MSP430程序库<十五>Flash 控制器

一般，在单片机中的 Flash 存储器用于存放程序代码，属于只读型存储器。而在 MSP430 些列的单片机中，都可以通过内置的 Flash 控制器擦除或改写任何一段的内容。另外，msp430的单片机内部还专门留有一段 Flash 区域(information memory)，用于存放掉电后需要永久保存的数据。利用430内部的 Flash 控制器，可以完成较大容量的数据记录、用户设置参数在掉电后的保存等功能。

1.硬件介绍：

要对 Flash 读写，首先要了解 MSP430的存储器组织。430单片机的存储器组织结构采用冯诺依曼结构，RAM 和 ROM 统一编址在同一寻址空间中，没有代码空间和数据空间之分。

一般430的单片机都统一编址在0-64k 地址范围中，只有少数高端的型号才能突破64k（如：FG461x 系列）。绝大多数的 msp430单片机都编址在64kB 范围内。地址的大概编码方式如下：

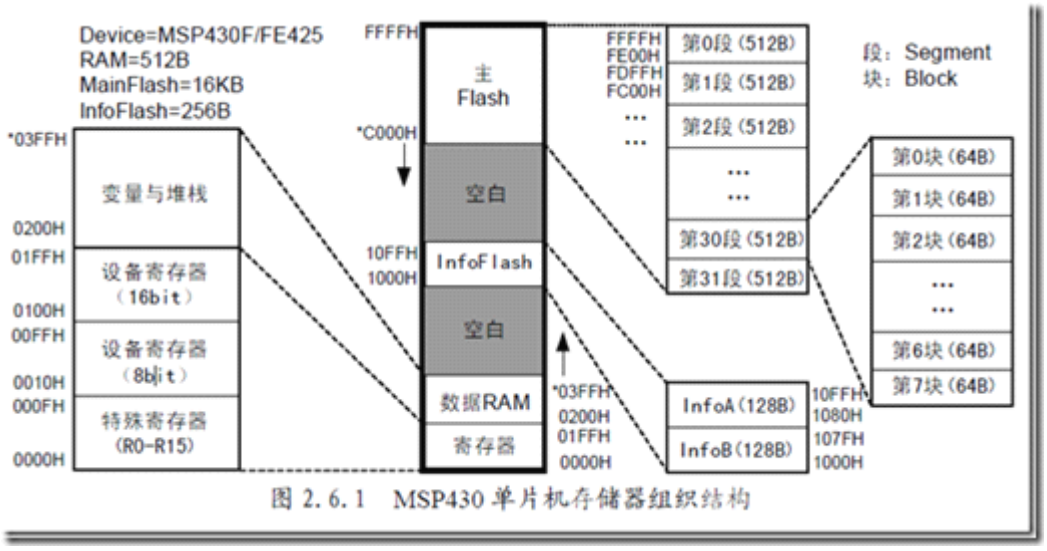


图 2.6.1　MSP430 单片机存储器组织结构

这是 msp430f425的存储器分配图，其他在64k 范围内的存储器的单片机编址方式与此类似：低256B 是寄存器区，然后是 RAM；空白；1000H 到10FFH 是信息 Flash 区；大于1100H-0FFFFH 是主存储器区(从0FFFFH 开始往低地址有单片机的主 Flash，多余的部分空白)。

