

CC253x Flash Programming Guide

1	Introduction	3
2	Flash Programming	4
2.1	Macros	4
2.2	Program flow	4
2.3	Algorithm	5
2.4	Verification	8
3	References	9
4	Document History	9
5	Important Notice	10

1 Introduction

This programming guide explains how to program the flash memory of a CC253x System-on-Chip RF-IC [1] in an efficient way using the two-wire serial debug interface. An understanding of the Debug Interface and its various commands, the DMA Controller and the Flash Controller is recommended before reading this guide. This information can be found in the CC253x User Guide [2].

An example flash programmer written in C is used as reference throughout this guide. The example programmer is written for CC2530 (8051). Note that not all elements of the example program are described in this guide. Only the main functions related to programming are described. You may think of this guide as a supplement to the program code.

DUP (*Device Under Programming*) is used throughout this document to refer to the device being programmed. The device that performs the programming is referred to as the “*Programmer*”. Figure 1 shows the setup used for the example flash programmer. The hardware assumed used for this implementation example is what is found as part of the CC2530 Development Kit [3]:

- 2 x SmartRF05 Evaluation Boards (SmartRF05EB)
- 2 x CC2530 Evaluation Modules (CC2530EM)

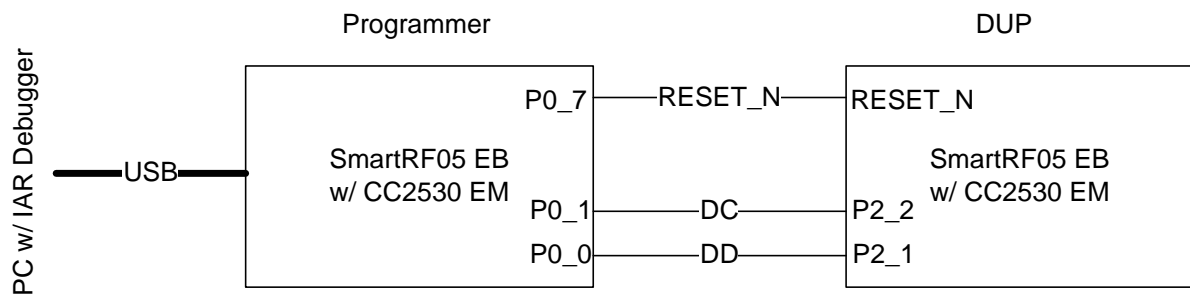


Figure 1: Flash Programming Test Setup

2 Flash Programming

This section describes the flash programming. The CPU in the DUP will be halted during the whole process. Reading/writing registers in the DUP is performed using the Debug Interface. The DMA Controller will perform all of the data transfer in the DUP. A double buffering scheme is used to guarantee a steady flow of data to the flash controller during write. Figure 2 shows how the four DMA channels are set up in the DUP. Maximum two of the channels will be active at the same time, channel 1 and 4 or channel 2 and 3. While DMA channel 1 fills up BUF0, DMA channel 4 writes the contents in BUF1 to the flash. While DMA channel 2 fills up BUF1, DMA channel 3 writes the contents in BUF0 to the flash.

