Application Report
SPRACN1–May 2019

C2000™ Software Controlled Firmware Update Process

Baskaran Chidambaram and David Foley

ABSTRACT

This application report describes a software controlled firmware update process on C2000 devices using existing boot modes without the need to manually select boot mode. The method described in this application note directly applies to TMS320F28004x device and can be applied to legacy devices with necessary modifications.

Contents

List of Figures

List of Tables

Trademarks

C2000, Code Composer Studio are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

C2000 devices support multiple boot modes say Serial Communications Interface (SCI), Universal Serial Bus (USB), Flash, Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI), and so forth. In most of the applications the firmware code is stored in flash memory and executes from there. But when it comes to firmware update (where JTAG support is not available) it is typically transferred through one of the peripheral boot modes – SCI, USB, I2C, SPI and then updated on to flash. This process is done with the help of flash loaders (Serial, USB) provided with C2000Ware. But this involves manual intervention (or external chip) to change the boot mode from flash to SCI/USB and once the update is completed the boot mode has to be changed back to flash. This app note suggests a method to do firmware update without changing the boot mode pins via an external source. This is achieved by configuring custom boot modes in user OTP region and connecting resistor and capacitor components to the boot mode select GPIO.

The method described here is validated using SCI peripheral boot on TMS320F28004x device. There are three key aspects detailed in this report are:

- Custom boot pin selection and boot mode definition in user OTP
- Hardware considerations
- Software considerations

2 Configuring Custom Boot Mode

This section details the method to select custom boot pin, define a custom boot table, and write these values to OTP.

NOTE: It is assumed that you are familiar with concepts described in the *ROM Code and Peripheral Booting* chapter of the *TMS320F28004x Piccolo Microcontrollers Technical Reference Manual*. Many of the keywords used in the following sections are detailed in the TRM.

2.1 Custom Boot Pin Selection

The default boot modes in TMS320F28004x need 2 GPIOs (GPIO32, GPIO24) to select one of the supported boot modes. For the purpose of firmware update we need to select between SCI or Flash boot modes which can be achieved with just one GPIO. This can be customized by configuring the boot configuration word in OTP to 0x5AFFFF0F. This selects GPIO 15 (0x0F) as the boot select pin.

BOOTPIN_CONFIG.BMSP0 to **0x0F** (this selects 15 as boot select pin)

BOOTPIN_CONFIG.BMSP1 to 0xFF

BOOTPIN_CONFIG.BMSP2 to 0xFF

BOOTPIN_CONFIG.KEY to **0x5A**

2.2 Custom Boot Mode Definition

The default boot definition fixes the value (boot mode number) to be driven (on boot select pins) for particular boot mode as given in Table 1.

Table 1. Default Boot Definition

Boot Modes	Boot Mode Number
PARALLEL IO	0
SCI BOOT	1
CAN Boot	2
Flash Boot	3

Table 2 shows a custom boot table.

Table 2. Custom Boot Table

Boot Modes	Boot Mode Number
Flash Boot	0
SCI BOOT	1

This is achieved by configuring BOOT_DEF_LOW (0xFFFF0103) and BOOT_DEF_HIGH (0xFFFFFFFF) user OTP locations as shown in Table 3.

Table 3. Custom Boot Table

Boot Mode Number	BOOTDEF Name	Value
0	BOOT_DEF0	03 (Flash boot, refer Table 1)
1	BOOT_DEF1	01 (SCI boot, refer Table 1)
2	BOOT_DEF2	0xFF
3	BOOT_DEF3	0xFF
4	BOOT_DEF4	0xFF
5	BOOT_DEF5	0xFF
6	BOOT_DEF6	0xFF
7	BOOT_DEF7	0xFF

The configurations derived above can be summarized as:

- BOOTPIN_CONFIG - 0x5AFFFF0F
- BOOT_DEF_LOW - 0xFFFF0103
- BOOT_DEF_HIGH - 0xFFFFFFFF

These values have to be written to user OTP locations as described in the next section.

NOTE: Care should be taken while defining the above configurations as the OTP locations can be written only once.

For detailed information on the above configurations, see the *Device Boot Modes* chapter in the *TMS320F28004x Piccolo Microcontrollers Technical Reference Manual*.