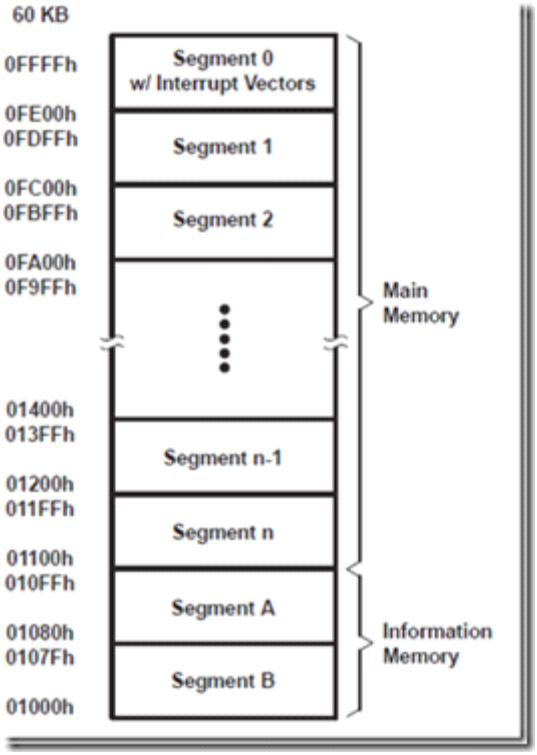
MSP430F14x 的 Flash 分布：

		MSP430F133	MSP430F135	MSP430F147 MSP430F1471	MSP430F148 MSP430F1481	MSP430F149 MSP430F1491
Memory Main: interrupt vector Main: code memory	Size Flash Flash	8KB 0FFFFh – 0FFE0h 0FFFFh – 0E000h	16KB 0FFFFh – 0FFE0h 0FFFFh – 0C000h	32KB 0FFFFh – 0FFE0h 0FFFFh – 08000h	48KB 0FFFFh – 0FFE0h 0FFFFh – 04000h	60KB 0FFFFh – 0FFE0h 0FFFFh – 01100h
Information memory	Size Flash	256 Byte 010FFh – 01000h	256 Byte 010FFh – 01000h	256 Byte 010FFh – 01000h	256 Byte 010FFh – 01000h	256 Byte 010FFh – 01000h
Boot memory	Size ROM	1KB 0FFFh – 0C00h	1KB 0FFFh – 0C00h	1KB 0FFFh – 0C00h	1KB 0FFFh – 0C00h	1KB 0FFFh – 0C00h
RAM	Size	256 Byte 02FFh – 0200h	512 Byte 03FFh – 0200h	1KB 05FFh – 0200h	2KB 09FFh – 0200h	2KB 09FFh – 0200h
Peripherals	16-bit 8-bit 8-bit SFR	01FFh – 0100h 0FFh – 010h 0Fh – 00h	01FFh – 0100h 0FFh – 010h 0Fh – 00h	01FFh – 0100h 0FFh – 010h 0Fh – 00h	01FFh – 0100h 0FFh – 010h 0Fh – 00h	01FFh – 0100h 0FFh – 010h 0Fh – 00h

MSP430F16x 的 Flash 分布：

		MSP430F167	MSP430F168	MSP430F169
Memory Main: interrupt vector Main: code memory	Size Flash Flash	32KB 0FFFFh – 0FFE0h 0FFFFh – 08000h	48KB 0FFFFh – 0FFE0h 0FFFFh – 04000h	60KB 0FFFFh – 0FFE0h 0FFFFh – 01100h
Information memory	Size Flash	256 Byte 010FFh – 01000h	256 Byte 010FFh – 01000h	256 Byte 010FFh – 01000h
Boot memory	Size ROM	1KB 0FFFh – 0C00h	1KB 0FFFh – 0C00h	1KB 0FFFh – 0C00h
RAM	Size	1KB 05FFh – 0200h	2KB 09FFh – 0200h	2KB 09FFh – 0200h
Peripherals	16-bit 8-bit 8-bit SFR	01FFh – 0100h 0FFh – 010h 0Fh – 00h	01FFh – 0100h 0FFh – 010h 0Fh – 00h	01FFh – 0100h 0FFh – 010h 0Fh – 00h

主 Flash 部分和信息 Flash 部分如下(60kB Flash 对应的单片机，如 msp430f149、msp430f149)：

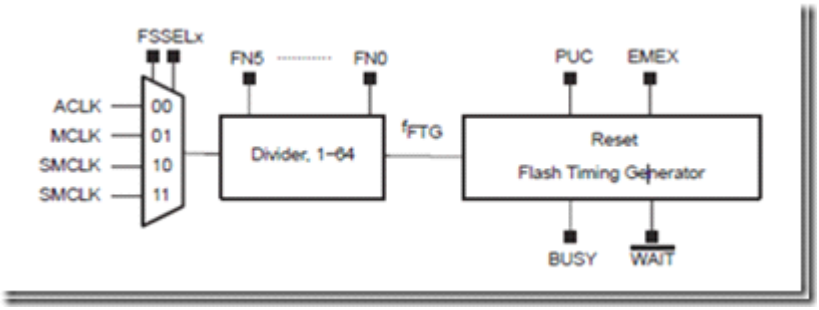


主 Flash 分为以512B 为段的单位，0段是单片机中断向量等程序入口地址，使用时不要擦除此段或改写此段，若要擦除或是改写，请先保存内容到 RAM 或其他段；主 Flash 各段内容均要避免写入或擦除，以免造成不可预料的后果。

信息 Flash 分为两段：段 A 和段 B，每段128B；可以保存用户自己的内容(主 Flash 也可以但是要避免与程序代码区冲突)；这里就把信息 Flash 的两段称为 InfoA(1080H-10FFh)和 InfoB(1000H-10FFH)。

Flash 的操作包括：字或字节写入；块写入；段擦除；主 Flash 擦除；全部擦除。任何的 Flash 操作都可以从 Flash 或从 RAM 中运行。

Flash 操作时需要时序发生器，Flash 控制器内部含有时序发生器用以产生所需的 Flash 时钟，Flash 时钟的范围必须在257kHz 到476kHz 之间。时序发生器的框图如下：



