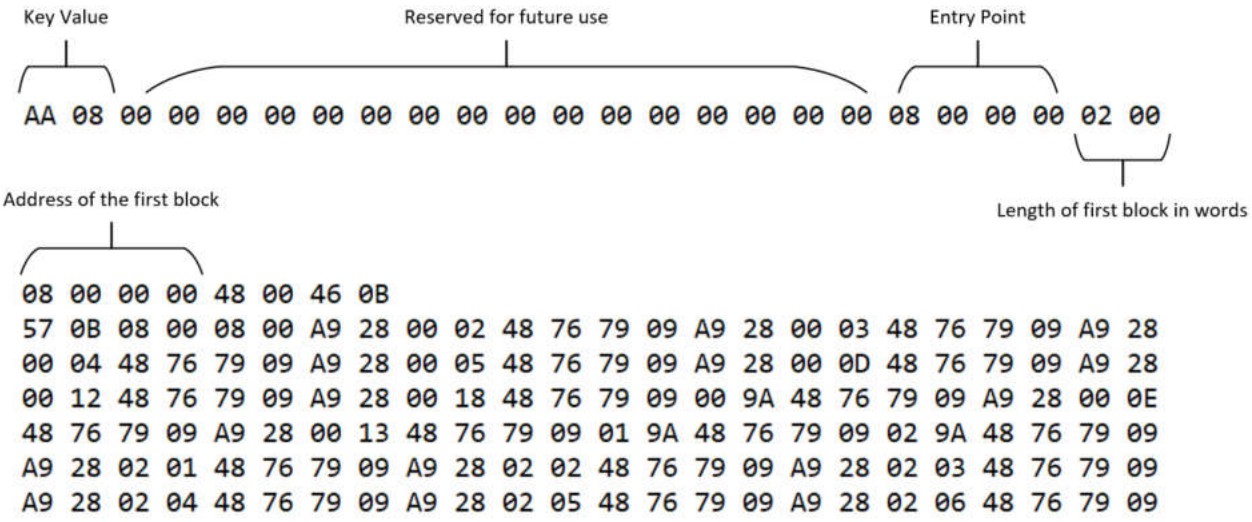

Figure 6-7. Image A

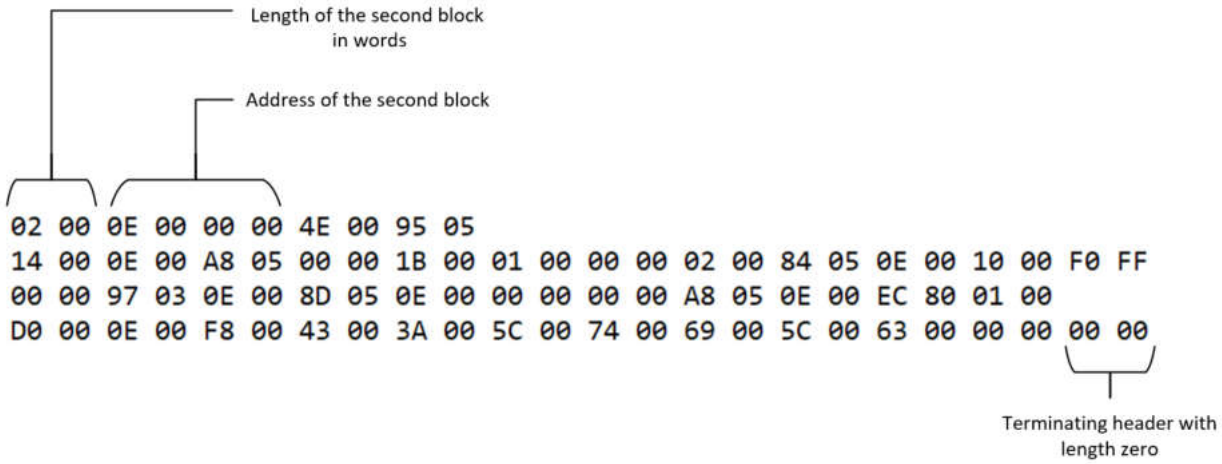

Figure 6-8. Image B

For example, one solution is to remove the terminating header as shown above in Image B and the start header from Image A (key value, reserved words, entry point). Both Image A and Image B have a header at the top and terminating header at the bottom of their respective files. Image A should be appended to the bottom of Image B. Once removed, we are able to successfully combine the two images in a single .txt file. Attention must be given to the address block of each image to ensure no data is lost.