2.3 Writing the Values to User OTP

The custom boot configurations (derived in above sections) can be updated in user OTP by following these steps:

1. Download and install C2000Ware.
2. Pick any of the C2000Ware examples and add the following code snippet above the main function (it can be anywhere in the file just outside the functions).

```
#pragma RETAIN(otp_z1_data_1)
#pragma DATA_SECTION(otp_z1_data_1,"dcsm_zsel_z1");
const long otp_z1_data_1 = 0x5AFFFF0F;

#pragma RETAIN(otp_z1_data_2)
#pragma DATA_SECTION(otp_z1_data_2,"dcsm_zsel_z1_2");
const long otp_z1_data_2 = 0xFFFF0103;
```

3. In the linker command file (28004x_generic_ram_lnk.cmd), add the following lines.

```
MEMORY
{
   PAGE 0:
       DCSM_ZSEL_Z1_P0: origin = 0x07800C, length = 0x000002
       DCSM_ZSEL_Z1_P1: origin = 0x07801C, length = 0x000002
}
SECTIONS
{
   dcsm_zsel_z1_1   : > DCSM_ZSEL_Z1_P0,     PAGE = 0
   dcsm_zsel_z1_2   : > DCSM_ZSEL_Z1_P1,     PAGE = 0
}
```

4. Re-compile the example and load to the target via JTAG using Code Composer Studio™ (CCS). The loader and flash API plugin in CCS will take care of writing these values to OTP location.

NOTE: These values have to be selected and written carefully as the OTP locations cannot be re-written.

3 Hardware Modifications

Once the above settings are programmed into the OTP, GPIO15 is selected as the boot select pin with a low value (0) on the pin selecting Flash boot mode and a high value (1) on the pin selecting SCI boot mode. A resistor and a capacitor need to be connected between GPIO15 and ground to enable software control of the boot mode, as and when needed by the application. The resistor rating needed is derived to be 1KΩ and the capacitor rating is derived to be 10 nF. These values are derived considering the worst case leakage current of 2 µA and other device characteristics defined in the *TMS320F28004x Piccolo™ Microcontrollers Data Manual*.

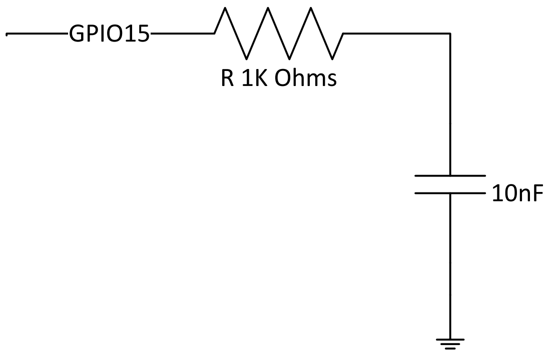


Figure 1. Hardware Connections

4 Software Modifications

4.1 Application Software Modifications

The following care-about has to be considered in the application software.

- Upon powering up, the application will boot to flash as GPIO15 will be low. In the beginning of the application drive GPIO as an output low to take care of any noise in the pin. This is just a precautionary measure to ensure we get into flash boot mode by default if reset happens at any time.
- When the application decides to update the firmware, the GPIO15 pin is driven high by the application software in order to charge the capacitor up. A sufficient time (50 µs) is allowed to ensure the capacitor is charged enough so that, once a reset occurs and the pin turns back into an input, it can drive high long enough for the boot code to read it as high.
- Then, trigger a software reset. Now when the boot code decodes the boot mode, it will read the GPIO15 pin as high and select the SCI boot mode.

The updated code has to be rebuilt and the hex files have to be generated by following the steps mentioned in readme.txt at \ti\c2000\C2000Ware_1_00_06_00\utilities\flash_programmers\serial_flash_programmer\.

The following is a code snippet from the led blink example highlighting the changes in red.

```
//
// Included Files
//
#include "driverlib.h"
#include "device.h"
/*
#pragma RETAIN(otp_z1_data)
#pragma DATA_SECTION(otp_z1_data,"dcsm_zsel_z1");
const long otp_z1_data = 0x5AFFFF0F;

#pragma RETAIN(otp_z1_data_2)
#pragma DATA_SECTION(otp_z1_data_2,"dcsm_zsel_z1_2");
const long otp_z1_data_2 = 0xFFFF0103;
*/
//
// Main
//
void main(void)
{
    uint32_t index = 0;

    //
    // Initialize device clock and peripherals
    //
    Device_init();

    //
    // Initialize GPIO and configure the GPIO pin as a push-pull output
    //
    Device_initGPIO();
    GPIO_setPadConfig(DEVICE_GPIO_PIN_LED1, GPIO_PIN_TYPE_STD);
    GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED1, GPIO_DIR_MODE_OUT);

    //
    // Drive GPIO pin low to take care of noises
    //
    GPIO_setPadConfig(15, GPIO_PIN_TYPE_STD);
    GPIO_setDirectionMode(15, GPIO_DIR_MODE_OUT);
    GPIO_writePin(15, 0);
    DEVICE_DELAY_US(50);

    //
    // Initialize PIE and clear PIE registers. Disables CPU interrupts.
    //
    Interrupt_initModule();

    //
    // Initialize the PIE vector table with pointers to the shell Interrupt
    // Service Routines (ISR).
    //
    Interrupt_initVectorTable();

    //
    // Enable Global Interrupt (INTM) and realtime interrupt (DBGM)
    //
    EINT;
    ERTM;

    //
    // Loop Forever
    //
    for(index = 0; index < 5; index++)
    {
        //
        // Turn on LED
        //
        GPIO_writePin(DEVICE_GPIO_PIN_LED1, 0);

        //
        // Delay for a bit.
        //
        DEVICE_DELAY_US(500000);

        //
        // Turn off LED
        //
        GPIO_writePin(DEVICE_GPIO_PIN_LED1, 1);

        //
        // Delay for a bit.
        //
        DEVICE_DELAY_US(500000);
    }

    //
    // Drive GPIO pin high to select flash boot mode
    //
    GPIO_writePin(15, 1);
    DEVICE_DELAY_US(50);

    SysCtl_resetDevice();
}
```