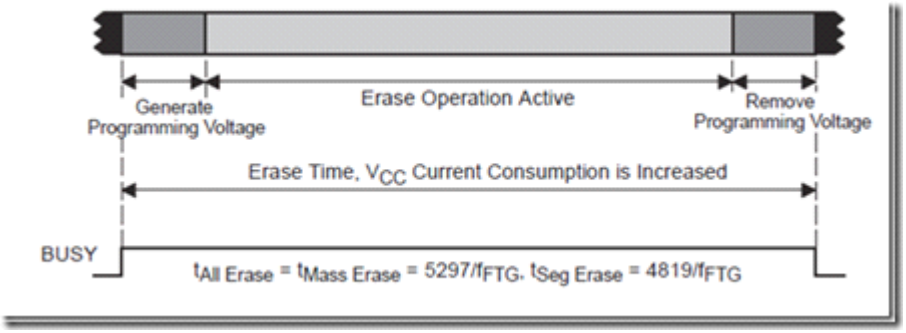
时序发生器可以选择 ACLK、MCLK、SMCLK 作为时钟源，通过分频获得所需的257kHz到476kHz 之间的 Flash 操作时钟。如果时钟频率不再这个范围内，将会产生不可预料的结果。

擦除：擦除之后，存储器中的 bit 都变为1；Flash 中的每一位都可以通过编程写入有1到0，但是要想由0变为1，必须通过擦除周期。擦除的最小单位是段。有三种擦除模式：

MERAS	ERASE	Erase Mode
0	1	Segment erase
1	0	Mass erase (all main memory segments)
1	1	Erase all flash memory (main and information .segments)

可以通过 MERAS、ERASE 位来设置擦除的模式：段擦除，主 Flash 擦除，全部擦除。

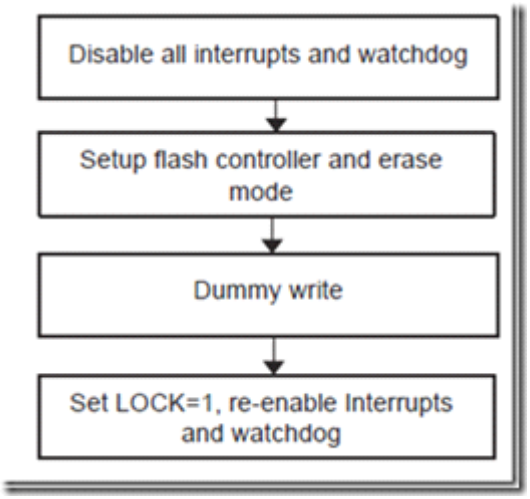
对要擦除段内的一个地址空写入启动擦出周期：空写入可以启动时序发生器和擦除操作。空写入后 BUSY 位立即变高直到擦除周期结束，这一位变为低(0)。BUSY, MERAS 和 ERASE 位在擦除周期结束后会自动复位。擦除周期的时间和要擦出的 Flash 大小无关，每次擦除的时间对于 MSP430F1xx 系系列单片机来说，所需时间是一样的。擦除的时序如下：



当空写入到的地址不在要擦除的段地址范围内的时候，空写入无效，直接被忽略。在擦除周期内，应该关中断，直到擦除完成，重新开中断，擦除期间的中断已经置标志位，开中断后立即响应。

从 Flash 中启动的擦除操作：擦除操作可以从 Flash 中启动或是从 RAM 中启动。当操作是从 Flash 中启动的时候，Flash 控制器控制了操作时序，CPU 运行被暂停直到擦除结束。擦除周期结束后，CPU 继续执行，从空写入之后的指令开始运行。当从 Flash 中启动擦除操作时，可以擦除即将运行的程序所在的段，如果擦除了即将运行的程序所在的 Flash 段时，擦除结束后，CPU 的运行不可预料。

从 Flash 启动时擦除周期如下：

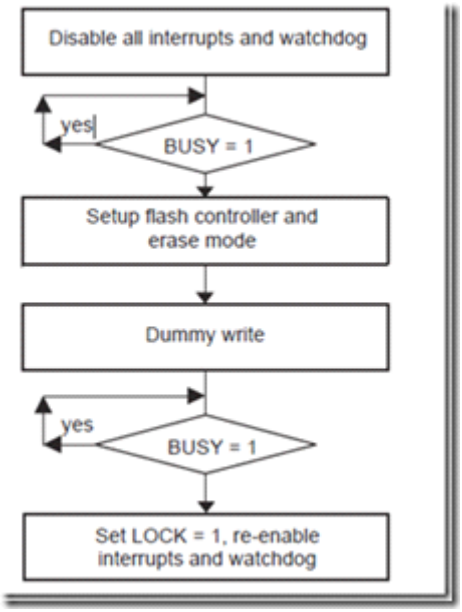


用户指南里面的示例汇编程序如下：

```
; 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
```

从 RAM 中启动擦除操作：任意擦除周期都可以从 RAM 启动，这时 CPU 不再暂停而是继续从 RAM 中运行接下来的程序。CPU 可以访问任何 Flash 地址之前，必须检查 BUSY 位以确定擦除周期结束。如果 BUSY = 1访问 Flash，这是一个访问冲突，这时 ACCVIFG 将被设置，而擦除的结果将是不可预测的的。

