Chapter 13

Flash Programming

To use the MSP430 in a stand-alone embedded application, the application code needs to be stored in flash memory. The MSP430 flash memory module is bit-, byte-, and word-addressable and programmable, using a controller that supervises the programming and erase operations.

The controller has three (or four) registers, a timing generator, and a voltage generator to supply program and erase voltages. This chapter covers flash memory module operation and segmentation, and finishes with a laboratory exercise.

Topic **Page**

13.1 Introduction

Memory, in general, is broadly classified as read-only memory (ROM) or random-access memory (RAM). Flash memory is a hybrid of ROM and RAM.

Flash memory is:

- ❑ Low cost;
- ❑ Electrically programmable;
- ❑ Fast to read from;
- ❑ High density;
- ❑ Reliable;
- ❑ Does not lose its configuration when the power is removed.

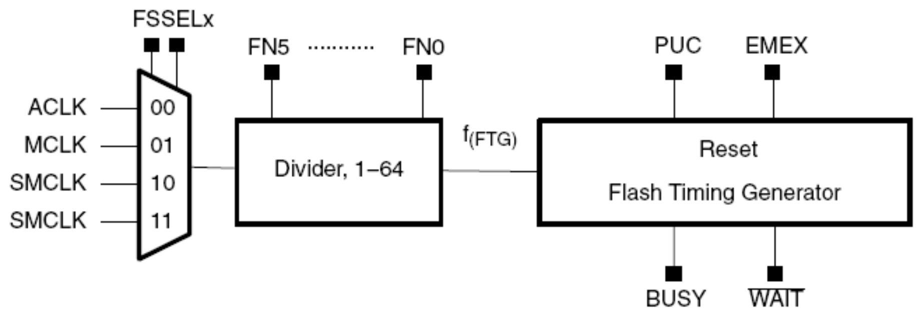
For these reasons, flash memory is one of the most popular technologies for storing program code and constant data values. Devices in the MSP430 family of microcontrollers have flash memory integrated onto them.

The MSP430Fxxx(x) devices handle the flash memory structure as a series of segments, allowing bit-, byte-, and word-addressing and programming. However, the flash memory must be erased in segments.

The flash memory module has an integrated controller that:

- ❑ Controls programming and erase operations;
- ❑ Has three or four registers (see the device-specific data sheet);
- ❑ Has a timing generator (see *Figure 13-1*):
  - ■ Can be sourced from ACLK, SMCLK, or MCLK;
  - ■ Flash timing generator operating frequency: ~ 257 kHz < $f_{(FTG)}$ < ~ 476 kHz (see device-specific data sheet);
  - ■ The selected clock source should be divided using the FNx bits, to meet the frequency requirements for $f_{(FTG)}$.
- ❑ Has a voltage generator, to supply program and erase voltages (must be stable).

Figure 13-1. Flash memory timing generator block diagram.



An MSP430 flash device can be programmed via:

- ❑ JTAG interface (requires four signals, ground and optionally VCC and RST/NMI);
- ❑ Bootstrap Loader (using a UART serial interface);
- ❑ Custom solution (using one of the interfaces available and through user developed software).

For more information, browse the TI web pages to find some Application Reports on the flash memory physics, recommendations for correct handling and user applications. Amongst these, it is important to highlight the contents of the following reports that are included in Annex E:

- ❑ MSP430 flash memory characteristics <slaa334a.pdf>
  - ■ Explanation of the physics behind MSP430 data-sheet specifications and recommendations for correct MSP430 flash handling.

- ❑ Understanding MSP430 flash data retention <slaa392.pdf>
  - ■ Discusses in detail the data retention for the MSP430 flash. Presents the effects of high temperatures on flash data retention.

- ❑ Features of the MSP430 bootstrap loader <slaa089d.pdf>
  - ■ Presents the features of the bootstrap loader (BSL), the specific command sequence applied to specific device pins, followed by an added sequence of commands to initiate the bootstrap loader function. The BSL is an external interface, similar to the JTAG, which may be used to program flash memory. Its main features are:
    - o The BSL code is stored in a special section of ROM (cannot be overwritten by other applications);
    - o It is triggered by toggling the TCK pin on the JTAG port;
    - o It uses the UART communication protocol (9600 baud);
    - o Performs essentially the same functions as the JTAG interface, with the exception of the security fuse programming.

- ❑ Application of the bootstrap loader in MSP430 with flash hardware and software proposal <slaa096.pdf>
  - ■ Describes a simple and low-cost hardware and software (C code) solutions to access the bootstrap loader functions of the MSP430 flash devices via the serial port (RS-232) of a PC.

- ❑ Solid state voice recorder using flash MSP430 <slaa123.pdf>
  - ■ The development of a solid state voice recorder (analogue voice pattern conversion to digital data) and storing it real-time in the MSP430 flash memory, allowing playback. It demonstrates the flexibility of the in-system programmable flash memory.

- ❑ Programming a flash-based MSP430 using the JTAG interface <slaa149d.pdf>
  - ■ Details the functions required to erase, program, and verify the MSP430's flash memory module using the JTAG communication port, as well as how to program the JTAG access security fuse. The security fuse is a one-time only burn, i.e., once it has been programmed, further flash memory writes and erasures are impossible. It not only restricts flash programming, but also prohibits further JTAG access.

- ❑ A Flash monitor for the MSP430 <slaa341.pdf>
  - ■ Development of a flash monitor program to evaluate the device memory and update the flash contents with new application code via a universal synchronous / asynchronous receive/transmit (USART) peripheral interface.

