TEXAS INSTRUMENTS

Application Report
SLAA103 - November 2000

MSP430 Flash Self-Programming Technique

Anton Muehlhofer *SSC MKT*

ABSTRACT

Flash self-programmability is becoming increasingly important. However, when accessing a flash-memory array for an erase/program operation, the CPU cannot simultaneously execute the code in the flash array. Thus, a microcontroller with only a single on-chip flash cannot execute code and modify its flash-memory contents at the same time.

The problem can be solved in two ways: (1) Instructions to erase/program flash memory are copied into RAM for execution by the CPU, and (2) The CPU is sent into an idle state while the flash memory erase/program process is being completed. This note describes how the flash erase/program can be done and presents the necessary software.

Contents

List of Figures

1 Introduction

Flash self-programmability is becoming increasingly important. However, when accessing a flash-memory array for an erase/program operation, the CPU cannot simultaneously execute the code in the flash array. Thus, a microcontroller with only a single on-chip flash cannot execute code and modify its flash-memory contents at the same time.

There are two approaches to solving the problem. In the first approach, the instructions to erase/program flash memory are copied into RAM for execution by the CPU. In the second, the CPU is sent into an idle state while the flash memory erase/program process is being completed.

All flash-based MSP430 devices from Texas Instruments incorporate a flash controller that allows the execution of code from the same flash module in which the software is concurrently modifying the data or re-programming code segments.

This note describes the way flash erase/program can be done, and the software necessary is presented.

2 Flash Module Implementation

The flash module of MSP430 consists of:

- *Control logic*: Machine-state and the timing-generator control of flash erase/program
- *Flash protection logic*: Protection against inadvertent erase/program operations
- *Programming voltage generator*: An integrated charge pump that provides all voltages required for flash erase/program

The three 16-bit control registers FCTL1, FCTL2, and FCTL3 control the complete Flash module and are shown in Figure 1. A more detailed description of these registers can be found in the description of the Flash Memory Module in the MSP430 User’s Guide, SLAU049 [1].

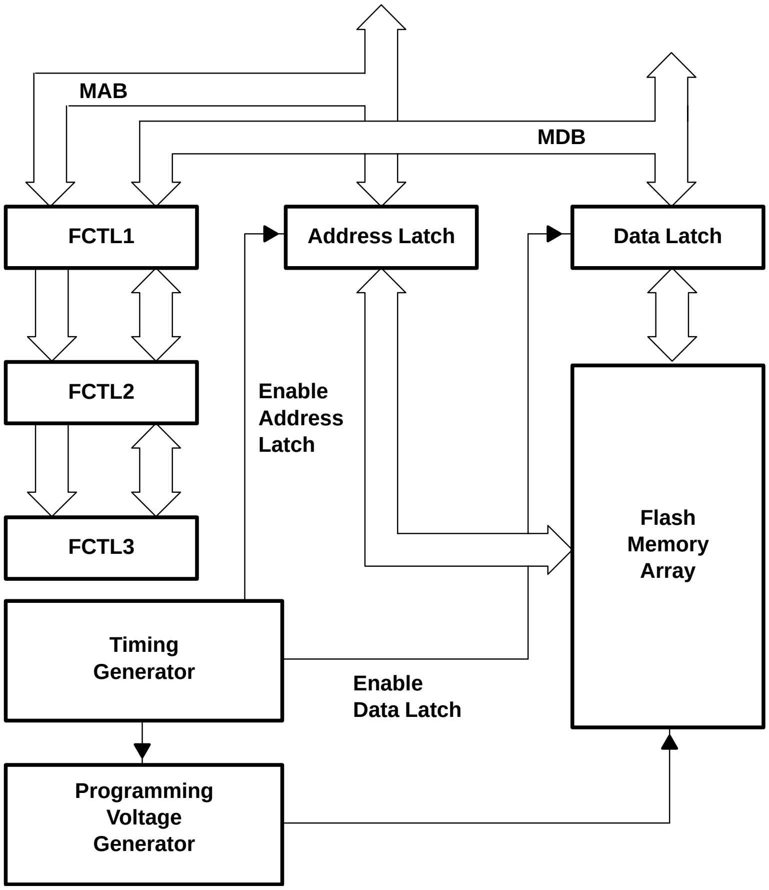

Figure 1. Flash Module Implementation

3 Erasing and Programming Flash

Usually the CPU reads the flash to access data or to execute a program. However, sometimes flash needs to be modified during program execution. During such a flash erase-and-program operation (i.e., flash reprogramming) the timing generator implemented in the flash module takes over control of flash. During this time, flash cannot be accessed by the CPU and, consequently, program instructions must come from some other place, such as RAM, or else the CPU must be sent into idle mode. After completing flash programming, the CPU again gains control of flash. Each of these two methods has its specific advantages.