从 RAM 中启动擦除操作时，过程如下：



要在擦除之前确认没有访问 Flash，然后擦除完成之前不允许访问 Flash。

```
; Segment Erase from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
MOV #WDTPW+WDTHOLD,&WDTCTL  ; Disable WDT
DINT                        ; Disable interrupts
L1 BIT #BUSY,&FCTL3         ; Test BUSY
JNZ L1                      ; Loop while busy
MOV #FWKEY+FSSEL1+FN0,&FCTL2; SMCLK/2
MOV #FWKEY,&FCTL3           ; Clear LOCK
MOV #FWKEY+ERASE,&FCTL1     ; Enable erase
CLR &0FC10h                 ; Dummy write, erase S1
L2 BIT #BUSY,&FCTL3         ; Test BUSY
JNZ L2                      ; Loop while busy
MOV #FWKEY+LOCK,&FCTL3      ; Done, set LOCK
...                         ; Re-enable WDT?
EINT                        ; Enable interrupts
```

写 Flash 操作：写入的模式由 WRT 和 BLKWRT 位来确定：

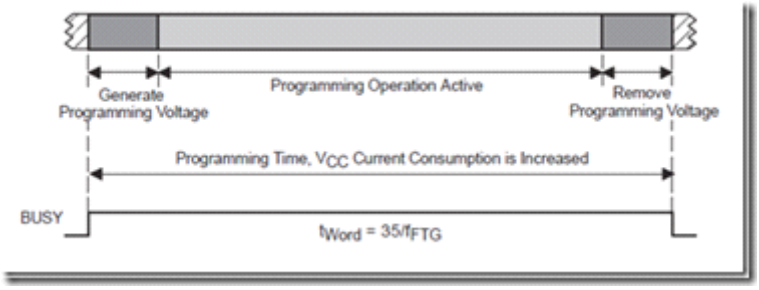
BLKWRT	WRT	Write Mode
0	1	Byte/word write
1	1	Block write

这两种模式中块写入大约是字或字节写操作时的两倍快，因为在块写入完成之前，变成电压一直维持直到块写入完成。同一个位置不能在擦除周期之前写入两次或以上，否则将发生数据损坏。写操作时，BUSY 位被置1，写入完成后，BUSY 被自动清零。如果写操作是从 RAM 发起的，在 BUSY=1时，程序不能访问 Flash，否则会发生访问冲突，置位 ACCVIFG，Flash 写入操作不可以预料。

字或字节写入：字或字节写入可以从 Flash 内部发起，也可以从 RAM 中发起。如果是从

Flash 中启动的写操作，时序将由 Flash 控制，在写入完成之前 CPU 运行将被暂停。写入完成后 CPU 将继续运行。

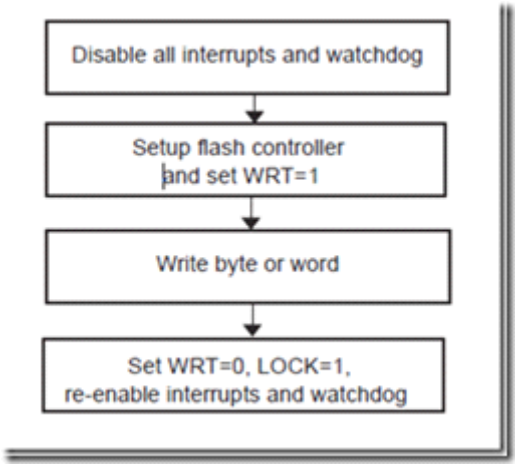
操作时序如下：



若是从 RAM 中启动写 Flash，程序将继续从 RAM 中运行。CPU 再次访问 Flash 之前必须确认 BUSY 位已经清零，否则会发生访问冲突，置位 ACCVIFG，写入的结果将不可预料。

字或字节写入模式下，内部产生的编程电压时适用于完整的64个字节块的写入
In byte/word mode, the internally-generated programming voltage is applied to the complete 64-byte block, each time a byte or word is written, for 32 of the 35 fFTG cycles. With each byte or word write, the amount of time the block is subjected to the programming voltage accumulates. The cumulative programming time, tCPT, must not be exceeded for any block. If the cumulative programming time is met, the block must be erased before performing any further writes to any address within the block.

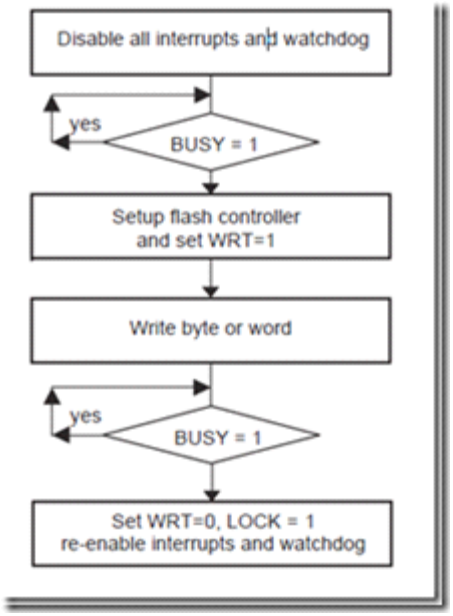
从 Flash 发起写字节或字时：




```
; 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
```