Note

The order of merging Image A and Image B matters for what the CAN/MCAN boot loader will recognize as the starting point address for CPU1. If Image A should be the application image for CPU1 usage, Image A's terminating header should be removed and Image B's top header should be removed as well. Image B must then be appended to Image A's .txt file.

7 Troubleshooting

Below are solutions to some common issues encountered by users when utilizing the CAN or MCAN flash kernel.

7.1 General

Question: I cannot find the kernel projects, where are they?

Answer:

Device	Build Configuration	Location
F28003x	RAM	C2000Ware_x_xx_xx_xx > driverlib > f28003x > examples > flash
F28P65x	RAM	C2000Ware_x_xx_xx_xx > driverlib > f28p65x > examples > c28x_dual > flash_kernel
F280015x	RAM	C2000Ware_x_xx_xx_xx > driverlib > f280015x > examples > flash

Question: What are the first things I should check if the flash kernel does not download?

Answer:

- One area of the program to check would be the linker command file - make sure that all flash sections are aligned to proper boundaries. The F28P65x and F280015x DCAN Flash Kernels display the 512-bit programming capability of their respective Flash API. Thus, applications loaded with these two kernels should be 512-bit aligned. Similarly, applications loaded with the F28P55x MCAN flash kernel should be 512-bit aligned. In SECTIONS, add a comma and "ALIGN(32)" after each line where a section is allocated to flash.

  For all other flash kernels mentioned in this document, the 128-bit programming function of the Flash API is used. Thus, applications loaded with these kernels should be 128-bit aligned. In SECTIONS, add a comma and "ALIGN(8)" after each line where a section is allocated to flash.
- One other common issue users encounter is that they are not using the correct boot pins for the CAN/MCAN boot mode. For example, on the F28003x devices, MCAN boot has three options for GPIO pins to use. Make sure that the pins for the default option are not being used for something else.
- When using long cables, use a lower baud rate to get rid of noise.
- For baud rate and connection issues, try running the CAN/MCAN loopback and transmit examples for the device (you should be able to find these in *C2000Ware* in the driverlib/examples folder for your device).
- The operational CAN-FD clock range of the PEAK PCAN USB-PRO FD Analyzer is 20MHz to 80MHz. The MCANxBIT Clock of the device must match the CAN-FD clock frequency of the PEAK tool in order for frames to transmit.
- Both the *dcan_flash_programmer* and the *can_flash_programmer* require use of the PEAK PCAN USB-Pro FD Analyzer, as the tool is capable of transmitting standard CAN frames and CAN-FD frames. The PEAK PCAN USB FD device is also compatible.

Question: I have combined the application images yet still cannot program, how can this be resolved?

Answer: Be sure that the first image in combined .txt file contains CPU1's image, followed by CPU2's image.

7.2 DCAN Boot

Question: I am receiving DCAN Bus Errors after downloading the DCAN Kernel. What steps can be taken to resolve this?

Answer: Make sure that the DCAN Kernel file being transmitted has correct values for the DCAN bit timing settings. Bytes 3 and 4 of the text file must be replaced with the hex value calculated from the final result of the bit timing register value (*CAN_CALC_BTRREG)* in order of least significant byte followed by the most significant byte.

Question: For the F28P65x or F280015x device, I cannot download the application image following the download of the DCAN Flash kernel to RAM in DCAN boot mode, what should I do?