Presently, only one flash module is implemented on MSP430 devices for program and data memory. This means that, during flash programming, the interrupt vectors are unavailable and no interrupt request can be served. Therefore, all possible interrupt sources, as well as the watchdog, must be disabled while flash is being modified. This can be done by executing the instructions shown in Figure 2 before starting flash programming.

```
dint                      ; disable all maskable interrupts
clr.b  &IE1               ; disable NMI, ACCV and OF interrupts
mov    #5A80h,&WDTCTL     ; disable Watchdog
```

Figure 2. Code to Disable Interrupts and Watchdog

3.1 Flash Reprogramming by Copying Into RAM

During flash programming, RAM is the only on-chip memory from which the CPU can access code for program execution. The software example in Figure 2 copies the flash program function onto the stack to execute code out of RAM while flash is unavailable. When flash becomes accessible again, the program counter once more points to flash memory, and the stack pointer is restored. The busy bit (bit 0 in FCTL3) indicates the accessibility status of flash memory. In *Appendix A: flash_ram.s43* three functions have been implemented for

- erase one segment
- program one byte
- program one word

Program execution from RAM keeps the CPU running while flash is being modified. This enables the MSP430 to program flash, for example, while still receiving data via UART. Of course, in this approach, the presence of a received character can only be detected by polling the UART receive flag.

```
MODULE  flash_ww
        PUBLIC  Flash_ww
        RSEG    CODE
#define   _CPU_ 5           /* 5=MSP430F1121, 6=MSP430F149 device */
#include  <Std_def.s43>
;***********************************************************************
;                          Flash write word
;***********************************************************************
; programs 1 word into the flash by copying a small function (18bytes)
; into the RAM by dynamic memory allocation onto the stack.
; function definition:  void Flash_ww( int *Data_ptr, int word )
; scratch register:     R12 = int *Data_ptr
;                       R14 = int word
;                       R13,R15 = general purpose
; Assumption: Flash is not busy if data are written into flash
(Flash_ww_start)
;***********************************************************************
Flash_ww
        dint                        ; prevent stack corruption
        clr.b  &IE1                 ; disable NMI, ACCV and OF interrupts
        mov    #5A80h,&WDTCTL       ; disable Watchdog
 mov   #Flash_ww_end,R13            ; define endaddress and lenght of
        mov   #Flash_ww_length,R15  ; function to be copied into RAM
        mov   #0A500h,&FCTL3        ; LOCK = 0
copy    push  @R13                  ; copy function into RAM
        decd  R13
        dec   R15
        jnz   copy
        mov   SP,R15
        mov   #0A540h,&FCTL1        ; WRT = 1
        call  R15                   ; call Flash write function in RAM
        mov   #0A500h,&FCTL1        ; WRT = 0
        mov   #0A510h,&FCTL3        ; LOCK = 1
        add   #2*Flash_ww_length,SP ; Stack housekeeping
        ret
;------------------------- Flash write function -----------------------
Flash_ww_start
        mov     R14,0(R12)      ; write databyte into Flash
wait_bf bit     #1,&FCTL3       ; wait for busy flag
        jnz     wait_bf
Flash_ww_end
        ret
; computation of word number of Flash write function to be copied into RAM
Flash_ww_length EQU (Flash_ww_end-Flash_ww_start+2)/2
        ENDMOD
```

Figure 3. Program to Copy Flash Program Function Onto Stack

3.2 Direct Flash Reprogramming

A unique feature of the MSP430 flash module is self-programmability without needing to copy the program into other memory. When the CPU fetches instructions from flash memory during flash reprogramming, flash returns 3FFFh (*JMP $)* to the CPU. This sends the CPU into an endless loop until flash reprogramming has been completed. On completion, flash returns the next instruction and program execution continues.