4.2 Flash Kernel Modifications

The following modifications have to be made in the flash kernel software (SCI/USB). In this example, the SCI kernel is used as a reference. To begin with, the GPIO15 pin has to be driven high. Once the firmware update is complete, the GPIO15 pin has to be driven low and then, after 50 µs, the device reset has to be triggered.

The updated code has to be rebuilt and the hex files generated by following the steps mentioned in readme.txt at <c2000Ware Installation Dir>\utilities\flash_programmers\serial_flash_programmer\.

- Drive the GPIO15 pin high to ensure the boot comes back to the firmware update mode (SCI boot) if any interruption occurs during the firmware update process.

```
uint32_t main(void)
{
    //
    // flush SCIA TX port by waiting while it is busy, driverlib.
    //
    sciaFlush();

    //
    // initialize device and GPIO, driverlib.
    //
    Device_init();
    Device_initGPIO();

    //
    // init interrupt and vectorTable, drivelib.
    //
    Interrupt_initModule();
    Interrupt_initVectorTable();

    //
    // Drive GPIO pin to high
    //
    GPIO_setPadConfig(15, GPIO_PIN_TYPE_STD);
    GPIO_setDirectionMode(15, GPIO_DIR_MODE_OUT);

    GPIO_writePin(15, 1);
    DEVICE_DELAY_US(50);

    //
    // Enable Global Interrupt (INTM) and realtime interrupt (DBGM)
    //
    EINT;
    ERTM;

    //
    // initialize flash_sectors, fapi + driverlib
    //
    initFlashSectors();

    uint32_t EntryAddr;

    //
    // parameter SCI_BOOT for GPIO28 (RX),29 (TX) is default.
    //
    EntryAddr = sciGetFunction(SCI_BOOT);
    return(EntryAddr);
}
```

- Once the firmware is updated, drive the GPIO15 pin low for 50 µs so as to drain the capacitor. This ensures that when reset occurs, the flash boot mode will be detected.

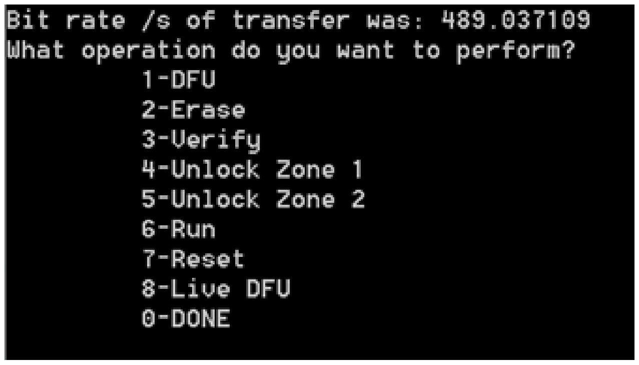
```
uint32_t sciGetFunction(uint32_t BootMode)
{

    volatile uint32_t EntryAddr;
    uint16_t command;
    uint16_t data[10]; // 16*10 = 128 + 32
    uint16_t length;

    < Removed rest of code to keep it short >

    while(command != RESET_CPU1)
    {
        < Removed rest of code to keep it short >

        GPIO_setPadConfig(15, GPIO_PIN_TYPE_STD);
        GPIO_setDirectionMode(15, GPIO_DIR_MODE_OUT);
        GPIO_writePin(15, 0);
        DEVICE_DELAY_US(50);

        //
        // Get next Packet
        //
        //command = sciGetPacket(&length, data); //get next packet
        command = RESET_CPU1;
    }

    //
    // Reset with WatchDog Timeout
    //
    EALLOW;
    SysCtl_setWatchdogMode(SYSCTL_WD_MODE_RESET);
    SysCtl_enableWatchdog();
    EDIS;

    while(1){}
}
```

4.3 Firmware Update Process

The updated kernel can be utilized to program the application image into flash by using the serial flash utility. When the application enters the firmware update mode, as shown in Figure 3, execute the following from the command prompt.

```
serial_flash_programmer.exe –d f28004x –k
f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel.txt -
a f28004x_fw_upgrade_example\led_ex1_blinky.txt -b 9600 -p COM8
```

Once the kernel is downloaded, the serial flash utility presents a menu on the command line:



```
Bit rate /s of transfer was: 489.037109
What operation do you want to perform?
          1-DFU
          2-Erase
          3-Verify
          4-Unlock Zone 1
          5-Unlock Zone 2
          6-Run
          7-Reset
          8-Live DFU
          0-DONE
```

Figure 2. Menu on the Command Line

Select option “1-DFU”. Now the application image will get downloaded and updated into Flash.