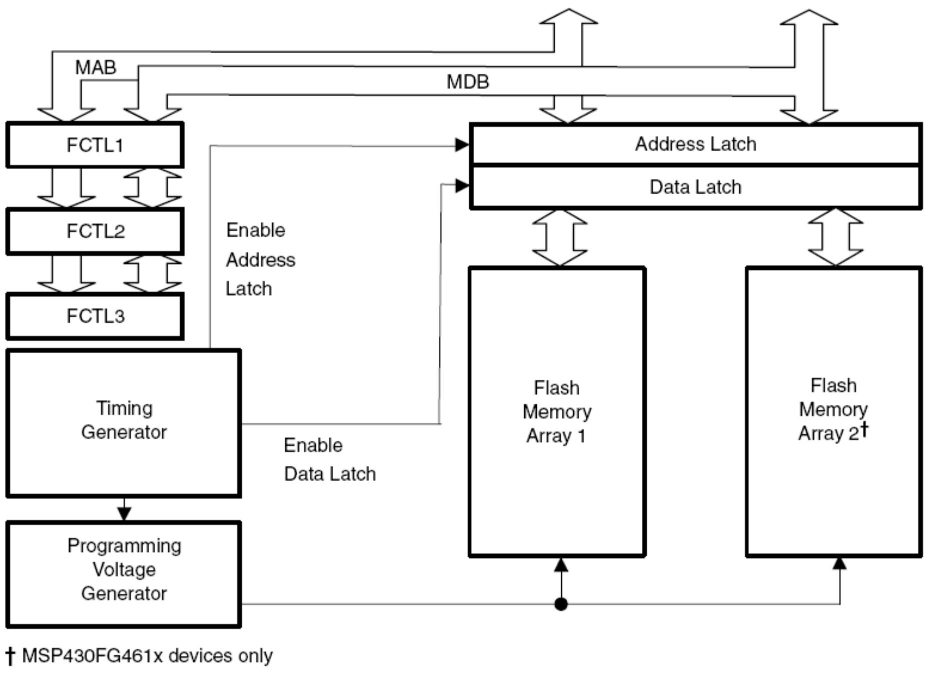
13.2 Flash memory operation and segmentation

MSP430 flash memory block diagram is shown in *Figure 13-2*. The MSP430FG4618 device present on the Experimenter's board has two flash memory arrays.

All flash memory is partitioned into segments:

- ❑ Write: single bits, bytes, or words;
- ❑ Erase: by segment.

Figure 13-2. Flash memory block diagram.



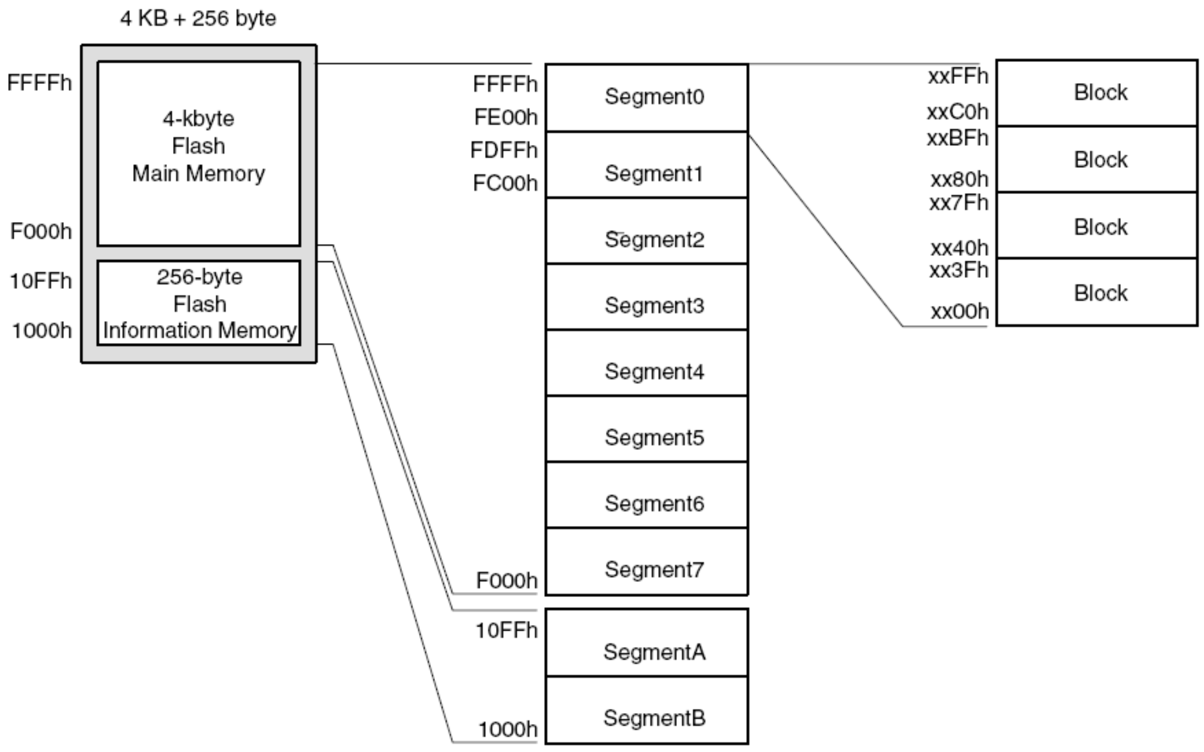
The flash memory is partitioned into (see the device-specific data sheet):

- ❑ Main memory section (two or more 512-byte segments);
- ❑ Information memory section (two 128-byte segments), located at lower memory addresses immediately following the RAM address space.