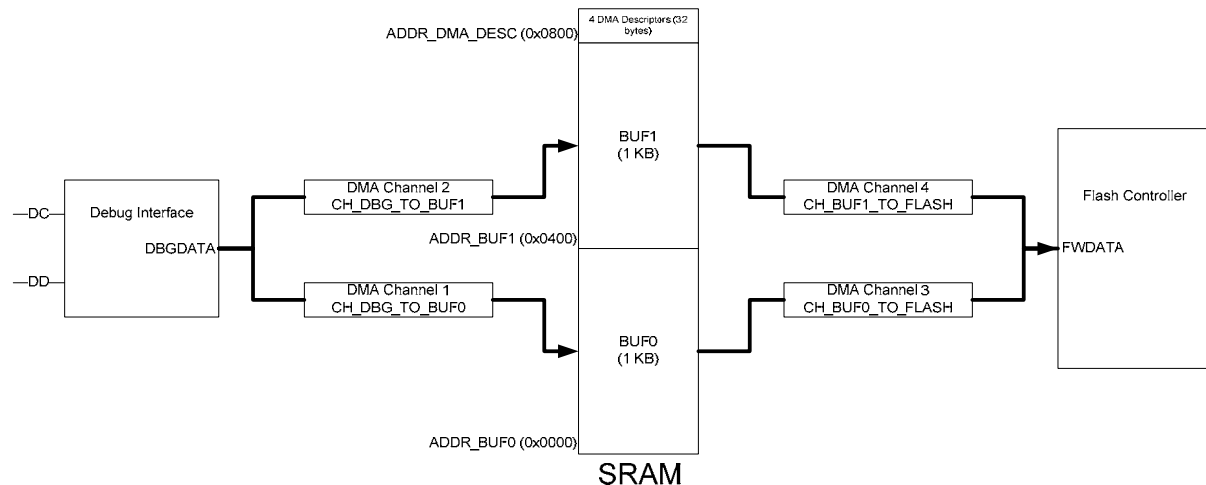


Figure 2: DMA Channels and data buffers in DUP

2.1 Macros

The XREG_TO_INT() macro is used to convert an XREG register declaration to a 16-bit value, i.e. get the address of the XREG register. HIBYTE() extracts the high byte of a 16-bit word. LOBYTE() extracts the low byte of a 16-bit word.

2.2 Program flow

Before starting programming a few steps must be completed.

First the DUP must be put in debug mode (the DUP is reset):

```
debug_init();
```

Clear DMA_PAUSE (bit 2) in debug configuration byte. This enables the DMA controller in the DUP.

```
debug_config = 0x22;
debug_command(CMD_WR_CONFIG, &debug_config, 1);
```

Switch to 32 MHz XOSC on the DUP. Programming at 32 MHz will be faster than programming at 16 MHz RCOSC.

```
write_xdata_memory(XREG_TO_INT(X_CLKCONCMD), 0x80);
while (read_xdata_memory(XREG_TO_INT(X_CLKCONSTA)) != 0x80);
```

Issue a FLASH_ERASE command on the debug interface and wait for it to finish. This step is not always required. All devices are initially erased when shipped to customer.

```
chip_erase();
```

With the flash memory in an erased state (all 1's) we can start programming the flash. The `init_flash_ptr()` sets the current flash image source address to start (0). Subsequent calls to `get_next_flash_byte()` (during programming) will return the byte at the current address and increase the address.

```
init_flash_ptr();
```

The call to `program_flash(16)` writes 16 KB to the flash. Data is transferred to the DUP in 1 KB blocks. In this example, the data written is taken from the constant byte array `flash_image`, declared in `flash_image.h`. The `program_flash()` function is explained in further detail in section 2.3. **Error! Reference source not found..**

```
program_flash(16);
```

A verification of the written flash memory is then performed. Each byte is read from the flash using the debug interface and checked. This is one way to verify the flash contents. A more efficient method is to run a CRC check on the flash memory with a program running on the DUP and executing code from SRAM.

```
init_flash_ptr();
num_bytes_ok = verify_flash(16384);
if (num_bytes_ok == 16384) {
    return 0; // Verification succeeded.
} else {
    return 1; // Verification failed.
}
```

2.3 Algorithm

The algorithm (in the form of a C function) for programming the flash is shown in Figure 3. An explanation of all the steps involved is outlined in this section.

First the four 8-byte DMA descriptors (`dma_desc` byte array) are written into the SRAM of the DUP starting at XDATA address `ADDR_DMA_DESC`.

In step 2, the `DMA1CFGH:DMA1CFGH` registers is set to `ADDR_DMA_DESC`. DMA channel 0 is not used.

In step 3, the start address for flash programming `FADDRH:FADDRL` is initialized to `0x0000`. This is not strictly necessary in all situations since the reset value is `0x0000`. However, it has been included here for safety and clarity.

Step 4 is the programming loop. The number of iterations is the number of buffers to program. The first step (4.1) of the loop is to arm one of the two DMA channels that transfer data from the `DBGDATA` register to SRAM buffer (`BUF0` for even iteration, `BUF1` for odd iteration). Then, a `BURST_WRITE` command transfers the buffer data to the DUP. The armed DMA channel will move the data from the burst write command into `BUF0/BUF1` during the `BURST_WRITE` command. In the first iteration of the loop (when `i == 0`), this will be the only thing going on in the DUP. In subsequent iterations flash programming (started in step 4.4) will execute in parallel with this step, i.e. two DMA channels will be active simultaneously.

Next, in step 4.2 a loop polls `FCTL.BUSY` (bit 7) until it is low. The `FCTL.BUSY` bit is high while the Flash Controller is busy programming. In an optimized implementation where the `BURST_WRITE` command transfers the data faster than the flash is programmed, this loop will spin a few times (waiting for the flash programming to finish) in each iteration except the first. The `wait` variable will be set to 1 in this case. If the flash programming is faster than `BURST_WRITE` `wait` will be 0 at the end of the while loop.

In step 4.3 the `max_speed` return status variable is cleared if the flash block programming finished before BURST_WRITE command completed and we are not in the first iteration. This step is optional and can be removed when an optimal solution has been implemented.