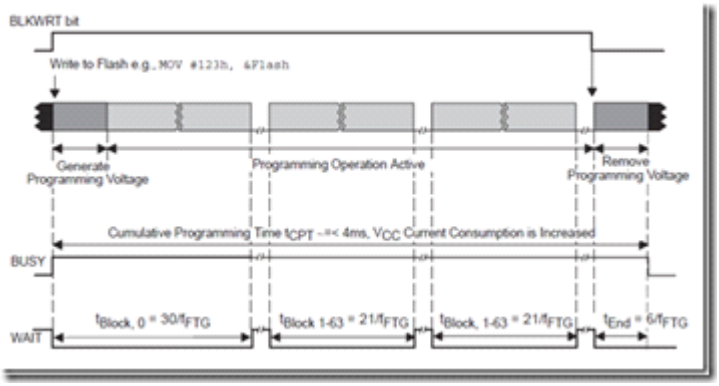
```
MOV #FWKEY,&FCTL1                  ; Done. Clear WRT
MOV #FWKEY+LOCK,&FCTL3               ; Set LOCK
...                                ; Re-enable WDT?
EINT                                 ; Enable interrupts
```

从 RAM 中启动写入操作时：

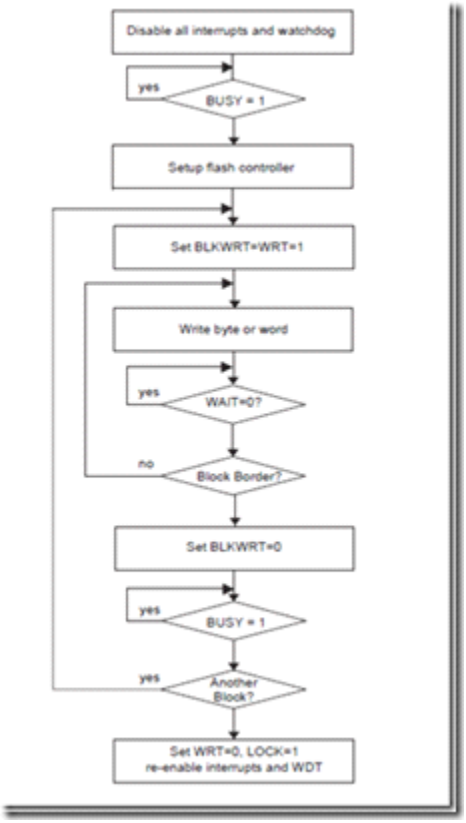



```
; Byte/word write from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
MOV #WDTPW+WDTHOLD,&WDTCTL         ; Disable WDT
DINT                               ; Disable interrupts
L1 BIT #BUSY,&FCTL3                  ; Test BUSY
JNZ L1                             ; Loop while busy
MOV #FWKEY+FSSEL1+FN0,&FCTL2       ; SMCLK/2
MOV #FWKEY,&FCTL3                    ; Clear LOCK
MOV #FWKEY+WRT,&FCTL1                ; Enable write
MOV #0123h,&0FF1Eh                   ; 0123h -> 0FF1Eh
L2 BIT #BUSY,&FCTL3                  ; Test BUSY
JNZ L2                             ; Loop while busy
MOV #FWKEY,&FCTL1                    ; Clear WRT
MOV #FWKEY+LOCK,&FCTL3               ; Set LOCK
...                                ; Re-enable WDT?
EINT                                 ; Enable interrupts
```

块写入：当需要写入连续的多个字或字节时，块写入能够能够提高 Flash 访问速度。块写入时，内部产生的编程电压一直存在在64个字节的块写入期间。块写入不能有 Flash 存储器内启动，必须从 RAM 中发起块写入操作。块写入期间，BUSY 位被置位。在写入每个字节或字时必须检测 WAIT 位。下一个字或字节能够被写入时， WAIT 位被置位。



块写入的过程如下：




```
; Write one block starting at 0F000h.
; Must be executed from RAM, Assumes Flash is already erased.
; 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
MOV #32,R5                          ; Use as write counter
MOV #0F000h,R6                      ; Write pointer
MOV #WDTPW+WDTHOLD,&WDTCTL          ; Disable WDT
DINT                                ; Disable interrupts
L1 BIT #BUSY,&FCTL3                 ; Test BUSY
   JNZ L1 ; Loop whilebusy
MOV #FWKEY+FSSEL1+FN0,&FCTL2        ; SMCLK/2
MOV #FWKEY,&FCTL3                   ; Clear LOCK
MOV #FWKEY+BLKWRT+WRT,&FCTL1        ; Enable block write
```

```
L2 MOV Write_Value,0(R6)          ; Write location
L3 BIT #WAIT,&FCTL3               ; Test WAIT
JZ L3                            ; Loop while WAIT=0
INCD R6                           ; Point to next word
DEC R5                            ; Decrement write counter
JNZ L2                           ; End of block?
MOV #FWKEY,&FCTL1                    ; Clear WRT,BLKWRT
L4 BIT #BUSY,&FCTL3               ; Test BUSY
JNZ L4                           ; Loop while busy
MOV #FWKEY+LOCK,&FCTL3                ; Set LOCK
...                              ; Re-enable WDT if needed
EINT                              ; Enable interrupts
```