The operation is the same in these memory sections and code or data can be located in either section.

The flash memory segmentation into main and information segments and then into blocks is shown in *Figure 13-3*.

Figure 13-3. 4 KB Flash memory segmentation: 8 main segments and 2 information segments.



The SegmentA (information A) of the 2xx family devices (used by the eZ430-F2013 and eZ430-RF2500) of the information memory is locked separately from all other segments with the LOCKA bit (toggle state):

- LOCKA = 1:
  - SegmentA cannot be written or erased;
  - All information memory is protected from erasure during a mass erase or production programming.

- LOCKA = 0:
  - SegmentA can be erased and written;
  - All information memory is erased during a mass erase or production programming.

Flash memory write/erase modes

The default mode is the read mode. In this mode, Flash memory operates identically to ROM:

- Flash memory is not erased or written;
- Flash timing generator is off;
- Voltage generator is off.

The Flash memory write/erase modes are selected by the BLKWRT, WRT, GMERAS, MERAS, and ERASE bits.

To stop any write or erase operation before its normal completion, set the EMEX bit. When EMEX = 1:

- All flash operations cease;
- The flash returns to read mode;
- All bits in the FCTL1 register are reset.

Erase modes

Any erase cycle can be initiated from within flash memory or from RAM.

- Initiated from within flash memory:
  - All timing is controlled by the flash controller;
  - CPU is held while the erase cycle completes (dummy write);
  - CPU resumes code execution after the erase cycle finishes.
- Initiated from RAM:
  - CPU is not held and (can continue to execute code from RAM);
  - CPU can access any flash address again when BUSY = 0 (end of the erase cycle).

The erase modes are shown in *Table 13-1*.

Table 13-1. Erase modes.

Bits				Mode description	
GMERAS[1]	LOCKA[2]	MERAS	ERASE	MSP430FG461x	MSP430F2xxx
X	-	0	1	Segment erase	Segment erase
0	-	1	0	Mass erase (main memory segments-selected array)	Mass erase (all main memory segments)
0	0	1	1	Erase all flash memory (main and information segments – selected array)	Erase main and information flash memory
1	-	1	0	Global mass erase (all main memory segments – both arrays)	Mass erase (all main memory segments)
1	1	1	1	Erase main memory and information segments- both arrays	Erase main flash memory

[1] This bit is only present in the MSP430FG461x devices
[2] This bit is only present in the MSP430F2xxx devices

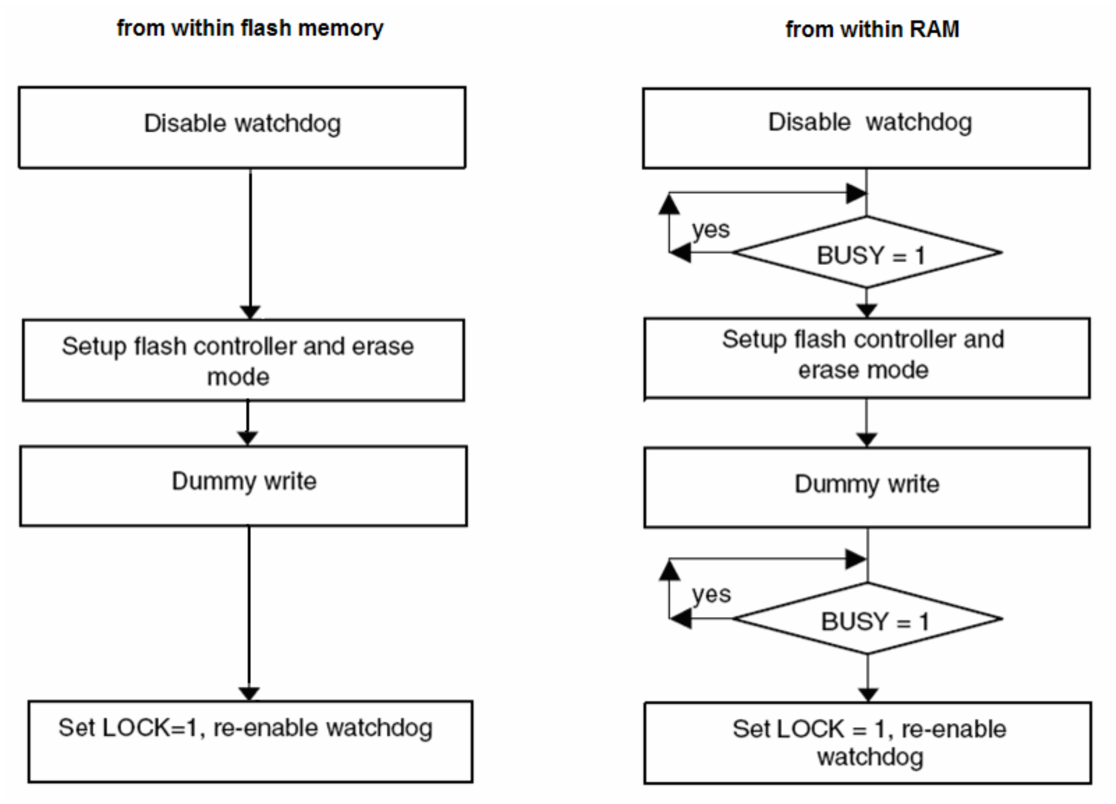
The erase mode procedure for both segment and mass erase are shown in *Figure 13-4*. The procedure for each one is as follows:

- Segment Erase:
  - Check BUSY = 0 (FCTL3 register);
  - LOCK = 0 (FCTL3 register);
  - ERASE = 1 (FCTL1 register);
  - Perform a dummy write to the segment to be erased (Any write, erase, or logical operation);
  - A segment erase requires approximately 5000 cycles of the timing generator (during this period BUSY = 1);
  - Wait for BUSY = 0 (FCTL3 register).
  - LOCK = 1 (FCTL3 register) to prevent accidental writes.