As can be seen from the brevity of the code in Figure 3, this is the easiest way to reprogram MSP430 flash memory. One disadvantage, however, is that the CPU is in idle mode during flash reprogramming, so no program can be executed and no interrupt can be processed. Furthermore, this flash reprogramming technique works only in byte/word-program mode (bit 7 in FCTL1 is zero), and the faster segment-write mode cannot be used. A more detailed example for erase and program flash memory using this feature is shown in *Appendix B: flash_idle.c* and in *Appendix C: flash_test.c*.

```
/***********************************************************************/
/*                         Flash_ww                                    */
/* programs 1 word (16 bits) into the flash memory                     */
/***********************************************************************/
void Flash_ww( int *Data_ptr, int word )
{
  FCTL3 = 0x0A500;          /* Lock = 0 */
  FCTL1 = 0x0A540;          /* WRT = 1 */
  *Data_ptr=word;           /* program Flash word */
  FCTL1 = 0x0A500;          /* WRT = 0 */
  FCTL3 = 0x0A510;          /* Lock = 1 */
}
```

Figure 4. Program to Idle CPU During Flash Reprogramming

4 Example Program

The two flash programming techniques have been implemented: (a) programming the flash out of RAM and (b) direct flash programming. The file *flash_ram.s43* uses the technique of flash programming from RAM; the program is written in assembly language and can be called from C. The file *flash_idle.c* uses the direct flash-reprogramming method. These files contain the same set of functions, namely:

- void Flash_wb(char *Data_ptr, char byte);
- void Flash_ww(int *Data_ptr, int word);
- void Flash_clr(int *Data_ptr);

Because the same calling conventions have been used, these two files can be exchanged easily just by interchanging them in the project (see Figure 5).

The example program needs three source files:

- *flash_ram.s43* or *flash_idle.c*
  Functions for flash programming using programming-out-of-RAM or direct flash self-programming

- *flash_test.c*
  Main program that calls flash programming functions
- *flash_var.s43*
  Example of how to define global C variables at an absolute address within information memory

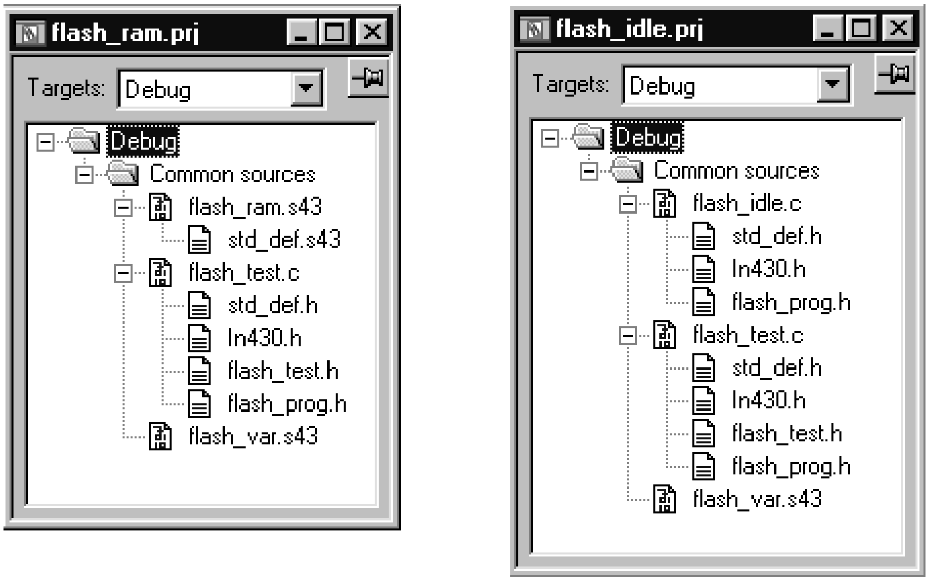

Figure 5. Project for Flash Programming-out-of-RAM and Direct Flash Programming

5 Summary

Two possible software solutions to reprogram data or code into flash memory while executing the code stored there have been described. In situations where the CPU must be able to respond quickly to an event, such as data communication via UART, use the method in which the flash program code is copied into RAM, but recognize that some software overhead is necessary to perform the copy. The simpler and more direct method is to put the CPU into idle mode during the flash reprogramming period (about 80 µs for programming one word, 12 ms for a segment erase).