当任何写或擦除操作是从 RAM 启动，而 BUSY = 1，CPU 不能读取或写入或从任何 Flash 位置。否则，发生访问冲突，ACCVIFG 设置，结果是不可预知的。此外，如果闪存写入让 WRT= 0，ACCVIFG 中断标志设置，Flash 不受影响。

如果写入或擦除操作时从Flash 启动的，CPU 访问下一条指令时(从 Flash 读取指令)，Flash 控制器返回03FFFH 给 CPU；03FFFH 是指令 JMP PC，这让 CPU 一直循环直到 Flash 操作完成。Flash 写入或擦除操作完成后，允许 CPU 继续访问接下来的指令。

当 BUSY=1时，Flash 访问时：

Flash Operation	Flash Access	WAIT	Result
Any erase, or Byte/word write	Read	0	ACCVIFG = 0. 03FFFh is the value read
	Write	0	ACCVIFG = 1. Write is ignored
	Instruction fetch	0	ACCVIFG = 0. CPU fetches 03FFFh. This is the JMP PC instruction.
Block write	Any	0	ACCVIFG = 1, LOCK = 1
	Read	1	ACCVIFG = 0, 03FFFh is the value read
	Write	1	ACCVIFG = 0, Flash is written
	Instruction fetch	1	ACCVIFG = 1, LOCK = 1

在开始 Flash 操作之前，需要停止所有的中断源。如果在 Flash 操作期间有中断响应，读中断服务程序的地址时，将收到03FFFH 作为中断服务程序的地址。如果 BUSY=1；CPU 将一直执行难 IMP PC 指令；Flash 操作完成后，将从03FFFH 执行中断服务程序而不是正确的中断程序的地址。

停止写入或擦除：任何写入和擦除操作都可以在正常完成之前，通过设置紧急退出位 EMEX 退出操作。设置 EMEX 时，立即停止当前活动的操作，停止 Flash 控制器；所有的 Flash 操作停止，Flash 返回可读模式，FCTL1的所有位复位；操作的结果不可预料。

设置和访问 Flash 控制器：FCTLx 是16位的、密码保护的、可读写的寄存器。写入这些寄存器都必须在高位包含密码0A5H，如果写入的不是0A5H，将会引起复位。读寄存器时高位读出的是96H。

在擦除或写入字或字节时写 FCTL1寄存器将会引起访问冲突，置位 ACCVIFG.块写入时，WAIT=1时可以写 FCTL1寄存器，当 WAIT=0时写 FCTL1寄存器是访问冲突，置位

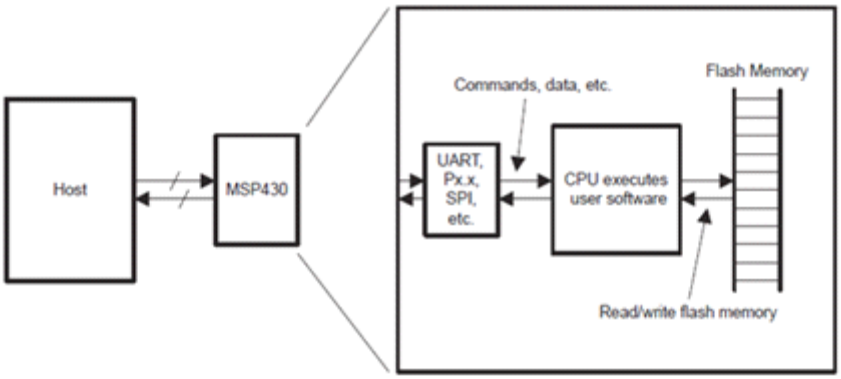
ACCVIFG。BUSY=1时，所有写入 FCTL2寄存器都是访问冲突。BUSY=1时，所有的 FCTLx 都可以读操作，不会引起访问冲突。

Flash 的中断：Flash 控制器有两个中断源：KEYV, 和 ACCVIFG。ACCVIFG 在访问冲突的时候被置位。当 ACCVIE 在 Flash 操作完成后被重新使能后 ACCVIFG 会引起中断请求。ACCIFG 和 NMI 同样的中断向量，所以这个中断不需要 GIE 位允许即可产生中断请求。必须通过软件检测 ACCVIFG 位，以确定发生了访问冲突；ACCVIFG 位必须软件复位。KEYV 是关键值错误当写 Flash 的寄存器时没有写正确的高位密码时被置位，这是会立刻引起 PUC 信号复位整个硬件。

编程 Flash 的硬件：编程430的 Flash 内容有三种选择，通过 JTAG、通过 BSL 和用户定制。用户定制即是通过单片机的程序访问自己的 Flash。

The ability of the MSP430 CPU to write to its own flash memory allows for in-system and external custom programming solutions as shown in Figure 5–12. The user can choose to provide data to the MSP430 through any means available (UART, SPI, etc.). User-developed software can receive the data and program the flash memory. Since this type of solution is developed by the user, it can be completely customized to fit the application needs for programming, erasing, or updating the flash memory.