- Mass Erase (all main memory segments):
  - Similar to Segment erase;
  - Requires setting MERAS bit instead of ERASE bit in the FCTL1 register.

- All Erase (all segments):
  - Requires setting (GMERAS), MERAS and ERASE bits in the FCTL1 register.

Figure 13-4. Segment and mass erase modes procedure.


from within flash memory
Disable watchdog
Setup flash controller and erase mode
Dummy write
Set LOCK=1, re-enable watchdog
from within RAM
Disable watchdog
yes
BUSY = 1
Setup flash controller and erase mode
Dummy write
yes
BUSY = 1
Set LOCK = 1, re-enable watchdog

- ***Write modes***

A byte/word write cycle can be initiated from within flash memory or from RAM. However, a block write cycle cannot be initiated from within flash memory. The block write must be initiated from RAM.

For the byte/word write cycle, the characteristics of the cycle are identical to the erase cycle, configuring the required bits and instead of writing a dummy word, it writes the byte or word required.

The block write can be used to accelerate the flash write process (twice as fast as byte/word mode) when many sequential bytes or words need to be programmed.

The write modes are shown in *Table 13-2* (for MSP430FG461x and F2xxx devices).

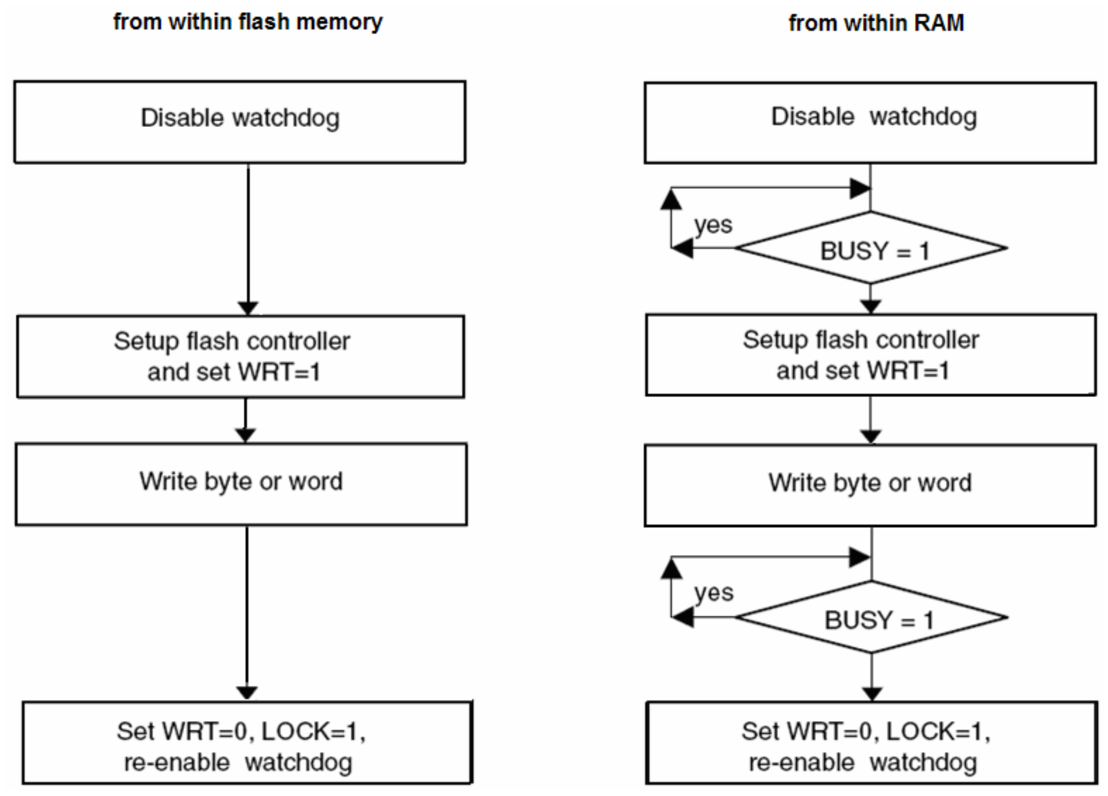
Table 13-2. Write modes.

Bits		Mode description
BLKWRT	WRT	MSP430FG461x and MSP430F2xxx
0	1	Byte/word write
1	1	Block write

The write mode procedure for byte/word write is shown in *Figure 13-5*. The procedure for this mode is as follows:

- Byte/word write:
  - Check BUSY = 0 (FCTL3 register);
  - LOCK = 0 (FCTL3 register);
  - WRT = 1 (FCTL1 register);
  - Write the element to the appropriate address (starts the timing generator);
  - A element write requires 33 cycles of the timing generator (during this period BUSY = 1);
  - Wait for BUSY = 0 (FCTL3 register).
  - LOCK = 1 (FCTL3 register) to prevent accidental writes.