Answer: Make sure that the clock used in generation of the DCAN Flash kernel project is using the internal crystal oscillator known as INTOSC2 (*device.h)*. At power-up, the device boot ROM is clocked from an on-chip 10MHz oscillator (INTOSC2). This value needs to be set as the primary internal clock source and is the default clock at reset.

Question: For the F28P65x device, I am able to download the application image following download of the DCAN Flash kernel to RAM in DCAN boot mode, but CPU2 is not able to execute its application. How can I resolve this?

Answer: CPU1's application image will have to set CPU2's selected banks, GSxRAM and give GPIO control to CPU2. CPU1 must then execute its application image before CPU2 executes its application image, otherwise, it can enter the illegal ISR section of memory. For more details on these items, consult the device-specific TRM [7].

Question: Example projects exist for F28003x, F28P65x, and F280015x, how can I adapt these to F20013x?

Answer: To implement your own version of an F280013x DCAN flash kernel is fairly straightforward. The changes required to adapt the existing F280015x DCAN flash kernel for the F280013x device are minimal. Listed below are the major changes required to port the project:

- Copy the existing .projectspec file within C2000Ware (C2000Ware_x_xx_xx_xx > driverlib > f280015x > examples > flash>CCS>f280015x_flash_eex5_dcan_flash_kernel.projectspec
- Change the name of the project
- Change the filepaths that are specific to F280015x to the F280013x equivalent
  - Example: <file action="copy" path="../../../../../libraries/flash_api/f280015x/lib/FAPI_F280015x_EABI_v2.00.10.lib" targetDirectory="" /> should now become <file action="copy" path="../../../../../libraries/flash_api/f280013x/lib/FAPI_F280013x_EABI_v2.00.10.lib" targetDirectory="" />
- Import the new .projectspec to CCS
- Change any references to F280015x header files to the F280013x equivalent
  - Example: FlashTech_F280015x_C28x.h > FlashTech_F280013x_C28x.h
- Verify that the proper GPIOs and boot modes are configured by the kernel

7.3 MCAN Boot

Question: For the F28P65x or F28P55x devices, I cannot download the application image following the download of the MCAN Flash kernel to RAM in MCAN boot mode, what should I do?

Answer: Make sure that the clock used in generation of the MCAN Flash kernel project is using the internal crystal oscillator known as INTOSC2 (*device.h)*. At power-up, the device boot ROM is clocked from an on-chip 10MHz oscillator (INTOSC2). This value needs to be set as the primary internal clock source and is the default clock at reset.

Question: For the F28P65x device, I am able to download the application image following download of the MCAN Flash kernel to RAM in MCAN boot mode, but CPU2 is not able to execute its application. How can I resolve this?

Answer: CPU1's application image will have to set CPU2's selected banks, GSxRAM and give GPIO control to CPU2. CPU1 must then execute its application image before CPU2 executes its application image, otherwise, it can enter the illegal ISR section of memory. For more details on these items, consult the device-specific TRM [7].

8 References

1. Texas Instruments: *ROM Code and Peripheral Booting* section from the *TMS320F2837xD Dual-Core Delfino Microcontrollers Technical Reference Manual*
2. Texas Instruments: *TMS320C28x Assembly Language Tools User's Guide*
3. Texas Instruments: *USB Flash Programming of C2000 Microcontrollers*
4. Texas Instruments: *TMS320F28003x Boot Features and Configurations*
5. Texas Instruments: *LAUNCHXL-F280039C Overview User's Guide*
6. Texas Instruments: *TMS320F28003x Real-Time Microcontrollers Technical Resource Manual*
7. Texas Instruments: *TMS320F28P65x Real-Time Microcontrollers Technical Resource Manual*
8. Texas Instruments: *Introduction to the Controller Area Network (CAN)*
9. Texas Instruments: *Getting Started with the MCAN (CAN FD) Module*

9 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (December 2023) to Revision A (April 2024) **Page**

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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, regulatory 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 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.

TI objects to and rejects any additional or different terms you may have proposed.

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