

COMPUTER ENGINEERING DESIGN II

Tutorial

MSP430F149 Flash Memory

Prepared by Zexin Pan
January 2005

This tutorial describes how to utilize MSP430F149 flash memory to store program data. By default, the program data (variables) are stored in RAM, which is quite limited in size. For example, MSP430F149 has only 2KB RAM. As program becomes complex, programmers need more memory to accommodate data (variables). The best solution is to use flash memory, which usually has much more space than RAM. For example, MSP430F149 has 60KB+256B Flash Memory. This flash memory can be allocated for both code and data.

1. Background

MSP430F149 has 60KB+256B Flash Memory which can be programmed via in-system by the CPU (user program). The flash memory is partitioned into multiple segments, each of which is either 128 bytes or 512 bytes, depending on which section it belongs to. There are two sections in the flash memory: the main and the information memory sections. There is no difference in the operation of the main and information memory sections. Code or data can be located in either section. The differences between the two sections are the segment size and the physical addresses. The information memory has two 128-byte segments. The main memory has two or more 512-byte segments. For example, MSP430F149 information memory has two 128-byte segments (A and B, 256B in total) and one hundred twenty 512-byte segments (0-119, 60KB in total).

Below is the “msp430F149C.xcl” file (XLINK command file for the ICC430 C-compiler and the MSP430F149) showing the segment address space in MSP430F149C.

```

// msp430F149C.XCL
//
// Texas Instruments, Revision: 1.1
//
// XLINK command file for the ICC430 C-compiler and the MSP430F149
//
// Usage: xlink your_file(s) -f msp430F149C
//
// The following segments are used by the compiler:
//
// Data read/write segments (RAM)
// =====
// segment      address range    max size      usage (compiler option)
// -----
// UDATA0       0200-09FF        2 Kbytes      Uninitialized variables
// IDATA0       0200-09FF        2 Kbytes      Initialized variables
// CSTACK       0200-09FF        2 Kbytes      Run-time stack/auto
variables
// ECSTR       0200-09FF        2 Kbytes      Writeable string
literals (-y)
//
// Program and non-volatile segments (FLASH)
// =====
// segment      address range    max size      usage (compiler option)
// -----
// INFO         1000-10FF        256 bytes     Information memory
// CODE         1100-FFDF        <60 Kbytes   Program code
// CONST        1100-FFDF        <60 Kbytes   Constant "const"
variables
// CSTR        1100-FFDF        <60 Kbytes   String literals (not -y)
// CDATA0      1100-FFDF        <60 Kbytes   Initializers for IDATA0
// CCSTR       1100-FFDF        <60 Kbytes   Initializers for ECSTR
(-y)
// INTVEC     FFE0-FFFF        32 bytes     Interrupt vectors (-e)
//
// Note:
// Option -y stores strings in ECSTR (init value in CCSTR) instead of
CSTR,
// Option -e enables language extensions
// Special function registers and peripheral modules occupy addresses
0-01FFh
//
// Define CPU

-cMSP430

// RAM
// Note: The stack is allocated from the top of RAM downward

-Z(DATA)UDATA0, IDATA0, ECSTR=0200-09FF
-Z(DATA)CSTACK#0200-0A00

// Information memory (FLASH)

-Z(CODE)INFO=1000-10FF

// Main memory (FLASH)

-Z(CODE)CODE, CONST, CSTR, CDATA0, CCSTR=1100-FFDF

// Interrupt vectors (FLASH)

-Z(CODE)INTVEC=FFE0-FFFF

```

2. Erase/write operation

The default mode of the flash memory is read mode. In order to erase/write the flash memory, programmers need to program the flash memory controller.

The procedure to initiate an erase operation from flash (which means that user program resides in the flash memory also) is shown below.

- Disable all interrupts and watchdog
- Set up flash timing generator for erase operation
- Unlock flash
- Enable segment erase
- Erase the flash segment
- Disable segment erase
- Lock flash
- Restore all interrupts and watchdog

Here is the Assembly code showing this flow of operations.

```
; Segment Erase from flash. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.

MOV  #WDTPW+WDTHOLD, &WDTCTL           ; Disable WDT
DINT                                     ; Disable interrupts
MOV  #FWKEY+FSSEL1+FN0, &FCTL2         ; SMCLK/2
MOV  #FWKEY, &FCTL3                    ; Clear LOCK
MOV  #FWKEY+ERASE, &FCTL1              ; Enable segment erase
CLR  &0FC10h                            ; Dummy write, erase S1
MOV  #FWKEY+LOCK, &FCTL3               ; Done, set LOCK
...                                      ; Re-enable WDT?
EINT                                     ; Enable interrupts
```

The procedures to initiate a byte/word write operation from flash (which means that user program resides in the flash memory also) is shown below.