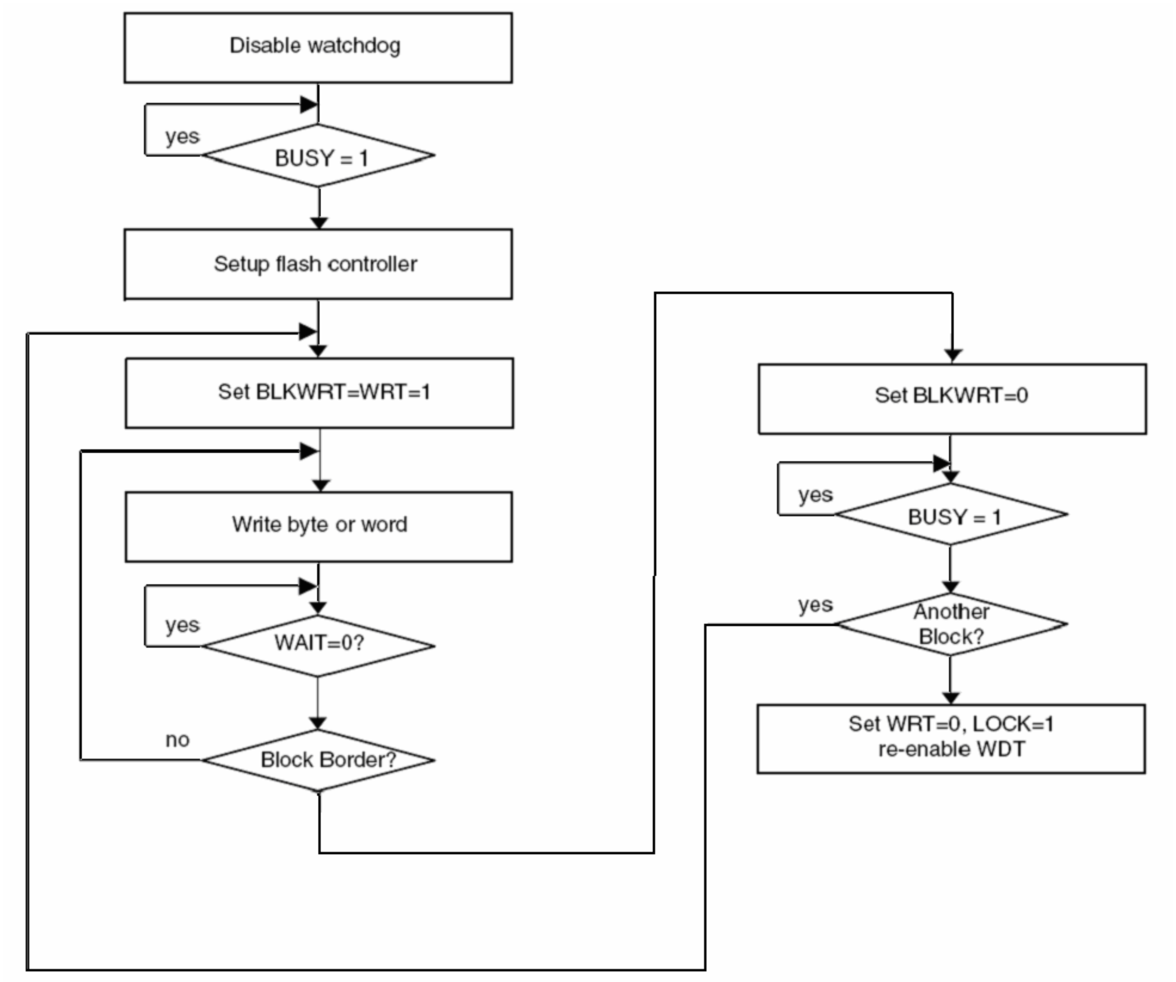
Figure 13-5. Byte/word write mode procedure.


from within flash memory
from within RAM
Disable watchdog
Setup flash controller and set WRT=1
Write byte or word
Set WRT=0, LOCK=1, re-enable watchdog
Disable watchdog
yes
BUSY = 1
Setup flash controller and set WRT=1
Write byte or word
yes
BUSY = 1
Set WRT=0, LOCK=1, re-enable watchdog

The write mode procedure for block write is shown in *Figure 13-6*. The procedure for this mode is as follows:

- Block (64-byte blocks) write:
  - Similar to a successive element write;
  - Check BUSY = 0 (FCTL3 register);
  - LOCK = 0 (FCTL3 register);
  - WRT = 1 and BLKWRT = 1 (FCTL1 register);
  - Write the byte or word in the block to the appropriate address (starts the timing generator);
  - Loop until WAIT = 1 (FCTL3 register).
  - Repeat write next byte or word of the block to the appropriate address until all elements of the current block have been written;
  - Set WRT = 0 and BLKWRT = 0 (FCTL1 register);
  - A block write requires 20 cycles of the Timing Generator per element, plus overhead of about 15 more cycles (during this period BUSY = 1);
  - Wait for BUSY = 0 (FCTL3 register).
  - LOCK = 1 (FCTL3 register) to prevent accidental writes.

Figure 13-6. Block write mode procedure.



❑ ***Flash memory access during write or erase***

While BUSY = 1, any write or any erase operation initiated from RAM or from within flash memory, triggers the conditions shown in *Table 13-3*.

Table 13-3. Flash access conditions.