6 References

[1] MSP430x1xx Family User’s Guide, SLAU049, 2000

[2] MSP430x11x1 Datasheet, SLAS241C, June 2000

[3] MSP430x13x,MSP430x14x Datasheet, SLAS272A, July 2000

Appendix A flash_ram.s43

```
;***********************************************************************
; flash_ram.c                                                2000-06-20
;
; this module contains functions necessary to program flash memory by
; running program out of RAM. The CPU is fully active during flash
; programming and could be used for example to still receive data from
; UART. The necessary copy function is contained in the dedicated flash
; program or erase function.
;
; Implemented functions are:
; --------------------------
; void Flash_wb( char *Data_ptr, char byte );
; void Flash_ww( int *Data_ptr, int word );
; void Flash_clr( int *Data_ptr );
;
; Anton Muehlhofer                        Texas Instruments Incorporated
;***********************************************************************
        MODULE  Flash_wb
        PUBLIC  Flash_wb
        RSEG    CODE
#define   _CPU_ 5           /* 5=MSP430F1121, 6=MSP430F149 device */
#include  <std_def.s43>     /* ports */
;***********************************************************************
;                          Flash write byte
;***********************************************************************
; programs 1 byte into the flash by copying a small function (18bytes)
; into the RAM by dynamic memory allocation onto the stack.
; function definition:  void Flash_wb( char *Data_ptr, char byte )
; scratch register:     R12 = char *Data_ptr
;                       R14 = char byte
;                       R13,R15 = general purpose
;***********************************************************************
Flash_wb
        dint                        ; prevent stack corruption
        mov   #Flash_wb_end,R13     ; define endaddress and lenght of
        mov   #Flash_wb_length,R15  ; function to be copied into RAM
        mov   #0A500h,&FCTL3        ; LOCK = 0
copy    push  @R13                  ; copy function into RAM
        decd  R13
        dec   R15
        jnz   copy
        mov   SP,R15
        mov   #0A540h,&FCTL1        ; WRT = 1
        call  R15                   ; call Flash write function in RAM
        mov   #0A500h,&FCTL1        ; WRT = 0
        mov   #0A510h,&FCTL3        ; LOCK = 1
        add   #2*Flash_wb_length,SP ; Stack housekeeping
        ret
;------------------------- Flash write function ------------------------
Flash_wb_start
        mov.b R14,0(R12)      ; write databyte into Flash
```

```
wait_bf bit   #1,&FCTL3       ; wait for busy flag
        jnz   wait_bf
Flash_wb_end
        ret
; computation of word number of Flash write function to be copied into RAM
Flash_wb_length EQU (Flash_wb_end-Flash_wb_start+2)/2
        ENDMOD
;========================================================================
        MODULE  flash_ww
        PUBLIC  Flash_ww
        RSEG    CODE
#define   _CPU_ 5           /* 5=MSP430F1121, 6=MSP430F149 device */
#include  <Std_def.s43>
;***********************************************************************
;                          Flash write word
;***********************************************************************
; programs 1 word into the flash by copying a small function (18bytes)
; into the RAM by dynamic memory allocation onto the stack.
; function definition:  void Flash_ww( int *Data_ptr, int word )
; scratch register:     R12 = int *Data_ptr
;                       R14 = int word
;                       R13,R15 = general purpose
;***********************************************************************
Flash_ww
        dint                        ; prevent stack corruption
        mov   #Flash_ww_end,R13     ; define endaddress and lenght of
        mov   #Flash_ww_length,R15  ; function to be copied into RAM
        mov   #0A500h,&FCTL3        ; LOCK = 0
copy    push  @R13                  ; copy function into RAM
        decd  R13
        dec   R15
        jnz   copy
        mov   SP,R15
        mov   #0A540h,&FCTL1        ; WRT = 1
        call  R15                   ; call Flash write function in RAM
        mov   #0A500h,&FCTL1        ; WRT = 0
        mov   #0A510h,&FCTL3        ; LOCK = 1
        add   #2*Flash_ww_length,SP ; Stack housekeeping
        ret
;------------------------- Flash write function ------------------------
Flash_ww_start
        mov     R14,0(R12)      ; write databyte into Flash
wait_bf bit     #1,&FCTL3       ; wait for busy flag
        jnz     wait_bf
Flash_ww_end
        ret
; computation of word number of Flash write function to be copied into RAM
Flash_ww_length EQU (Flash_ww_end-Flash_ww_start+2)/2
        ENDMOD
;========================================================================
        MODULE  Flash_clr
        PUBLIC  Flash_clr
        RSEG    CODE
#define   _CPU_ 5           /* 5=MSP430F1121, 6=MSP430F149 device */
#include <Std_def.s43>
;***********************************************************************
;                          Flash clear segment
;***********************************************************************
; erase 1 segment of the flash by copying a small function (xbytes)
; into the RAM by dynamic memory allocation onto the stack.
; function definition:  void Flash_clr( int *Data_ptr )
; scratch register:     R12 = int *Data_ptr
;                       R13,R15 = general purpose
;***********************************************************************
Flash_clr
        dint                        ; prevent stack corruption
        mov   #Flash_clr_end,R13    ; define endaddress and lenght of
        mov   #Flash_clr_length,R15 ; function to be copied into RAM
        mov   #0A500h,&FCTL3        ; LOCK = 0
copy    push  @R13                  ; copy function into RAM
        decd  R13
        dec   R15
        jnz   copy
        mov   SP,R15
        mov   #0A502h,&FCTL1        ; ERASE = 1
        call  R15                   ; call Flash write function in RAM
        mov   #0A500h,&FCTL1        ; ERASE = 0
        mov   #0A510h,&FCTL3        ; LOCK = 1
        add   #2*Flash_clr_length,SP ; Stack housekeeping
        ret
;------------------------- Flash clear function ------------------------
Flash_clr_start
        mov   #0,0(R12)             ; erase Flash segment
wait_bf bit   #1,&FCTL3             ; wait for busy flag
        jnz   wait_bf
Flash_clr_end
        ret
; computation of word number of Flash erase function to be copied into RAM
Flash_clr_length EQU (Flash_clr_end-Flash_clr_start+2)/2
        END
```