Figure 5–12. User-Developed Programming Solution



Flash 的寄存器列表如下：

Register	Short Form	Register Type	Address	Initial State
Flash memory control register 1	FCTL1	Read/write	0128h	09600h with PUC
Flash memory control register 2	FCTL2	Read/write	012Ah	09642h with PUC
Flash memory control register 3	FCTL3	Read/write	012Ch	09618h with PUC
Interrupt Enable 1	IE1	Read/write	000h	Reset with PUC

Flash 的硬件部分就介绍这么多了，有什么不大懂的地方请参考 TI 提供的用户指南。

2.程序实现：

首先设置 Flash 的时钟，初始化 Flash 控制器：

```
voidFlashInit()
{
 FCTL2 = FWKEY + FSSEL_2 + FN1;            // 默认 SMCLK/3 =333KHz
}
```

这个函数仅仅设置了时钟。

擦除函数：

```
voidFlashErase(unsigned intAddr)
{
    char*FlashPtr;
    FlashPtr = (char*)Addr;
 FCTL1 = FWKEY + ERASE;                      // Set Erase bit
FCTL3 = FWKEY;                              // Clear Lock bit
   DINT;
 *FlashPtr = 0;                             // Dummy write to erase Flash segment B
WaitForEnable();                            //Busy
   EINT;
 FCTL1 = FWKEY;                             // Lock
FCTL3 = FWKEY + LOCK;                       // Set Lock bit
}
```

这个和上面给出的流程一样，参数是要被擦除的段的首地址。WaitForEnable 函数等等待 BUSY 标志变回零即操作完成。

```
voidWaitForEnable()
{
 while((FCTL3 & BUSY) == BUSY);        //Busy
}
```

写入字节：

```
voidFlashWriteChar(unsigned intaddr,charData)
{
 char*FlashPtr = (char*)addr;               // Segment A pointer
FCTL1 = FWKEY + WRT;                        // Set WRT bit for write operation
FCTL3 = FWKEY;                              // Clear Lock bit
   DINT;
 *FlashPtr = Data;                          // Save Data
WaitForEnable();                            //Busy
   EINT;
 FCTL1 = FWKEY;                             // Clear WRT bit
FCTL3 = FWKEY + LOCK;                       // Set LOCK bit
}
```

写入字：

```
voidFlashWriteWord(unsigned intaddr,unsigned intData)
```

```
{
    unsigned int*FlashPtr = (unsigned int*)addr;
  FCTL1 = FWKEY + WRT;                          // Set WRT bit for write
operation
  FCTL3 = FWKEY;                               // Clear Lock bit
    DINT;
  *FlashPtr = Data;                            // Save Data
  WaitForEnable();                            //Busy
    EINT;
  FCTL1 = FWKEY;                               // Clear WRT bit
  FCTL3 = FWKEY + LOCK;                        // Set LOCK bit
}
```

写入字或字节两个函数差别仅仅是指针类型不同。

读取字或字节：

```
charFlashReadChar(unsigned intAddr)
{
    charData;
    char*FlashPtr = (char*) Addr;
    Data = *FlashPtr;
    return(Data);
}

unsigned intFlashReadWord(unsigned intAddr)
{
    unsigned intData;
    unsigned int*FlashPtr = (unsigned int*)Addr;
    Data = *FlashPtr;
    return(Data);
}
```