Flash operation	Flash access	Wait	Result
Any erase Byte/word write	Read	0	ACCVIFG = 0 Value read: 03FFFh
	Write	0	ACCVIFG = 0 Write ignored
	Instruction fetch	0	ACCVIFG = 0 CPU fetch: 03FFFh
Block write	Any	0	ACCVIFG = 1 LOCK = 1
	Read	1	ACCVIFG = 0 Value read: 03FFFh
	Write	1	ACCVIFG = 0 Flash written
	Instruction fetch	1	ACCVIFG = 1 LOCK = 1

- ***Flash Memory Controller Interrupts***

The flash controller has two interrupt sources:

- KEYV = 1 (key violation flag):
  - Flash control registers are written with an incorrect password.
- ACCVIFG = 1(access violation flag):
  - When ACCVIE = 1 it generates an interrupt request;
  - Must be reset by software.

13.3 Registers

The flash memory control registers are shown below for the MSP430FG4618:

FCTL1, Flash Memory Control Register 1

15	14	13	12	11	10	9	8
(FCTLx password) Read: FRKEY = 096h Write (must be): FWKEY = 0A5h							

7	6	5	4	3	2	1	0
BLKWRT	WRT	Reserved	EEIEX[1]	EEI[1] GMERAS[2]	MERAS	ERASE	Reserved

[1] MSP430F2xx(x) family devices. Not present on MSP430F2013.

[2] MSP430FG461x devices.

Bit		Description
7	BLKWRT	Block write mode when BLKWRT = 1 (WRT must also be set)
6	WRT	Write when WRT = 1
4	EEIEX[1]	Enable Emergency Interrupt Exit when EEIEX = 1 and GIE = 1
3	EEI[1] GMERAS[2]	[1] Enable segment Erase to be interrupted by an interrupt request when EEI = 1 [2] Global mass erase (see *Table 13-1*)
2	MERAS	Mass erase (see *Table 13-1*)
1	ERASE	Erase (see *Table 13-1*)

FCTL2, Flash Memory Control Register 2

15	14	13	12	11	10	9	8
(FCTLx password) Read: FRKEY = 096h Write (must be): FWKEY = 0A5h							

7	6	5	4	3	2	1	0
FSSELX		FNx					

Bit		Description
7-6	FSSELx	Flash controller clock source: FSSEL1 FSSEL0 = 00 ⇒ ACLK FSSEL1 FSSEL0 = 01 ⇒ MCLK FSSEL1 FSSEL0 = 10 ⇒ SMCLK FSSEL1 FSSEL0 = 11 ⇒ SMCLK

5-0	FNx	Flash controller clock divider
		FNx=00h ⇒ /1
		...
		FNx=03Fh ⇒ /64

FCTL3, Flash Memory Control Register 3

15	14	13	12	11	10	9	8
(FCTLx password) Read: FRKEY = 096h Write (must be): FWKEY = 0A5h							

7	6	5	4	3	2	1	0
FAIL(1)	LOCKA(1)	EMEX	LOCK	WAIT	ACCVIFG	KEYV	BUSY

(1) MSP430F2xx(x) family devices.

Bit		Description
7	FAIL	Operation failure of the clock source, $f_{(FTG)}$, or a flash operation is aborted from an interrupt when EEIEX = 1 when FAIL = 1
6	LOCKA(1)	Segment A locked and all information memory is protected from erasure during a mass erase when LOCKA = 1
5	EMEX	Emergency exit when EMEX = 1
4	LOCK	Locks the flash memory for writing or erasing when LOCK = 1
3	WAIT	WAIT = 0 ⇒ while flash memory is being written to WAIT = 1 ⇒ when flash memory is ready for the next byte/word write
2	ACCVIFG	Access violation interrupt flag ACCVIFG = 1 when interrupt is pending
1	KEYV	Flash security key violation KEYV = 1 when FCTLx password was written incorrectly (not 0A5h)
0	BUSY	Flash timing generator is busy when BUSY = 1

13.4 Laboratory 9: Flash write/read operations

Overview

The TI MSP430 has an internal flash memory that can be used for data storage. Two different methods of writing to the flash memory are studied in this laboratory. The first method requires the CPU execution of the code resident in flash memory. The consequences of this procedure are discussed. In the second part of the laboratory, the flash write and erase operations are conducted with the CPU executing the code resident in RAM. The important details are highlighted.

13.4.1 Lab9a. Flash memory programming with the CPU executing the code from flash memory

Project files

❑ C source files:	**Chapter 13 > Lab9 > Lab9a_student.c**
	Chapter 13 > Lab9 > Lab9_a.c
	Chapter 13 > Lab9 > lnk_msp430fg4618.cmd
Solution file:	**Chapter 13 > Lab9 > Lab9a_solution.c**

A. Resources

This laboratory uses the flash memory controller. The operation of this device is monitored using a digital output port (P2.1).

The project must be compiled using the files ***Lab9_a.c*** and the command file ***lnk_msp430fg4618.cmd***.

The code is resident in the flash memory, so whenever a flash write or erase operation occurs, the CPU access to this memory is automatically inhibited.

B. Software application organization

The software begins by disabling the Watchdog Timer. Then, port P2.1 is set as an output with a logic low level.

The flash memory controller is configured with the clock MCLK divided by 3. Thus the f_{FTG} operating frequency lies within the specified limits of 257 kHz to 476 kHz.

A set of routines are provided to erase, write and copy the contents of a segment. The main tasks related to the flash memory handling are presented using this set of routines.

The information Segments A and B are erased first. Then, bytes are written to SegmentA and words are written to SegmentB. The contents of the information memory SegmentA are copied to the information SegmentB, overwriting the previous contents.