The programming is started in step 4.4. One of the DMA channels that transfer data from a buffer (BUF0 for even iteration, BUF1 for odd iteration) to FWDATA is armed and the flash write is started by writing 0x06 to FCTL (setting FCTL.WRITE bit).

In step 5, the programming loop has completed, but the flash write of the last buffer has just been started (in step 4.4). So, wait for that write operation to finish.

In step 6, the `max_speed` status variable is returned. This step is optional and can be removed.

```

uint8 program_flash(uint16 num_buffers) {
    uint8 dbg_arm, flash_arm;
    uint8 max_speed = 1;
    uint16 i;
    uint8 wait;

    // 1. Write the 4 DMA descriptors
    write_xdata_memory_block(ADDR_DMA_DESC, dma_desc, 32);

    // 2. Set the pointer to the DMA descriptors
    write_xdata_memory(XREG_TO_INT(X_DMA1CFGH), HIBYTE(ADDR_DMA_DESC));
    write_xdata_memory(XREG_TO_INT(X_DMA1CFL), LOBYTE(ADDR_DMA_DESC));

    // 3. Make sure FADDR is 0x0000
    write_xdata_memory(XREG_TO_INT(FADDRH), 0);
    write_xdata_memory(XREG_TO_INT(FADDRL), 0);

    // 4. Transfer buffers and program flash memory in parallel
    for (i = 0; i < num_buffers; i++) {
        // Set what is to be written to DMAARM, based on loop iteration
        if ((i & 0x0001) == 0) {
            dbg_arm = CH_DBG_TO_BUF0;
            flash_arm = CH_BUF0_TO_FLASH;
        } else {
            dbg_arm = CH_DBG_TO_BUF1;
            flash_arm = CH_BUF1_TO_FLASH;
        }

        // 4.1 transfer next buffer (first buffer when i == 0)
        write_xdata_memory(XREG_TO_INT(X_DMAARM), dbg_arm);
        burst_write();

        // 4.2 wait for write to finish
        wait = 0;
        while (read_xdata_memory(XREG_TO_INT(FCTL)) & 0x80) {
            wait = 1;
        }

        // 4.3 no waiting means programming finished before burst write
        if (i > 0 && wait == 0)
            max_speed = 0;

        // 4.4 start programming current buffer
        write_xdata_memory(XREG_TO_INT(X_DMAARM), flash_arm);
        write_xdata_memory(XREG_TO_INT(FCTL), 0x06);
    }

    // 5. Programming last buffer in progress, wait until done
    while (read_xdata_memory(XREG_TO_INT(FCTL)) & 0x80);

    // 6. Return efficiency status
    return max_speed;
}

```

Figure 3: program_flash() function

2.4 Verification

In the code example verification is done by reading out the flash contents one byte at a time using the debug interface and comparing it against the flash image in the programmer. A more efficient method is to run a CRC check on the flash contents using a program on the DUP executing from SRAM.

One alternative method for fast verification of correct programming is to calculate the CRC16 of each 2 kB flash page inside the DUP, and then read out. The CRC16 for the same page can be calculated on the Programmer (or have a pre-calculated look-up table) and if they match it is very likely that the programming was successful. If the two CRC16 checksums differ, the flash page on the DUP might need to be reprogrammed.

The 16-bit linear-feedback shift register (LFSR) in the CC253x's Random Number Generator can be used to calculate CRC16, which is described in [2], section 13.2.3.

It is possible to quickly transfer the data from the flash into the LFSR by using a high priority DMA channel in 8-bit block transfer mode. To be able to do this, the flash bank with the flash page to calculate checksum of must be mapped into the XBANK in XDATA memory space, because the DMA Controller only operates on XDATA memory. The XBANK mapping is controlled via MEMCTR.XBANK register field.

A DMA channel can be configured to copy flash contents from the flash mapped in XBANK into the Random Number Generator, i.e. the XDATA address for the RNDH register. Before the DMA is armed and then manually trigger it to start the transfer, it is important that the LFSR has been seeded with a known value by writing to RNDL twice.

When the DMA channel has completed its transfer, it is possible to read out the CRC16 checksum as the 16-bit combined value from the RNDH:RNDL registers.

3 References

- [1] CC2530 product page
<http://focus.ti.com/docs/prod/folders/print/cc2530.html>
- [2] CC253x User Guide
<http://www.ti.com/lit/pdf/swru191>
- [3] CC2530 Development Kit – CC2530DK
<http://focus.ti.com/docs/toolsw/folders/print/cc2530dk.html>

4 Document History

Version	Date	Description/Changes
0.5		Initial version

5 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 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.

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.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DSP	dsp.ti.com
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
Low Power Wireless	www.ti.com/lpw

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

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