5 Flowchart

Figure 3 shows the main flow for the application as well as the flow to update the firmware in flash through the SCI boot mode (after implementing the changes mentioned in previous section).

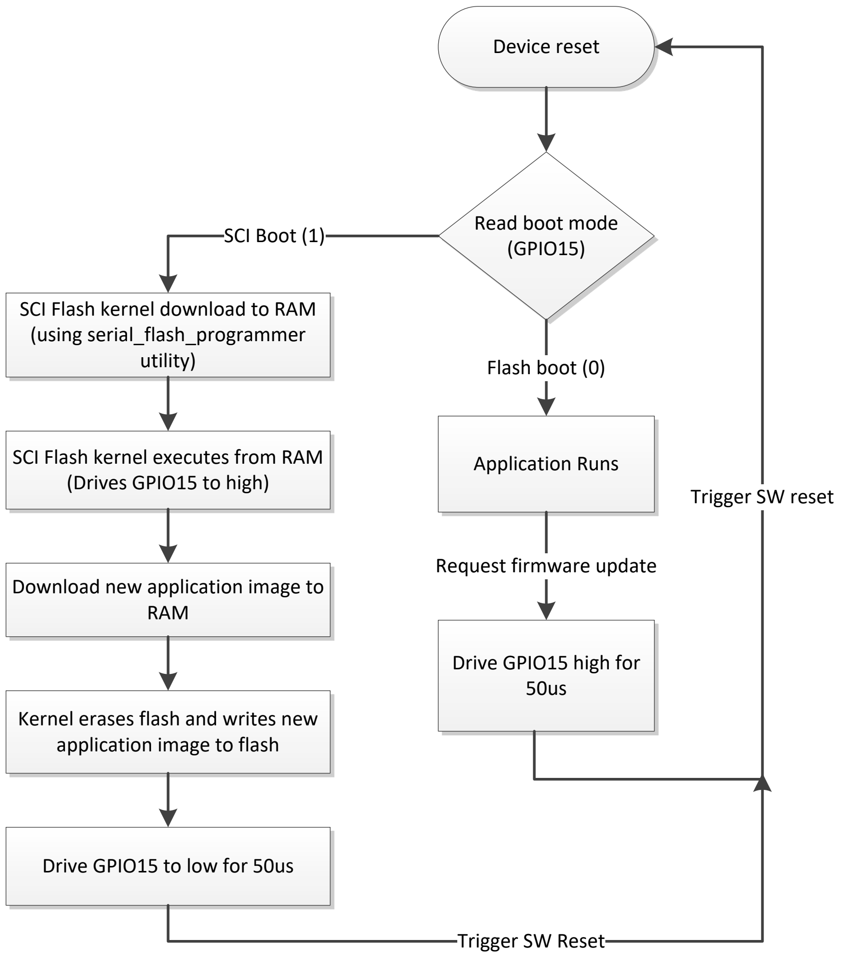

Device reset
Read boot mode (GPIO15)
SCI Boot (1)
SCI Flash kernel download to RAM (using serial_flash_programmer utility)
SCI Flash kernel executes from RAM (Drives GPIO15 to high)
Download new application image to RAM
Kernel erases flash and writes new application image to flash
Drive GPIO15 to low for 50us
Trigger SW Reset
Flash boot (0)
Application Runs
Request firmware update
Drive GPIO15 high for 50us
Trigger SW reset

Figure 3. Firmware Update Flow

6 Summary

The method mentioned above can be used to update the firmware in flash using the SCI boot option in end product, without the user manually toggling the boot switch. This also implies that there is no need to have an on board toggle switch to select the boot mode, thereby simplifying the board design. Thus, the whole process of a firmware update can be handled in software with minor modifications to hardware (adding a resistor and a capacitor).

7 References

- Texas Instruments: *TMS320F28004x Piccolo Microcontrollers Technical Reference Manual*
- Texas Instruments: *TMS320F28004x Boot Features and Configurations*
- Texas Instruments: *TMS320F28004x Piccolo™ Microcontrollers Data Manual*
- Texas Instruments: *Serial Flash Programming of C2000 Microcontrollers*
- C2000Ware Installer