C. System configuration

❑ Flash memory controller configuration

Configure the register FCTL2 to use clock MCLK divided by 3. Do not forget to enter the password to access the register.

FCTL2 = ______________________;

❑ Segment erase routine

Configure the registers FCTL1 and FCTL3 in order to initiate the flash segment erase process by writing an address belonging to the segment to be erased. Be sure to include the password to access the register.

FCTL1 = ______________________;

FCTL3 = ______________________;

Block flash write and erase operations are carried out after erasing the segment:

FCTL3 = ______________________;

❑ Flash write routine

Configure the registers in order to start writing to the flash memory. Be sure to include the password to access the register.

FCTL1 = ______________________;

FCTL3 = ______________________;

Configure flash block write and erase operations and disable the write bit:

FCTL1 = ______________________;

FCTL3 = ______________________;

D. Analysis of the operation

❑ Execution time for the information segments erase operation

Put the cursor at line of code 124, located just after the second port P2.1 switching state. Execute the software until the cursor position is reached. The erase operation timing can be seen on an oscilloscope with the probe connected to pin 2 of the Header 4.

❑ Bytes write in the information memory A

The routine ***write_char_flash*** allows writing a byte to flash memory. It receives the memory address where the byte should be stored.

Open the ***memory*** window, and add the address of the information memory A. The content of this address becomes visible after ordering its ***rendering***. As we are writing a byte to flash, we must change the presentation of the memory contents. Choose the option ***Column Size 1***, from the context menu of the ***memory*** window, through the option ***Format***.

Now, during the execution of the ***for*** loop, the flash contents is written sequentially.

❑ Bytes written in the information B memory

This routine is similar to the previous one. Note that now the flash write address is increased by two because a word occupies two bytes of memory.

The information is more readily observed when the memory contents display mode is restored to its initial state. Reset the default conditions in the option ***Format*** of the context menu.

❑ Copy the contents of the information A memory to information B memory

The output port P2.1 is enabled before the copy process begins. The copy routine receives the start address of the source information segment and the start address of the destination information segment. The information is then successively read and written from one segment to another.

Port P2.1 is disabled at the end of the copy process. Thus, the task execution time can be measured using an oscilloscope.

MSP-EXP430FG4618 **SOLUTION**

Using the MSP-EXP430FG4618 Development Tool, implement flash memory programming with the CPU executing the code from flash memory.

❑ Flash memory controller configuration:

```
FCTL2 = FWKEY | FSSEL0 | FN1;
// MCLK/3 for Flash Timing Generator
```

❑ Segment erase routine:

```
FCTL1 = FWKEY | ERASE;                    // Set Erase bit
FCTL3 = FWKEY;                           // Clear Lock bit

//Flash block write and erase operations after erasing the
segment:
FCTL3 = FWKEY | LOCK;                      // Set LOCK bit
```

❑ Flash write routine:

```
FCTL1 = FWKEY | ERASE;                    // Set Erase bit
FCTL3 = FWKEY;                           // Clear Lock bit

// Flash block write and erase operations and disable the
write bit after the writing process to the segment:

FCTL3 = FWKEY | LOCK;                      // Set LOCK bit
```

13.4.2 Lab9b. Flash memory programming with the CPU executing the code in RAM

Project files

❑ C source files:	**Chapter 13 > Lab9 > Lab9b_student.c**
	Chapter 13 > Lab9 > Lab9_b.c
	Chapter 13 > Lab9 > lnk_msp430fg4618_RAM.cmd
Solution file:	**Chapter 13 > Lab9 > Lab9b_solution.c**

A. Resources

The tasks developed in the previous laboratory are executed again during this laboratory. The difference this time is that the software runs from RAM.

This process requires special procedures. The routines to run from RAM must be identified. The application must begin by copying the routines from flash to RAM.

The directive MEMORY determines the device's memory configuration. The memory can be organized in accordance with the system needs. This directive identifies the memory ranges that are physically present on the device. Each of these ranges has a set of features, such as:

- ❑ Name;
- ❑ Initial address;
- ❑ Length;
- ❑ Optional attributes set;
- ❑ Optional filling specifications.

The directive ***Memory*** is organized as described below.

MEMORY
{
name 1 [**(***attr***)**] **: origin =** *constant***, length =** *constant* [**, fill =** *constant*]
.
.
name n [**(***attr***)**] **: origin =** *constant***, length =** *constant* [**, fill =** *constant*]
}

The directive ***SECTIONS*** controls how the sections are built and reserved. The directive performs the following:

- ❑ Describes how the input sections are related to the output sections;
- ❑ Defines the output sections in the executable program;
- ❑ Defines where the output sections are placed in memory;
- ❑ Allows changing the name of the output sections;

The directive ***SECTIONS*** is organized as described below.

SECTIONS
{
name : [*property* [**, property**] [**, property**] . . .]
name : [*property* [**, property**] [**, property**] . . .]
name : [*property* [**, property**] [**, property**] . . .]
}

The following directives are possible:

Reserve memory space to load the section:

Syntax: **load =** *allocation* or
Allocation or
> *allocation*

Define the memory space where the code belonging to the section will run:

Syntax: **run =** *allocation* or
run > *allocation*

In this project, we intend to write the code to the flash memory, but we want it to be executed from RAM. The Linker offers a very simple way to accomplish this task. A memory space where the code is stored is associated with another memory space where it will run. The application transfers the code to the memory space, where it will be executed.