Appendix B flash_idle.c

```
/***********************************************************************/
/*  flash_idle.c                                            2000-06-20 */
/*                                                                     */
/*                  Flash erase and program functions                  */
/*                                                                     */
/* Below functions using the direct Flash programming algorithm.      */
/* After starting a flash write or erase cycle, the CPU will wait      */
/* until the flash is read-accessable again, so no program must be     */
/* copied into RAM. However, during flash programming, the CPU is      */
/* in ”idle” mode.                                                     */
/*                                                                     */
/* Note: Since all interrupt vectors are unavailable during flash      */
/* programming, all interrupts must be disabled.                       */
/*                                                                     */
/* Anton Muehlhofer                     Texas Instruments Incorporated */
/***********************************************************************/
#define   _CPU_ 5           /* 5=MSP430F1121, 6=MSP430F149 device */
#include  <std_def.h>       /* ports */
#include  ”flash_prog.h”    /* function prototypes */
/***********************************************************************/
/*                         Flash_wb                                    */
/* programs 1 byte (8 bit) into the flash memory                       */
/***********************************************************************/
void Flash_wb( char *Data_ptr, char byte )
{
  FCTL3 = 0x0A500;          /* Lock = 0 */
  FCTL1 = 0x0A540;          /* WRT = 1 */
  *Data_ptr=byte;           /* program Flash word */
  FCTL1 = 0x0A500;          /* WRT = 0 */
  FCTL3 = 0x0A510;          /* Lock = 1 */
}
/***********************************************************************/
/*                         Flash_ww                                    */
/* programs 1 word (16 bits) into the flash memory                     */
/***********************************************************************/
void Flash_ww( int *Data_ptr, int word )
{
  FCTL3 = 0x0A500;          /* Lock = 0 */
  FCTL1 = 0x0A540;          /* WRT = 1 */
  *Data_ptr=word;           /* program Flash word */
  FCTL1 = 0x0A500;          /* WRT = 0 */
  FCTL3 = 0x0A510;          /* Lock = 1 */
}
/***********************************************************************/
/*                         Flash_clr                                   */
/* erases 1 Segment of flash memory                                    */
/***********************************************************************/
void Flash_clr( int *Data_ptr )
{
  FCTL3 = 0x0A500;          /* Lock = 0 */
  FCTL1 = 0x0A502;          /* ERASE = 1 */
  *Data_ptr=0;              /* erase Flash segment */
  FCTL1 = 0x0A500;          /* ERASE = 0 */
  FCTL3 = 0x0A510;          /* Lock = 1 */
}
```

Appendix C flash_var.s43

```
; ****************************************************************************
; File:     flash_var.s43                                        30. May 2000
;
; defines 2 byte variables at a specific addresses in flash memory that can
; be accessed out of C as standard extern char variables (see ”flashd.h”):
;
; extern unsigned char SegA_last;       /* absolute address 010FFh */
; extern unsigned char SegB_last;       /* abosluet address 0107Fh */
;
; Anton Muehlhofer                             Texas Instruments Incorporated
; ****************************************************************************
;============================================================================
; extern unsigned char SegA_last;       /* absolute address 010FFh */
;============================================================================
        NAME    SegA_last
        PUBLIC  SegA_last
        ASEG    010FFh
SegA_last       DS 0
        ENDMOD
;============================================================================
; extern unsigned char SegB_last;       /* abosluet address 0107Fh */
;============================================================================
        NAME    SegB_last
        PUBLIC  SegB_last
        ASEG    0107Fh
SegB_last       DS 0
        ENDMOD

        END
```

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265