- Disable all interrupts and watchdog
- Set up flash timing generator for write operation
- Unlock flash
- Enable segment write
- Write a byte/word to flash memory
- Disable flash write
- Lock flash
- Restore all interrupts and watchdog

Here is the Assembly code showing this flow of operations.

```

; Byte/word write from flash. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.

MOV  #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
DINT                                     ; Disable interrupts
MOV  #FWKEY+FSSEL1+FN0, &FCTL2   ; SMCLK/2
MOV  #FWKEY,&FCTL3                ; Clear LOCK
MOV  #FWKEY+WRT, &FCTL1          ; Enable write
MOV  #0123h,&0FF1Eh              ; 0123h -> 0FF1Eh
MOV  #FWKEY,&FCTL1                ; Done. Clear WRT
MOV  #FWKEY+LOCK,&FCTL3          ; Set LOCK
...                                    ; Re-enable WDT?
EINT                                     ; Enable interrupts

```

Note: More information about the flash memory can be found from “*MSP430X1XX Family User’s Guide*” by Texas Instruments, Chapter 5: *Flash Memory Controller*.

3. C example

The following program shows how to erase flash information memory segment A and B, followed by populating A with some number. Then copy content of segment A to that of segment B.

```

/*;*****
;   MSP430F1121 Flash Write example
;
;   Description; This program first erases flash segs A and B, then it
;   increments all values in segment A, then it copies segment A to ;
;   segment B.
;
;   Mike Mitchell
;   Texas Instruments, Inc
;   July 2001
;*****
*/

#include <msp430x14x1.h>

char value;                          // 8-bit value to write to segment A

// Function prototypes
void write_SegA (char value);
void copy_A2B (void);

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;        // Stop watchdog timer
    FCTL2 = FWKEY + FSSEL0 + FN0;    // MCLK/2 for Flash Timing Generator
    value = 0;                       // initialize value

    while(1)                          // Repeat forever
    {
        write_SegA(value++);         // Write segment A, increment value
        copy_A2B();                  // Copy segment A to B
    }
}

void write_SegA (char value)

```

```

{
    char *Flash_ptr;           // Flash pointer
    unsigned int i;

    Flash_ptr = (char *) 0x1000; // Initialize Flash pointer
    FCTL1 = FWKEY + ERASE;      // Set Erase bit
    FCTL3 = FWKEY;             // Clear Lock bit
    *Flash_ptr = 0;            // Dummy write to erase Flash segment

    FCTL1 = FWKEY + WRT;      // Set WRT bit for write operation

    for (i=0; i<128; i++)
    {
        *Flash_ptr++ = value; // Write value to flash
    }

    FCTL1 = FWKEY;           // Clear WRT bit
    FCTL3 = FWKEY + LOCK;    // Reset LOCK bit
}

void copy_A2B (void)
{
    char *Flash_ptrA;        // Segment A pointer
    char *Flash_ptrB;        // Segment B pointer
    unsigned int i;

    Flash_ptrA = (char *) 0x1000; // Initialize Flash segment A pointer
    Flash_ptrB = (char *) 0x1080; // Initialize Flash segment B pointer
    FCTL1 = FWKEY + ERASE;      // Set Erase bit
    FCTL3 = FWKEY;             // Clear Lock bit
    *Flash_ptrB = 0;           // Dummy write to erase Flash segment B
    FCTL1 = FWKEY + WRT;      // Set WRT bit for write operation

    for (i=0; i<128; i++)
    {
        *Flash_ptrB++ = *Flash_ptrA++; // copy value segment A to segment B
    }

    FCTL1 = FWKEY;           // Clear WRT bit
    FCTL3 = FWKEY + LOCK;    // Reset LOCK bit
}

```

If you want to declare some variables stored in the flash memory. Remember the C program for D437 LCD Demo board you tested in last semester? Here is a piece of code from that program showing how it works:

```

// Global "variables" (in flash-INFO).
// The following variables are allocated in the flash Information
// memory; they can be read directly, but the flash
// controller must be used to erase and write them. These variables are
// non-volatile.
#pragma memory=dataseg(INFO)
static word Refcal_flash ; // ADC12 reference calibration in Flash INFO
memory.
static word Temp_slope; // Temperature sensor slope calibration in Flash
INFO memory.
static word Temp_offset; // Temperature sensor offset calibration in Flash
INFO memory.
#pragma memory=default

```

...

```
void flash_write(word* address, word data) // Write the (integer) data  
to the addressed flash.
```

```
{  
    word gie = _BIC_SR(GIE) & GIE; // Disable interrupts.  
  
    FCTL3 = FWKEY;           // Unlock the flash.  
    FCTL1 = FWKEY | WRT;    // Enable flash write.  
    *address = data;        // Write the data to the flash.  
    FCTL1 = FWKEY;         // Disable flash write.  
    FCTL3 = FWKEY | LOCK;   // Lock the flash.  
    _BIS_SR(gie);          // Restore interrupts (to previous state).  
} // flash_write(word* address, data)
```

```
void flash_erase(word* address) // Erase the addressed flash segment.
```

```
{  
    word gie = _BIC_SR(GIE) & GIE; // Disable interrupts.  
  
    FCTL3 = FWKEY;           // Unlock the flash.  
    FCTL1 = FWKEY | ERASE;  // Enable flash segment erase.  
    *address = 0;          // Erase the flash segment.  
    FCTL1 = FWKEY;         // Disable flash segment erase.  
    FCTL3 = FWKEY | LOCK;   // Lock the flash.  
    _BIS_SR(gie);          // Restore interrupts (to previous state).  
} // flash_erase(word* address, data)
```