The memory spaces needed to store the routines are defined in the ***lnk_msp430fg4618_RAM.cmd*** file.

```
RAM_MEM              : origin = 0x1100, length = 0x0200
FLASH_MEM            : origin = 0x3100, length = 0x0200
```

The following sections are also defined:

```
.FLASHCODE            : load = FLASH_MEM, run = RAM_MEM
.RAMCODE              : load = FLASH_MEM
```

B. Software application organization

The software for this laboratory has the same structure as the previous one.

The directive ***#pragma CODE_SECTION (symbol, "section name")*** reserves space for the "***symbol***" in a section called "***section name***". Thus, the routines are stored in the section "***.FLASHCODE***".

The routine ***copy_flash_to_RAM*** runs from the beginning of the program. It is responsible for transferring the flash contents to RAM.

The files ***Lab9_b.c*** and ***lnk_msp430fg4618_RAM.cmd*** must be included during the compilation.

Now, the code is executed from RAM. Check, whenever appropriate, the Wait bit state of the register FCTL3.

C. System configuration

❑ Flash storage management routines

To store the flash management routines in the section "***.FLASHCODE***" complete the empty spaces:

```
#pragma CODE_SECTION(______________,____________)
void erase_segment(int address)

#pragma CODE_SECTION(______________,____________)
void write_char_flash(int address, char value)

#pragma CODE_SECTION(______________,____________)
void write_int_flash(int address, int value)

#pragma CODE_SECTION(______________,____________)
void copy_seg_flash(int address_source, int
address_destination)
```

❑ Check the flag wait

At software key points, and whenever writing or erasing the flash memory, perform a delay before proceeding with the data writes. Complete the following line of code in order to suspend the program flow while the busy flag is not active.

```
while(_____________);
```

D. Analysis of operation

Analyse the differences between the different versions of the routines. Note that successive delays are placed in the versions to be executed from RAM.

MSP-EXP430FG4618 **SOLUTION**

Using the MSP-EXP430FG4618 Development Tool, implement flash memory programming with the CPU executing the code from RAM.

- ❑ Flash management routines storage:

```
#pragma CODE_SECTION(erase_segment,".FLASHCODE")
void erase_segment(int address)

#pragma CODE_SECTION(write_char_flash,".FLASHCODE")
void write_char_flash(int address, char value)

#pragma CODE_SECTION(write_int_flash,".FLASHCODE")
void write_int_flash(int address, int value)

#pragma CODE_SECTION(copy_seg_flash,".FLASHCODE")
void copy_seg_flash(int address_source, int address_destination)
```

- ❑ Check the busy flag:

```
while(FCTL3&BUSY);                    // Check BUSY flag
```

13.5 Quiz

1. The features of flash memory are:

(a) Low cost;

(b) Fast to read from;

(c) Non-volatile;

(d) All of above.

2. The timing generator of the MSP430 flash memory can be sourced by:

(a) ACLK;

(b) SMCLK;

(c) MCLK;

(d) All of above.

3. A MSP430 flash device can be programmed via:

(a) JTAG interface;

(b) Bootstrap loader;

(c) Custom software solution;

(d) All of above.

4. Flash memory is partitioned into:

(a) Two sections;

(b) Three sections;

(c) Four sections;

(d) None of above.

5. When LOCKA is set:

(a) SegmentA can be written and erased;

(b) SegmentA cannot be written or erased;

(c) All information memory is protected from erasure during a mass erase or production programming;

(d) All information memory is erased during a mass erase or production programming.

6. When EMEX is set:

(a) Write operation begins;

(b) Erase operation begins;

(c) All flash operations cease;

(d) None of above.

7. An erase cycle can be initiated from:

(a) Within flash memory;

(b) From RAM;

(c) Within flash memory or from RAM;

(d) Simultaneously within flash memory and from RAM.

8. For the MSP430FG4618 when GMERAS and MERAS bits are set:

(a) A segment is erased;

(b) All main memory is erased – selected array;

(c) All information memory is erased – selected array;

(d) All flash memory is erased – both arrays.

9. When BLKWRT and WRT are set:

(a) A byte/word write cycle is initiated within flash memory;

(b) A block write cycle is initiated from RAM;

(c) A block write cycle is initiated within flash memory;

(d) A byte/word write cycle is initiated from RAM.

10. When reading within flash memory while BUSY = 1:

(a) ACCVIFG = 1 and the value read is 03FFFh;

(b) ACCVIFG = 0 and the value read is 03FFFh;

(c) ACCVIFG = 1 and LOCK = 1;

(d) None of above.

11. The 16 bits of the FCTL1 flash memory controller control register must have:

(a) All its high-byte bits set to 0;

(b) A 0x096h password in the high-byte to read from flash memory and a 0x0A5h password must be written in the high byte to write to the flash memory;

(c) A 0x05Ah password in the high-byte to read and write to the flash memory;

(d) All its high-byte bits set to 1.

12. When the FAIL bit of the control register is set:

(a) The clock source has failed;

(b) The flash operation was aborted;

(c) An access violation occurred;

(d) The flash security key was incorrect.

Solution: 1. (d); 2. (d); 3. (d); 4. (a); 5. (b); 6. (c); 7. (c); 8. (d); 9. (b); 10. (a); 11. (b); 12. (a)

13.6 FAQs

1. What must be the minimum voltage during a flash write or erase operation?

The minimum V_{CC} voltage is between 2.2 V and 2.7 V to avoid unpredictable results.