这两个函数的差别也是仅仅指针类型不同。

这些函数和前面硬件介绍部分的程序流程相同，这里不再详细说明。

3.使用示例：

使用方法和之前的一样，工程中加入 C 文件，源代码文件中文件包含 H 文件，即可使用，具体参考示例项目：

演示主要程序主要如下：

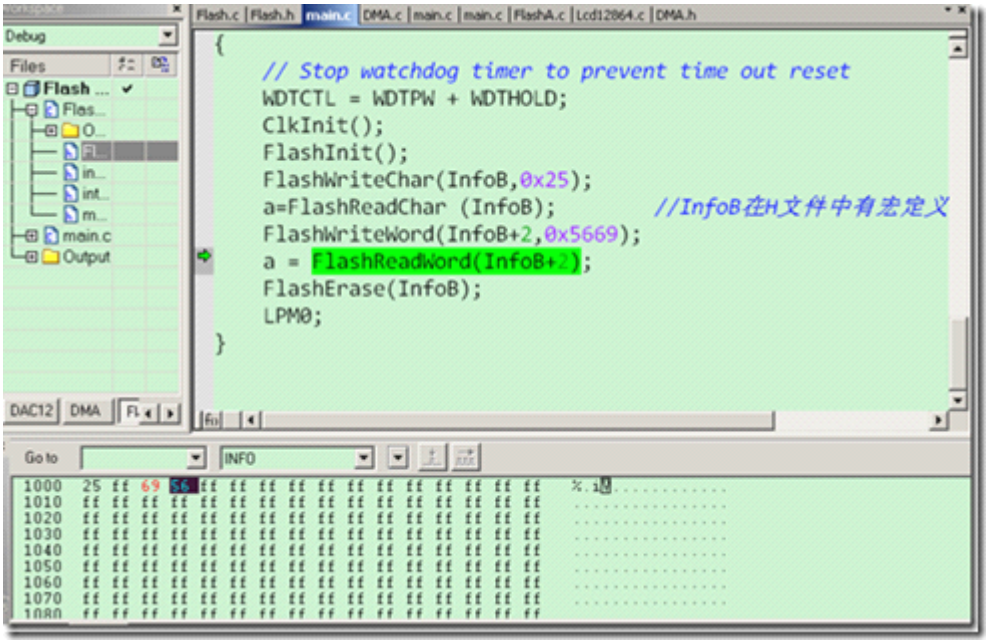
```
#include<msp430x16x.h>
#include"Flash.h"

inta;
voidmain( void)
{
  // Stop watchdog timer to prevent time out reset
```

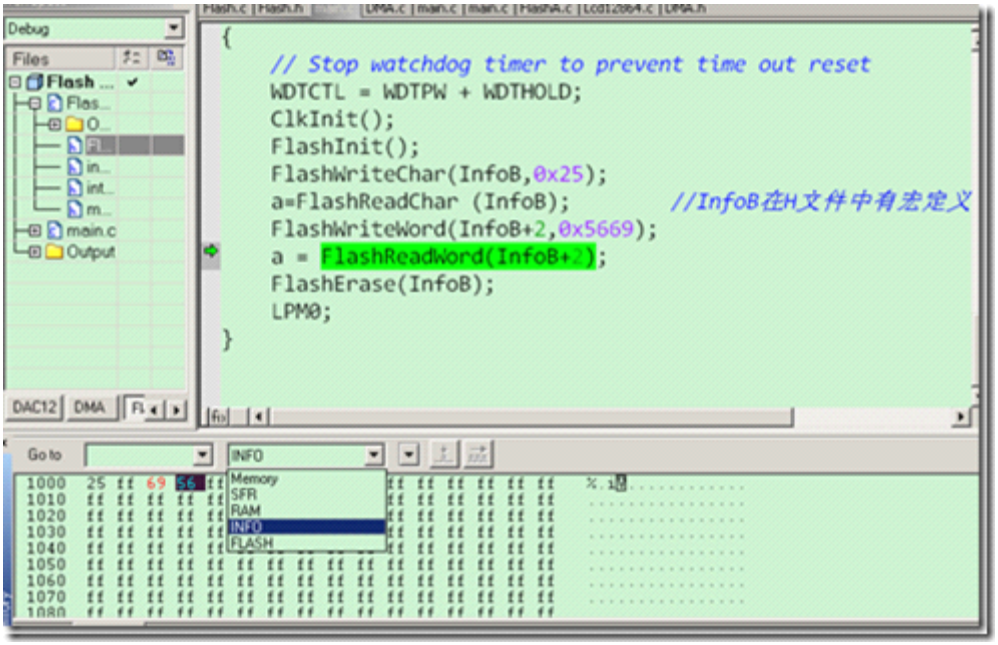
```
    WDTCTL = WDTPW + WDTHOLD;
     ClkInit();
     FlashInit();
     FlashWriteChar(InfoB,0x25);
  a=FlashReadChar (InfoB);          //InfoB 在 H 文件中有宏定义
    FlashWriteWord(InfoB+2,0x5669);
     a = FlashReadWord(InfoB+2);
     FlashErase(InfoB);
     LPM0;
}
```

这里向 InfoB(1000h)首地址开始写数据，先写一个字节 再写入一个字（注意写入字时，必须是偶数地址，奇数地址会写在这个地址所在的前一个偶数地址），读出，然后擦除。这里的程序都是在 Flash 中运行的，没有演示 RAM 中运行的程序。如果在 RAM 运行程序，则需要先把程序从 Flash 中移到 RAM 中，然后跳转到 RAM 中运行。

调试截图如下：



调试时，view-Memory 菜单，调出存储器窗口；观察 Flash 内容。



这里写入的是 Info Flash 部分，观察这部分的结果，和写入的结果同。

Flash 程序控制器的程序就到这儿了。Flash 可以用于存储长期保存的数据。

相关文章及附件下载：http://www.ideyi.org/bbs/article_1077_374381.html